Міністерство освіти і науки України
Міжнародний гуманітарний університет
Кафедра інформаційних технологій

С.Б. Приходько

ПАТЕРНИ ТА ФРЕЙМВОРКИ

Методичні вказівки до практичних робіт
для здобувачів вищої освіти,
які навчаються за спеціальностями
галузі «Інформаційні технології»

Одеса 2025

Затверджено Вченою Радою Міжнародного гуманітарного університету.
Протокол № 5 від 20 січня 2025 р.

С.Б. Приходько Патерни та фреймвокри. Методичні вказівки до практичних робіт для здобувачів вищої освіти, які навчаються за спеціальностями галузі «Інформаційні технології» [Електронне видання]. Кафедра інформаційних технологій Міжнародного гуманітарного університету. Одеса, 2025. 84 с.

Дисципліна «Патерни та фреймворки» входить до завершального етапу підготовки здобувачів вищої освіти, які займаються створенням програмного забезпечення. Вона спрямована на формування системного уявлення про сучасні підходи до проєктування та реалізації програмних систем. У межах курсу розглядаються принципи повторного використання програмних рішень, патерни проєктування програмного забезпечення та типові архітектури програмних систем. Значна увага приділяється аналізу породжувальних, структурних і поведінкових патернів, їх призначенню та особливостям застосування під час створення гнучких і модульних програмних систем. Окремий розділ дисципліни присвячений використанню фреймворків під час створення програмного забезпечення. Розглядаються найбільш поширені типи фреймворків, зокрема сучасні фреймворки для розробки застосунків для персональних комп’ютерів, мобільних та вебзастосунків. Особлива увага приділяється принципам їх вибору та практичного використання.

ЗМІСТ

ЧАСТИНА 1

ПАТЕРНИ ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ТЕМА 1. ПАТЕРНИ У ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ

Практична робота 1. Аналіз поняття патерну у програмному забезпеченні

Ключові положення

Патерн у програмному забезпеченні є узагальненим типовим рішенням, що застосовується для розв'язання повторюваних задач проєктування програмних систем. Він описує підхід до організації коду, взаємодії об'єктів і структури системи без прив'язки до конкретної реалізації або мови програмування. Основною ідеєю патернів є формалізація накопиченого досвіду розробників у вигляді перевірених рішень, які можуть бути повторно використані у різних програмних проєктах.

Патерни не є готовими програмними модулями або бібліотеками. Вони визначають концепцію побудови рішення, яку розробник реалізує самостійно залежно від контексту задачі. Таким чином, один і той самий патерн може мати різні реалізації у різних мовах програмування або фреймворках.

Кожен патерн має усталену структуру опису, що містить назву, призначення, область застосування, опис проблеми, яку він вирішує, а також спосіб її розв'язання. Опис також може містити приклади використання, переваги, недоліки та наслідки застосування патерну. Такий підхід дає змогу стандартизувати процес проєктування та полегшує комунікацію між розробниками.

Використання патернів сприяє підвищенню якості програмного забезпечення за рахунок зменшення складності коду, покращення його структури та забезпечення можливості подальшого розширення системи. Патерни дають змогу зменшити зв'язаність між компонентами та підвищити їх повторне використання. Це особливо важливо при розробленні великих програмних систем, де необхідно забезпечити масштабованість та підтримуваність коду.

Існує декілька класифікацій патернів, проте найбільш поширеною є класифікація, що поділяє їх на породжувальні, структурні та поведінкові. Породжувальні патерни визначають способи створення об'єктів, структурні – способи організації класів і об'єктів, поведінкові – механізми взаємодії між об'єктами. Такий поділ дає змогу систематизувати підходи до проєктування та полегшує вибір відповідного рішення для конкретної задачі.

Поняття патерну тісно пов'язане з принципами об'єктно-орієнтованого програмування, зокрема інкапсуляцією, спадкуванням та поліморфізмом. Багато патернів базуються саме на цих принципах і дають змогу ефективно їх застосовувати у практичній діяльності. Крім того, патерни часто використовуються разом із принципами SOLID, що забезпечують гнучкість та масштабованість програмних систем.

Важливим аспектом використання патернів є розуміння доцільності їх застосування. Надмірне використання патернів може ускладнювати програмний код та знижувати його зрозумілість. Тому вибір патерну повинен бути обґрунтованим і відповідати конкретній задачі. Розробник повинен аналізувати умови задачі, визначати проблеми проєктування та обирати ті патерни, які забезпечують оптимальне рішення.

Таким чином, патерни проєктування є важливим інструментом інженерії програмного забезпечення, що дає змогу формалізувати досвід розроблення, підвищити якість програмних

систем та забезпечити ефективну взаємодію між розробниками у процесі створення програмного продукту.

Практичне завдання

1. Дати визначення поняття патерну у програмному забезпеченні.
2. Проаналізувати роль патернів у процесі розроблення програмних систем.
3. Розглянути приклади застосування патернів у сучасних програмних продуктах або фреймворках.
4. Визначити основні складові опису патерну.
5. Проаналізувати класифікацію патернів проєктування.
6. Пояснити відмінність між патерном та готовим програмним рішенням.
7. Навести приклад реалізації одного простого патерну (наприклад, Singleton або Factory Method) у вибраній мові програмування.
8. Сформулювати висновки щодо доцільності використання патернів у програмному забезпеченні.

Методичні вказівки до виконання практичного завдання

Перед виконанням роботи необхідно ознайомитися з теоретичними відомостями щодо поняття патернів проєктування, їх класифікації та ролі у розробленні програмного забезпечення. Особливу увагу слід приділити розумінню того, що патерн є концептуальним рішенням, а не конкретною реалізацією.

На першому етапі необхідно сформулювати визначення поняття патерну. Доцільно розглянути декілька джерел та узагальнити інформацію, виділивши ключові ознаки патернів, такі як повторюваність, універсальність та незалежність від конкретної реалізації.

Далі слід проаналізувати роль патернів у процесі розроблення програмного забезпечення. Необхідно визначити, які проблеми вони дозволяють вирішувати, та пояснити, як їх використання впливає на структуру програмної системи.

На наступному етапі потрібно розглянути приклади застосування патернів у сучасних програмних продуктах або фреймворках. Для цього можна використати відкриті джерела інформації або документацію фреймворків, що застосовуються у курсі. Важливо встановити зв’язок між теоретичними положеннями та їх практичною реалізацією.

Окрему увагу необхідно приділити структурі опису патерну. Слід визначити основні елементи опису, такі як назва, призначення, проблема, рішення та наслідки застосування. Це дає змогу краще зрозуміти логіку побудови патернів.

Під час виконання практичної частини потрібно реалізувати приклад одного з простих патернів. Реалізація повинна демонструвати основну ідею патерну та пояснювати, яку проблему він вирішує. Код має бути структурованим та зрозумілим.

Після виконання всіх етапів необхідно сформулювати висновки. У висновках слід оцінити роль патернів у програмному забезпеченні, їх переваги та обмеження, а також доцільність використання у різних типах програмних систем.

Під час виконання роботи необхідно дотримуватися принципів академічної доброчесності, використовувати достовірні джерела інформації та коректно оформлювати посилання.

Контрольні питання

1. Що таке патерн у програмному забезпеченні?
2. Які основні ознаки патернів проєктування?
3. У чому полягає відмінність між патерном і готовим програмним рішенням?
4. Яку роль відіграють патерни у процесі розроблення програмних систем?
5. Які основні категорії патернів проєктування існують?
6. Що містить типовий опис патерну?
7. Які переваги дає використання патернів у програмному забезпеченні?
8. Які можливі недоліки використання патернів?
9. Яким чином патерни пов’язані з принципами об’єктно-орієнтованого програмування?
10. У яких випадках використання патернів є недоцільним?

Зміст звіту

Звіт з практичної роботи повинен містити такі структурні елементи:
– назву практичної роботи;
– мету виконання практичної роботи;
– визначення поняття патерну у програмному забезпеченні;
– аналіз ролі патернів у розробленні програмних систем;
– опис класифікації патернів;
– характеристику структури опису патерну;
– приклад реалізації патерну з поясненням;
– висновки за результатами виконання практичної роботи;
– перелік використаних джерел інформації.

Практична робота 2. Класифікація патернів програмного забезпечення

Ключові положення

Класифікація патернів програмного забезпечення є важливим елементом систематизації знань у галузі проєктування програмних систем. Вона дає змогу впорядкувати різноманітні підходи до розв’язання типових задач та полегшує вибір відповідного патерну залежно від характеру проблеми. Класифікація базується на функціональному призначенні патернів та їх ролі у структурі програмної системи.

Найбільш поширеною є класифікація, запропонована у межах підходу Gang of Four, відповідно до якої патерни поділяються на три основні групи: породжувальні, структурні та поведінкові. Кожна з цих груп відображає окремий аспект проєктування програмного забезпечення та відповідає за вирішення певного класу задач.

Породжувальні патерни визначають способи створення об’єктів і забезпечують контроль над процесом їх ініціалізації. Вони дають змогу відокремити процес створення об’єкта від його використання, що підвищує гнучкість системи. До цієї групи належать Singleton, Factory Method, Abstract Factory, Builder та Prototype. Основна ідея полягає у тому,

щоб приховати деталі створення об’єктів та забезпечити можливість зміни способу їх створення без впливу на інші частини системи.

Структурні патерни описують способи організації класів і об’єктів у більш складні структури. Вони визначають, яким чином компоненти системи поєднуються між собою та як забезпечується їх взаємодія. До структурних патернів належать Adapter, Bridge, Composite, Decorator, Facade, Flyweight та Proxy. Основною метою цієї групи є спрощення структури системи та зменшення зв’язаності між її компонентами.

Поведінкові патерни визначають алгоритми взаємодії між об’єктами та розподіл відповідальності між ними. Вони описують способи організації обміну даними та управління поведінкою об’єктів. До поведінкових патернів належать Chain of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method та Visitor. Використання цих патернів дає змогу забезпечити гнучкість поведінки системи та спростити її модифікацію.

Крім базової класифікації, існують також інші підходи до систематизації патернів. Зокрема, виділяють архітектурні патерни, що визначають загальну структуру програмної системи, та патерни інтеграції, що описують взаємодію між підсистемами. До архітектурних патернів належать Model-View-Controller, Layered Architecture, Microservices та інші.

Класифікація патернів має практичне значення, оскільки дає змогу швидко орієнтуватися у великій кількості можливих рішень. Розробник, розуміючи характер задачі, може визначити, до якої групи належить відповідний патерн, і обрати найбільш доцільний варіант. Це сприяє підвищенню ефективності процесу проєктування та якості програмного забезпечення.

Важливим аспектом є те, що один і той самий патерн може застосовуватися у різних контекстах і поєднуватися з іншими патернами. У складних програмних системах часто використовується комбінація декількох патернів, що забезпечує оптимальну структуру та поведінку системи.

Таким чином, класифікація патернів програмного забезпечення є основою для їх ефективного використання у практиці розроблення програмних систем, забезпечує системний підхід до проєктування та сприяє підвищенню якості програмного продукту.

Практичне завдання

1. Ознайомитися з підходами до класифікації патернів програмного забезпечення.
2. Описати основні групи патернів: породжувальні, структурні та поведінкові.
3. Навести приклади патернів, що належать до кожної групи.
4. Проаналізувати призначення кожної групи патернів.
5. Розглянути приклади застосування патернів у сучасних програмних системах.
6. Дослідити архітектурні патерни та їх особливості.
7. Порівняти різні підходи до класифікації патернів.
8. Сформулювати висновки щодо практичного значення класифікації патернів.

Методичні вказівки до виконання практичного завдання

Перед виконанням роботи необхідно повторити теоретичний матеріал щодо поняття патернів та ознайомитися з їх класифікацією. Особливу увагу слід приділити розумінню відмінностей між основними групами патернів та їх призначенням.

На першому етапі необхідно розглянути загальні підходи до класифікації патернів. Доцільно проаналізувати класифікацію Gang of Four як базову модель, що використовується у більшості сучасних джерел.

Далі слід детально описати кожну з основних груп патернів. Для кожної групи необхідно визначити її призначення, характерні особливості та типові задачі, які вона дозволяє вирішувати.

На наступному етапі потрібно навести приклади патернів для кожної групи та коротко охарактеризувати їх. Важливо показати, як саме ці патерни застосовуються у практиці розроблення програмного забезпечення.

Окрему увагу слід приділити архітектурним патернам. Необхідно пояснити їх роль у побудові програмних систем та навести приклади їх використання у сучасних фреймворках і програмних платформах.

Під час виконання практичного завдання доцільно використовувати приклади з реальних програмних систем або фреймворків, що застосовуються у курсі. Це дає змогу поєднати теоретичні знання з практичними навичками.

Після завершення аналізу необхідно сформулювати висновки. У висновках слід оцінити значення класифікації патернів для процесу проєктування програмного забезпечення та визначити її роль у практичній діяльності розробника.

Контрольні питання

1. Що таке класифікація патернів програмного забезпечення?
2. Які основні групи патернів виділяють у класифікації Gang of Four?
3. Яке призначення породжувальних патернів?
4. Яке призначення структурних патернів?
5. Яке призначення поведінкових патернів?
6. Які приклади патернів належать до породжувальних?
7. Які приклади патернів належать до структурних?
8. Які приклади патернів належать до поведінкових?
9. Що таке архітектурні патерни?
10. Чому класифікація патернів є важливою для розробника програмного забезпечення?

Зміст звіту

Звіт з практичної роботи повинен містити такі структурні елементи:
– назву практичної роботи;
– мету виконання практичної роботи;
– опис підходів до класифікації патернів;
– характеристику основних груп патернів;
– приклади патернів для кожної групи;
– аналіз архітектурних патернів;
– порівняння підходів до класифікації;
– висновки за результатами виконання практичної роботи;
– перелік використаних джерел інформації.

Практична робота 3. Аналіз структури опису патерну

Ключові положення

Опис патерну проєктування є формалізованим поданням типового рішення, що дає змогу стандартизувати підходи до розроблення програмного забезпечення та забезпечити єдине розуміння патернів серед розробників. Чітка структура опису патерну забезпечує можливість швидкого аналізу, порівняння та вибору відповідного рішення для конкретної задачі.

Класичний підхід до опису патернів передбачає використання уніфікованої структури, яка була сформована у межах підходу Gang of Four. Така структура містить набір обов'язкових елементів, кожен з яких виконує певну функцію у поясненні сутності патерну та умов його застосування.

Першим елементом опису є назва патерну. Назва повинна бути короткою, однозначною та відображати основну ідею рішення. Використання усталених назв забезпечує ефективну комунікацію між розробниками та дає змогу швидко ідентифікувати патерн у професійному середовищі.

Другим важливим елементом є призначення патерну. У цьому розділі коротко описується, яку проблему вирішує патерн та у яких умовах його доцільно застосовувати. Призначення дає змогу швидко оцінити релевантність патерну до конкретної задачі.

Опис проблеми є ключовим елементом структури патерну. У цьому розділі формулюється ситуація, у якій виникає необхідність застосування патерну, визначаються обмеження та умови задачі. Чітке формулювання проблеми дає змогу краще зрозуміти контекст використання патерну.

Наступним елементом є рішення. У цьому розділі описується загальний підхід до розв'язання проблеми, визначається структура класів і об'єктів, а також принципи їх взаємодії. Рішення не містить конкретної реалізації, а описує абстрактну модель, яку можна адаптувати до різних мов програмування та платформ.

До важливих елементів також належить розділ учасників (participants), у якому визначаються основні класи або об'єкти, що беруть участь у реалізації патерну, та їх ролі у системі. Це дає змогу зрозуміти структуру взаємодії компонентів.

Розділ взаємодії (collaborations) описує, яким чином учасники патерну взаємодіють між собою у процесі виконання задачі. У цьому розділі розглядаються послідовності викликів, обмін даними та координація дій між об'єктами.

Результати (consequences) відображають переваги та недоліки використання патерну, а також вплив його застосування на структуру та поведінку програмної системи. У цьому розділі аналізуються компроміси, пов'язані із застосуванням патерну.

Додатковими елементами опису можуть бути приклади реалізації, варіанти застосування, а також пов'язані патерни. Приклади реалізації дають змогу краще зрозуміти практичне застосування патерну, а інформація про пов'язані патерни допомагає визначити альтернативні підходи до розв'язання задачі.

Аналіз структури опису патерну є важливим для формування навичок роботи з патернами. Розробник повинен вміти читати та інтерпретувати опис патерну, визначати його ключові елементи та оцінювати доцільність використання у конкретній ситуації. Це дає змогу ефективно застосовувати патерни у процесі розроблення програмного забезпечення.

Таким чином, структура опису патерну забезпечує систематизоване подання знань про типові рішення у програмній інженерії та є важливим інструментом для їх практичного використання.

Практичне завдання

1. Ознайомитися зі структурою опису патернів проєктування.
2. Визначити основні елементи опису патерну.
3. Обрати один патерн проєктування (наприклад, Singleton, Factory Method або Observer).
4. Проаналізувати структуру опису обраного патерну.
5. Описати кожен елемент структури патерну (назва, призначення, проблема, рішення, учасники, взаємодія, результати).
6. Навести приклад реалізації обраного патерну.
7. Проаналізувати переваги та недоліки використання патерну.
8. Сформулювати висновки щодо структури опису патернів.

Методичні вказівки до виконання практичного завдання

Перед виконанням роботи необхідно ознайомитися з теоретичним матеріалом щодо структури опису патернів проєктування. Особливу увагу слід приділити стандартному підходу до опису патернів, що використовується у класичній літературі.

На першому етапі необхідно визначити основні елементи структури опису патерну. Доцільно скласти перелік цих елементів та коротко охарактеризувати їх призначення.

Далі слід обрати конкретний патерн для аналізу. Рекомендується обирати прості патерни, що мають зрозумілу структуру та широко використовуються у практиці розроблення програмного забезпечення.

Після вибору патерну необхідно виконати його детальний аналіз. Для кожного елемента структури слід надати опис та пояснення його ролі у загальній моделі патерну. Важливо зберігати логічну послідовність викладу та чітко розмежовувати окремі елементи.

Особливу увагу слід приділити аналізу взаємодії між учасниками патерну. Необхідно пояснити, яким чином об’єкти обмінюються інформацією та як координується їх робота у межах реалізації патерну.

Під час виконання практичної частини необхідно навести приклад реалізації патерну. Реалізація повинна демонструвати основні ідеї патерну та відповідати описаній структурі. Код має бути зрозумілим та структурованим.

Після виконання роботи необхідно сформулювати висновки. У висновках слід оцінити значення структури опису патернів для їх розуміння та практичного застосування.

Контрольні питання

1. Що таке структура опису патерну?
2. Які основні елементи містить опис патерну?
3. Яке призначення розділу "проблема" у описі патерну?
4. Що описується у розділі "рішення"?
5. Хто є учасниками патерну?

6. Що таке взаємодія у контексті патерну?

7. Які аспекти відображаються у розділі "результати"?

8. Чому важливо використовувати уніфіковану структуру опису патернів?

9. Яку роль відіграють приклади реалізації у розумінні патерну?

10. Як структура опису патерну допомагає у процесі розроблення програмного забезпечення?

Зміст звіту

Звіт з практичної роботи повинен містити такі структурні елементи:

– назву практичної роботи;

– мету виконання практичної роботи;

– перелік елементів структури опису патерну;

– аналіз обраного патерну відповідно до структури опису;

– приклад реалізації патерну;

– аналіз переваг та недоліків патерну;

– висновки за результатами виконання практичної роботи;

– перелік використаних джерел інформації.

Практична робота 4. Аналіз прикладів застосування патернів у програмних системах

Ключові положення

Патерни проєктування набувають практичного значення лише у контексті їх застосування у реальних програмних системах. Аналіз прикладів використання патернів дає змогу зрозуміти, яким чином абстрактні підходи до проєктування реалізуються у конкретних програмних рішеннях, а також оцінити їх ефективність у різних умовах.

У сучасних програмних системах патерни застосовуються на різних рівнях – від окремих класів до архітектури всієї системи. Вони можуть використовуватися як у невеликих програмних модулях, так і у складних розподілених системах. Застосування патернів забезпечує повторне використання рішень, підвищує якість коду та спрощує процес його супроводження.

Одним із найбільш поширених прикладів є використання патерну Singleton, який забезпечує існування лише одного екземпляра класу. Такий підхід часто використовується для організації доступу до спільних ресурсів, зокрема конфігураційних даних або логування. У багатьох фреймворках Singleton застосовується для реалізації сервісів, що повинні мати єдину точку доступу.

Патерн Factory Method широко використовується у фреймворках для створення об’єктів без прив’язки до конкретних класів. Це дає змогу розширювати систему без зміни існуючого коду. Наприклад, у вебфреймворках створення об’єктів обробників запитів часто реалізується із використанням цього патерну.

Патерн Observer застосовується для організації механізму сповіщення між об’єктами. Він широко використовується у системах із подієвою моделлю, зокрема у графічних

інтерфейсах користувача та вебзастосунках. Завдяки цьому патерну забезпечується автоматичне оновлення стану об’єктів при зміні даних.

Патерн Model-View-Controller є прикладом архітектурного підходу, що широко використовується у вебзастосунках. Він передбачає розділення системи на три основні компоненти: модель, що відповідає за дані, представлення, що відповідає за інтерфейс користувача, та контролер, що забезпечує взаємодію між ними. Такий підхід дає змогу підвищити модульність системи та спростити її супроводження.

У фреймворках, що використовуються у практичних роботах, патерни є невід’ємною частиною їх архітектури. Наприклад, у Flask та Django реалізовано патерни маршрутизації, контролерів та шаблонів представлення. У Qt використовується патерн сигналів і слотів, що має ознаки Observer. У .NET широко застосовуються патерни Dependency Injection та Repository.

Аналіз прикладів застосування патернів дає змогу встановити зв’язок між теоретичними знаннями та практичними навичками. Розробник повинен не лише знати класифікацію патернів, але й розуміти, де і як вони застосовуються у реальних системах. Це дає змогу ефективно використовувати патерни у власних програмних проєктах.

Таким чином, вивчення прикладів застосування патернів є необхідним етапом підготовки фахівців з інженерії програмного забезпечення, оскільки забезпечує формування практичних навичок проєктування програмних систем.

Практичне завдання

1. Ознайомитися з прикладами застосування патернів у програмних системах.
2. Обрати один або декілька патернів проєктування для аналізу.
3. Розглянути приклади реалізації обраних патернів у сучасних фреймворках або програмних продуктах.
4. Проаналізувати, яку проблему вирішує кожен обраний патерн.
5. Описати структуру реалізації патерну у конкретній програмній системі.
6. Визначити переваги використання патерну у розглянутому прикладі.
7. Проаналізувати можливі недоліки або обмеження застосування патерну.
8. Сформулювати висновки щодо ефективності використання патернів у програмних системах.

Методичні вказівки до виконання практичного завдання

Перед виконанням роботи необхідно повторити теоретичний матеріал щодо основних патернів проєктування та їх класифікації. Особливу увагу слід приділити розумінню призначення патернів та умов їх застосування.

На першому етапі необхідно обрати патерни для аналізу. Доцільно використовувати ті патерни, які широко застосовуються у сучасних програмних системах і підтримуються фреймворками, що використовуються у курсі.

Далі слід знайти приклади реалізації обраних патернів у програмних продуктах або фреймворках. Джерелами інформації можуть бути офіційна документація, технічні статті або відкриті репозиторії програмного коду.

Після вибору прикладів необхідно виконати їх аналіз. Слід визначити, яку проблему вирішує патерн, як організована структура компонентів та яким чином реалізується взаємодія між ними.

Особливу увагу слід приділити відповідності між теоретичним описом патерну та його практичною реалізацією. Необхідно визначити, які елементи патерну збережені у реалізації, а які адаптовані під конкретні умови.

У процесі аналізу доцільно використовувати приклади коду або структурні схеми, що ілюструють роботу патерну. Це дає змогу краще зрозуміти принципи його застосування.

Після виконання аналізу необхідно сформулювати висновки. У висновках слід оцінити ефективність використання патернів, їх вплив на структуру програмної системи та доцільність застосування у різних умовах.

Контрольні питання

1. Чому важливим є аналіз прикладів застосування патернів?
2. У яких програмних системах використовуються патерни проєктування?
3. Які патерни широко застосовуються у сучасних фреймворках?
4. Яку роль відіграє патерн Singleton у програмних системах?
5. У чому полягає призначення патерну Factory Method?
6. Як реалізується патерн Observer у програмних системах?
7. Що таке архітектурний патерн Model-View-Controller?
8. Які переваги дає використання патернів у програмних системах?
9. Які можливі недоліки застосування патернів?
10. Як оцінити доцільність використання патерну у конкретній задачі?

Зміст звіту

Звіт з практичної роботи повинен містити такі структурні елементи:
– назву практичної роботи;
– мету виконання практичної роботи;
– опис обраних патернів;
– приклади їх застосування у програмних системах;
– аналіз структури реалізації патернів;
– оцінку переваг та недоліків використання;
– висновки за результатами виконання практичної роботи;
– перелік використаних джерел інформації.

ТЕМА 2. ТИПОВІ АРХІТЕКТУРИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Практична робота 5. Аналіз архітектури Model–View–Controller

Ключові положення

Архітектура Model–View–Controller є одним із найбільш відомих підходів до побудови програмних систем, у яких необхідно розділити логіку оброблення даних, подання інформації та керування взаємодією з користувачем. Цей підхід дає змогу зменшити зв’язаність між окремими частинами програмної системи, полегшити її супроводження та забезпечити зручність подальшого розширення. Архітектура Model–View–Controller широко застосовується у вебзастосунках, настільних програмах та інших програмних системах, де важливим є чіткий розподіл відповідальності між компонентами.

Компонент Model відповідає за дані, правила їх оброблення та бізнес-логіку системи. Саме модель визначає, які дані використовуються у програмі, яким чином вони зберігаються, змінюються та перевіряються. У межах цього компонента може виконуватися взаємодія з базами даних, файлами, зовнішніми сервісами або іншими джерелами інформації. Model не повинна залежати від конкретного способу відображення даних, оскільки її головне призначення полягає у забезпеченні коректної роботи із предметною областю задачі.

Компонент View відповідає за подання даних користувачу. Його завданням є відображення інформації у зрозумілому та зручному вигляді. У вебзастосунках представлення може бути реалізоване у вигляді HTML-сторінок, шаблонів або інтерфейсних компонентів. У настільних програмах View подається у вигляді графічного інтерфейсу, вікон, форм, кнопок, списків та інших елементів. View не повинна містити складної логіки оброблення даних, оскільки її основна функція полягає у відображенні стану системи.

Компонент Controller забезпечує зв’язок між Model і View. Він приймає дії користувача, обробляє запити, викликає відповідні операції моделі та визначає, яке представлення повинно бути використане для відображення результату. Таким чином, Controller виконує координувальну функцію та забезпечує керування потоком виконання програми. У вебзастосунках контролер зазвичай отримує HTTP-запит, виконує необхідну обробку та повертає відповідь у вигляді сторінки або даних.

Основною перевагою архітектури Model–View–Controller є чіткий розподіл відповідальності. Завдяки цьому зміни у способі подання даних не потребують зміни бізнес-логіки, а модифікація моделі не вимагає повної переробки інтерфейсу користувача. Такий підхід дає змогу організувати код більш структуровано та забезпечує можливість паралельної роботи різних учасників команди над окремими частинами системи.

Ще однією перевагою є підвищення повторного використання компонентів. Одна й та сама модель може використовуватися з різними представленнями, а окремі контролери можуть забезпечувати різні сценарії взаємодії з користувачем. Це особливо важливо у великих програмних системах, де необхідно забезпечити масштабованість і підтримуваність програмного коду.

Разом із перевагами архітектура Model–View–Controller має і певні обмеження. У простих програмах її впровадження може бути надмірним і призводити до зайвого ускладнення структури. Крім того, у великих проєктах при недостатньо чіткому проєктуванні контролери можуть ставати перевантаженими логікою, що порушує початкову

ідею розподілу відповідальності. Тому використання цього підходу повинно бути обґрунтованим і відповідати складності програмної системи.

Архітектура Model–View–Controller є основою багатьох сучасних фреймворків. Вебфреймворки Django, Laravel, Ruby on Rails, ASP.NET MVC та інші використовують подібний підхід до організації програмного коду. У цих системах маршрутизація запитів, оброблення даних та формування представлення будуються відповідно до принципів розподілу на модель, представлення і контролер. Це підтверджує практичну значущість архітектури та її широке застосування у сучасній інженерії програмного забезпечення.

Аналіз архітектури Model–View–Controller дає змогу зрозуміти основи побудови багатьох сучасних програмних систем. Вивчення цього підходу є необхідним для формування у здобувачів уміння проєктувати структуровані, зрозумілі та підтримувані програмні застосунки.

Практичне завдання

1. Ознайомитися з основними принципами архітектури Model–View–Controller.
2. Визначити призначення компонентів Model, View і Controller.
3. Проаналізувати взаємодію між компонентами архітектури.
4. Розглянути приклад програмної системи або фреймворку, у якому використовується Model–View–Controller.
5. Визначити, які елементи прикладу відповідають моделі, представленню та контролеру.
6. Описати переваги використання архітектури Model–View–Controller.
7. Проаналізувати можливі недоліки або обмеження цього підходу.
8. Сформулювати висновки щодо доцільності використання архітектури Model–View–Controller у програмних системах.

Методичні вказівки до виконання практичного завдання

Перед виконанням практичної роботи необхідно ознайомитися з теоретичним матеріалом щодо архітектурних патернів програмного забезпечення та ролі архітектури Model–View–Controller у побудові програмних систем. Особливу увагу слід приділити розумінню функцій кожного з трьох основних компонентів цієї архітектури.

На першому етапі необхідно визначити сутність архітектури Model–View–Controller та коротко охарактеризувати її призначення. Доцільно показати, що цей підхід використовується для розмежування оброблення даних, відображення інформації та керування взаємодією з користувачем.

Далі потрібно окремо проаналізувати кожен компонент архітектури. Для Model слід визначити, які дані та правила оброблення вона містить. Для View необхідно описати спосіб подання інформації користувачу. Для Controller потрібно пояснити, як він приймає дії користувача та координує роботу системи.

На наступному етапі доцільно розглянути приклад реального застосування архітектури Model–View–Controller. Це може бути вебзастосунок, настільна програма або приклад із сучасного фреймворку. У межах аналізу слід встановити відповідність між конкретними елементами програмної системи та компонентами архітектури.

Особливу увагу необхідно приділити аналізу взаємодії між компонентами. Слід пояснити, яким чином дії користувача передаються контролеру, як контролер звертається до моделі та як результати оброблення даних передаються до представлення. Для більшої наочності доцільно подати цю взаємодію у вигляді структурованого опису або схеми.

Під час виконання роботи необхідно також оцінити переваги та обмеження архітектури Model–View–Controller. До переваг слід віднести чіткий розподіл відповідальності, зручність супроводження, можливість повторного використання компонентів і зменшення залежностей між частинами системи. До обмежень слід віднести ускладнення структури у простих програмах та можливість перевантаження контролерів при нераціональному проєктуванні.

Після виконання аналізу необхідно сформулювати висновки щодо ролі архітектури Model–View–Controller у сучасному програмному забезпеченні. У висновках слід узагальнити, у яких випадках її використання є доцільним та які переваги вона дає під час створення програмних систем.

Під час виконання роботи необхідно використовувати достовірні джерела інформації, дотримуватися принципів академічної доброчесності та коректно оформлювати використані матеріали.

Контрольні питання

1. Що таке архітектура Model–View–Controller?
2. Яке призначення компонента Model?
3. Яке призначення компонента View?
4. Яке призначення компонента Controller?
5. Яким чином взаємодіють компоненти Model, View і Controller?
6. Які переваги дає використання архітектури Model–View–Controller?
7. Які обмеження має архітектура Model–View–Controller?
8. У яких програмних системах широко використовується цей підхід?
9. Чому важливим є розподіл відповідальності між компонентами системи?
10. У яких випадках використання архітектури Model–View–Controller може бути недоцільним?

Зміст звіту

Звіт з практичної роботи повинен містити такі структурні елементи:
– назву практичної роботи;
– мету виконання практичної роботи;
– короткі теоретичні відомості про архітектуру Model–View–Controller;
– опис компонентів Model, View і Controller;
– аналіз взаємодії між компонентами;
– опис прикладу застосування архітектури у програмній системі або фреймворку;
– оцінку переваг та обмежень архітектури;
– висновки за результатами виконання практичної роботи;
– перелік використаних джерел інформації.

Практична робота 6. Побудова структурної схеми програмної системи з використанням архітектури MVC

Ключові положення

Побудова структурної схеми програмної системи є важливим етапом її проєктування, оскільки дає змогу визначити склад основних компонентів, характер зв'язків між ними та логіку функціонування всієї системи. При використанні архітектури MVC структурна схема повинна відображати розподіл програмної системи на три основні частини: модель, представлення та контролер. Такий підхід забезпечує логічне впорядкування компонентів, зменшує зв'язаність між ними та спрощує подальше розроблення, тестування і супроводження програмного забезпечення.

Архітектура MVC ґрунтується на принципі розподілу відповідальності. Компонент Model відповідає за дані, правила їх оброблення та бізнес-логіку системи. У межах моделі зосереджується функціональність, пов'язана зі зберіганням, зміною, перевіркою та передаванням даних. Model може взаємодіяти з базою даних, файлами, мережними сервісами або іншими джерелами інформації. У структурній схемі цей компонент зазвичай подається як окремий функціональний блок, що не залежить від конкретного способу відображення інформації.

Компонент View забезпечує подання інформації користувачу. Він містить елементи інтерфейсу, засоби виведення даних та механізми відображення результатів роботи системи. У структурній схемі представлення зазвичай розглядається як частина, що забезпечує взаємодію з користувачем, але не виконує складного оброблення даних. Основним завданням View є подання інформації у зручній формі відповідно до поточного стану системи.

Компонент Controller виконує координувальну функцію. Він приймає запити або дії користувача, виконує їх оброблення, звертається до моделі для зміни або отримання даних та визначає, яке представлення повинно бути використане для відображення результату. У структурній схемі контролер є проміжною ланкою між користувачем, представленням і моделлю. Саме через нього реалізується логіка керування роботою програмної системи.

Структурна схема системи, побудованої за архітектурою MVC, повинна чітко відображати напрямки інформаційних потоків. Користувач взаємодіє з представленням, представлення передає дії користувача контролеру, контролер звертається до моделі, модель виконує необхідні операції з даними та повертає результати, після чого представлення відображає оновлену інформацію. Такий ланцюг взаємодії є основою функціонування системи та повинен бути логічно відображений на схемі.

Правильна побудова структурної схеми дає змогу ще на етапі проєктування виявити надлишкові залежності, помилки розподілу функцій і потенційні труднощі у реалізації системи. Якщо у схемі модель безпосередньо залежить від представлення або представлення містить бізнес-логіку, це свідчить про порушення принципів MVC. Тому побудова структурної схеми має не лише ілюстративне, а й аналітичне значення.

У навчальній практиці структурні схеми часто будуються для вебзастосунків, інформаційних систем, програм керування даними або настільних застосунків. Наприклад, у вебзастосунку представленням можуть бути вебсторінки або шаблони, контролером – модулі оброблення HTTP-запитів, а моделлю – класи доступу до даних і бізнес-логіки. У

настільному застосунку представленням є графічний інтерфейс, контролером – обробники подій, а моделлю – внутрішні структури даних та алгоритми їх оброблення.

Побудова структурної схеми за архітектурою MVC сприяє формуванню системного мислення у здобувачів освіти. Вона дає змогу навчитися виділяти основні частини системи, встановлювати зв'язки між ними та розуміти логіку функціонування програмного застосунку ще до початку програмної реалізації. Це особливо важливо при розробленні складних програмних продуктів, де попереднє моделювання структури системи істотно знижує ризик помилок на наступних етапах.

Таким чином, побудова структурної схеми програмної системи з використанням архітектури MVC є важливим етапом проєктування, що забезпечує чіткий розподіл функцій між компонентами, спрощує реалізацію системи та створює основу для її подальшого розвитку і супроводження.

Практичне завдання

1. Ознайомитися з принципами побудови програмних систем на основі архітектури MVC.
2. Обрати приклад програмної системи для моделювання її структури.
3. Визначити основні функціональні компоненти обраної системи.
4. Розподілити компоненти системи за категоріями Model, View і Controller.
5. Побудувати структурну схему програмної системи з відображенням основних блоків і зв'язків між ними.
6. Пояснити призначення кожного елемента структурної схеми.
7. Проаналізувати напрямки інформаційних потоків між компонентами системи.
8. Сформулювати висновки щодо доцільності використання архітектури MVC для обраної програмної системи.

Методичні вказівки до виконання практичного завдання

Перед виконанням практичної роботи необхідно повторити теоретичний матеріал щодо архітектури MVC та особливостей побудови структурних схем програмних систем. Особливу увагу слід приділити розумінню ролі кожного компонента архітектури та принципів їх взаємодії.

На першому етапі необхідно обрати програмну систему, для якої буде побудовано структурну схему. Доцільно використовувати приклад нескладного вебзастосунку, інформаційної системи або настільної програми, у якій можна чітко виділити модель, представлення та контролер. Наприклад, це може бути система обліку користувачів, онлайн-каталог, система керування завданнями або програма для введення та оброблення даних.

Після вибору системи необхідно визначити її основні функціональні компоненти. Для цього слід встановити, які елементи відповідають за зберігання та оброблення даних, які забезпечують взаємодію з користувачем, а які координують роботу системи. На цьому етапі важливо не змішувати функції різних компонентів і зберігати чіткий розподіл відповідальності.

Далі потрібно виконати розподіл компонентів за трьома категоріями MVC. До моделі слід віднести класи або модулі, що містять дані та бізнес-логіку. До представлення – інтерфейсні елементи, сторінки, форми або шаблони. До контролера – елементи, що

обробляють запити, команди або події користувача та забезпечують взаємодію між моделлю і представленням.

Після цього необхідно побудувати структурну схему системи. Схема повинна містити основні блоки, що відповідають компонентам Model, View і Controller, а також стрілки або інші позначення, які відображають зв'язки між ними. Рекомендується використовувати прості та зрозумілі графічні позначення. У схемі важливо показати, як користувач взаємодіє із системою, як запити передаються до контролера, яким чином контролер звертається до моделі та як результати повертаються до представлення.

Після побудови схеми необхідно надати пояснення щодо призначення кожного її елемента. Опис повинен містити коротку характеристику функцій моделі, представлення і контролера, а також пояснення логіки взаємодії між ними. Особливу увагу слід приділити напрямкам передавання даних і команд у системі.

На завершальному етапі необхідно виконати аналіз побудованої структурної схеми. Слід оцінити, чи відповідає вона принципам MVC, чи правильно розподілені функції між компонентами, чи немає зайвих або нелогічних зв'язків. У висновках доцільно вказати, які переваги дає застосування архітектури MVC для обраної системи та які труднощі можуть виникати при її реалізації.

Під час виконання роботи необхідно дотримуватися принципів академічної доброчесності, використовувати коректну термінологію та забезпечувати логічну узгодженість між теоретичним описом і побудованою схемою.

Контрольні питання

1. Що таке структурна схема програмної системи?
2. Які компоненти містить архітектура MVC?
3. Яке призначення моделі у структурі програмної системи?
4. Яке призначення представлення у структурі програмної системи?
5. Яке призначення контролера у структурі програмної системи?
6. Яким чином у структурній схемі відображаються зв'язки між компонентами MVC?
7. Чому важливо правильно розподіляти функції між Model, View і Controller?
8. Які помилки можуть виникати при побудові структурної схеми за архітектурою MVC?
9. У яких типах програмних систем доцільно використовувати архітектуру MVC?
10. Які переваги дає попередня побудова структурної схеми перед програмною реалізацією системи?

Зміст звіту

Звіт з практичної роботи повинен містити такі структурні елементи:
– назву практичної роботи;
– мету виконання практичної роботи;
– короткі теоретичні відомості про архітектуру MVC;
– опис обраної програмної системи;
– перелік її основних функціональних компонентів;
– розподіл компонентів за категоріями Model, View і Controller;
– структурну схему програмної системи;

– пояснення елементів схеми та зв’язків між ними;
– висновки за результатами виконання практичної роботи;
– перелік використаних джерел інформації.

Практична робота 7. Аналіз багаторівневої архітектури програмного забезпечення

Ключові положення

Багаторівнева архітектура програмного забезпечення є підходом до побудови програмних систем, при якому система поділяється на логічні рівні, кожен з яких виконує окремі функції. Такий підхід дає змогу чітко розмежувати оброблення даних, бізнес-логіку та взаємодію з користувачем, що сприяє підвищенню якості програмного забезпечення, його масштабованості та зручності супроводження.

Основною ідеєю багаторівневої архітектури є принцип розподілу відповідальності між рівнями. Кожен рівень виконує чітко визначені функції та взаємодіє лише з сусідніми рівнями. Це дає змогу зменшити зв’язаність компонентів системи та забезпечити незалежність окремих частин програмного забезпечення.

Найбільш поширеною є трирівнева архітектура, що містить рівень представлення, рівень бізнес-логіки та рівень доступу до даних. Рівень представлення відповідає за взаємодію з користувачем і відображення інформації. Рівень бізнес-логіки реалізує основні правила оброблення даних та функціональність системи. Рівень доступу до даних забезпечує зберігання, отримання та модифікацію даних у базах даних або інших джерелах.

Рівень представлення містить інтерфейс користувача, засоби введення та виведення інформації. У вебзастосунках це можуть бути вебсторінки, шаблони або клієнтські застосунки. У настільних програмах цей рівень представлений графічним інтерфейсом користувача. Основною функцією цього рівня є відображення даних та передавання дій користувача до наступних рівнів системи.

Рівень бізнес-логіки є центральною частиною системи. Він визначає правила оброблення даних, виконує основні обчислення та забезпечує реалізацію функціональних вимог. Саме на цьому рівні реалізуються алгоритми, перевірка коректності даних та взаємодія між різними компонентами системи.

Рівень доступу до даних відповідає за взаємодію з базами даних або іншими сховищами інформації. Він забезпечує виконання операцій читання, запису, оновлення та видалення даних. Використання окремого рівня доступу до даних дає змогу змінювати спосіб зберігання інформації без впливу на інші частини системи.

У складніших системах можуть використовуватися додаткові рівні, наприклад рівень сервісів, рівень інтеграції або рівень безпеки. Це дає змогу ще більше деталізувати структуру системи та забезпечити більш гнучке управління її компонентами.

Багаторівнева архітектура широко застосовується у сучасних програмних системах, зокрема у вебзастосунках, корпоративних інформаційних системах та розподілених сервісах. Вона є основою багатьох фреймворків та технологій, що використовуються у практиці розроблення програмного забезпечення.

Перевагами багаторівневої архітектури є модульність, можливість масштабування, зручність тестування та спрощення супроводження системи. Завдяки чіткому розподілу

функцій зміни в одному рівні не потребують значних змін в інших частинах системи. Це особливо важливо при розробленні великих програмних продуктів.

Разом із тим, багаторівнева архітектура має і певні недоліки. Зокрема, вона може ускладнювати структуру системи, збільшувати кількість компонентів та вимагати додаткових ресурсів для реалізації взаємодії між рівнями. Тому її використання повинно бути обґрунтованим і відповідати складності задачі.

Аналіз багаторівневої архітектури дає змогу зрозуміти принципи побудови сучасних програмних систем, оцінити їх структуру та визначити доцільність використання такого підходу у конкретних умовах. Це є важливим етапом підготовки фахівців з інженерії програмного забезпечення.

Практичне завдання

1. Ознайомитися з принципами багаторівневої архітектури програмного забезпечення.
2. Визначити основні рівні багаторівневої архітектури.
3. Описати призначення кожного рівня.
4. Розглянути приклад програмної системи, побудованої за багаторівневою архітектурою.
5. Визначити, які компоненти системи належать до кожного рівня.
6. Проаналізувати взаємодію між рівнями системи.
7. Оцінити переваги використання багаторівневої архітектури.
8. Визначити можливі недоліки або обмеження цього підходу.
9. Порівняти багаторівневу архітектуру з архітектурою MVC.
10. Сформулювати висновки щодо доцільності використання багаторівневої архітектури.

Методичні вказівки до виконання практичного завдання

Перед виконанням роботи необхідно ознайомитися з теоретичним матеріалом щодо архітектурних підходів до побудови програмного забезпечення. Особливу увагу слід приділити принципам розподілу відповідальності та взаємодії між компонентами системи.

На першому етапі необхідно визначити сутність багаторівневої архітектури та її основні принципи. Доцільно пояснити, що система поділяється на окремі рівні, кожен з яких виконує певні функції.

Далі потрібно розглянути основні рівні архітектури. Для кожного рівня слід визначити його призначення, функції та місце у загальній структурі системи. Особливу увагу слід приділити взаємозв’язкам між рівнями та обмеженням їх взаємодії.

На наступному етапі необхідно обрати приклад програмної системи для аналізу. Це може бути вебзастосунок, інформаційна система або інший програмний продукт, у якому використовується багаторівнева архітектура.

Після вибору системи необхідно визначити її компоненти та розподілити їх за рівнями архітектури. Важливо забезпечити логічну відповідність між функціями компонентів та їх належністю до певного рівня.

Далі слід проаналізувати взаємодію між рівнями. Необхідно пояснити, яким чином дані передаються між рівнями, які інтерфейси використовуються та як забезпечується незалежність компонентів.

Окрему увагу необхідно приділити порівнянню багаторівневої архітектури з архітектурою MVC. Слід визначити спільні риси та відмінності цих підходів, а також сфери їх застосування.

Після виконання аналізу необхідно сформулювати висновки. У висновках слід оцінити ефективність використання багаторівневої архітектури, її переваги та обмеження, а також доцільність застосування у різних типах програмних систем.

Контрольні питання

1. Що таке багаторівнева архітектура програмного забезпечення?
2. Які основні принципи цього підходу?
3. Які рівні містить трирівнева архітектура?
4. Яке призначення рівня представлення?
5. Яке призначення рівня бізнес-логіки?
6. Яке призначення рівня доступу до даних?
7. Яким чином взаємодіють рівні системи?
8. Які переваги має багаторівнева архітектура?
9. Які недоліки має цей підхід?
10. Чим відрізняється багаторівнева архітектура від MVC?

Зміст звіту

Звіт з практичної роботи повинен містити такі структурні елементи:
– назву практичної роботи;
– мету виконання практичної роботи;
– короткі теоретичні відомості про багаторівневу архітектуру;
– опис основних рівнів архітектури;
– аналіз прикладу програмної системи;
– розподіл компонентів системи за рівнями;
– аналіз взаємодії між рівнями;
– порівняння з архітектурою MVC;
– висновки за результатами виконання практичної роботи;
– перелік використаних джерел інформації.

Практична робота 8. Порівняння типових архітектур програмного забезпечення

Ключові положення

Архітектура програмного забезпечення визначає загальну структуру системи, принципи організації її компонентів та характер взаємодії між ними. Вибір архітектурного підходу має суттєвий вплив на масштабованість, продуктивність, супроводжуваність і гнучкість програмної системи. Тому порівняння різних типових архітектур є важливим етапом підготовки фахівців у галузі інженерії програмного забезпечення.

До найбільш поширених архітектур належать монолітна архітектура, багаторівнева архітектура, архітектура Model–View–Controller та мікросервісна архітектура. Кожен із цих

підходів має свої особливості, переваги та обмеження, що визначають доцільність їх використання у конкретних умовах.

Монолітна архітектура передбачає побудову програмної системи у вигляді єдиного цілого, де всі компоненти тісно пов’язані між собою. У такій системі інтерфейс користувача, бізнес-логіка та доступ до даних реалізуються в межах одного застосунку. Перевагою цього підходу є простота розроблення та розгортання, проте він має обмеження щодо масштабування та супроводження великих систем.

Багаторівнева архітектура передбачає розподіл системи на окремі рівні, кожен з яких виконує визначені функції. Це дає змогу зменшити зв’язаність компонентів і забезпечити більш гнучку структуру системи. У порівнянні з монолітною архітектурою, багаторівнева архітектура забезпечує кращу підтримуваність та можливість масштабування, проте може ускладнювати структуру системи.

Архітектура Model–View–Controller орієнтована на розподіл відповідальності між компонентами системи на рівні організації взаємодії з користувачем. Вона забезпечує відокремлення даних, інтерфейсу та логіки керування. У порівнянні з багаторівневою архітектурою, MVC є більш спеціалізованим підходом, що застосовується переважно для організації інтерфейсної частини програмних систем.

Мікросервісна архітектура передбачає побудову системи у вигляді набору незалежних сервісів, кожен з яких виконує окрему функцію. Такі сервіси взаємодіють між собою через мережеві протоколи. Основною перевагою цього підходу є висока масштабованість та можливість незалежного розгортання компонентів. Разом із тим, мікросервісна архітектура є складною у реалізації та потребує розвиненої інфраструктури.

Порівняння архітектур доцільно виконувати за рядом критеріїв, зокрема складність реалізації, масштабованість, продуктивність, гнучкість, зручність тестування та супроводження. Наприклад, монолітна архітектура є простою у реалізації, але складною у масштабуванні, тоді як мікросервісна архітектура забезпечує високу масштабованість, але потребує значних ресурсів для підтримки.

Важливим аспектом є те, що архітектури можуть комбінуватися. Наприклад, у межах мікросервісної системи окремі сервіси можуть бути реалізовані із використанням багаторівневої архітектури або підходу MVC. Це дає змогу поєднувати переваги різних підходів та адаптувати архітектуру до конкретних вимог проєкту.

Аналіз і порівняння типових архітектур програмного забезпечення дає змогу сформувати системне уявлення про підходи до проєктування програмних систем. Це є необхідним для прийняття обґрунтованих рішень при розробленні програмного забезпечення.

Практичне завдання

1. Ознайомитися з основними типами архітектур програмного забезпечення.
2. Описати монолітну архітектуру та її особливості.
3. Описати багаторівневу архітектуру.
4. Описати архітектуру Model–View–Controller.
5. Описати мікросервісну архітектуру.
6. Визначити критерії порівняння архітектур.
7. Виконати порівняння архітектур за обраними критеріями.
8. Визначити переваги та недоліки кожної архітектури.

9. Проаналізувати можливість комбінування архітектур.
10. Сформулювати висновки щодо вибору архітектури для програмної системи.

Методичні вказівки до виконання практичного завдання

Перед виконанням роботи необхідно повторити теоретичний матеріал щодо архітектурних підходів у програмному забезпеченні. Особливу увагу слід приділити характеристикам кожної архітектури та умовам їх застосування.

На першому етапі необхідно визначити перелік архітектур, що будуть розглядатися у роботі. Доцільно включити найбільш поширені підходи, такі як монолітна, багаторівнева, MVC та мікросервісна архітектури.

Далі потрібно описати кожну архітектуру окремо. Опис повинен містити її основні характеристики, принципи побудови та сфери застосування. Важливо забезпечити єдність підходу до опису, щоб надалі виконати коректне порівняння.

Наступним етапом є визначення критеріїв порівняння. До таких критеріїв можуть належати складність реалізації, масштабованість, продуктивність, гнучкість, зручність тестування, супроводження та вимоги до інфраструктури.

Після визначення критеріїв необхідно виконати порівняння архітектур. Доцільно представити результати у вигляді таблиці або структурованого опису, що дає змогу наочно оцінити відмінності між підходами.

Окрему увагу слід приділити аналізу переваг та недоліків кожної архітектури. Необхідно пояснити, у яких умовах використання певного підходу є доцільним, а у яких – недоцільним.

На завершальному етапі необхідно сформулювати висновки. У висновках слід узагальнити результати порівняння та визначити, які фактори впливають на вибір архітектури програмного забезпечення.

Контрольні питання

1. Що таке архітектура програмного забезпечення?
2. Які основні типи архітектур програмного забезпечення існують?
3. Які особливості монолітної архітектури?
4. Які особливості багаторівневої архітектури?
5. У чому полягає сутність архітектури Model–View–Controller?
6. Які принципи мікросервісної архітектури?
7. За якими критеріями доцільно порівнювати архітектури?
8. Які переваги та недоліки монолітної архітектури?
9. Які переваги та недоліки мікросервісної архітектури?
10. Як обрати архітектуру для програмної системи?

Зміст звіту

Звіт з практичної роботи повинен містити такі структурні елементи:
– назву практичної роботи;
– мету виконання практичної роботи;
– опис розглянутих архітектур;

– визначення критеріїв порівняння;
– результати порівняння архітектур;
– аналіз переваг та недоліків;
– висновки за результатами виконання практичної роботи;
– перелік використаних джерел інформації.

ТЕМА 3. ПОРОДЖУВАЛЬНІ ПАТЕРНИ

Практична робота 9. Розробка застосунку на основі патерну Factory Method

Ключові положення

Патерн Factory Method належить до породжувальних патернів проєктування та використовується для організації створення об’єктів без жорсткої прив’язки клієнтського коду до конкретних класів. Основна ідея цього патерну полягає у винесенні логіки створення об’єктів в окремий метод, який визначає, який саме клас екземпляра необхідно створити у певній ситуації. Такий підхід дає змогу підвищити гнучкість програмної системи, зменшити залежність між її компонентами та спростити подальше розширення функціональності.

У звичайному підході до програмування створення об’єктів часто виконується безпосередньо в клієнтському коді за допомогою виклику конструктора конкретного класу. Це призводить до того, що код, який використовує об’єкти, починає залежати від деталей їх реалізації. Якщо у подальшому виникає потреба змінити тип створюваного об’єкта або додати нові варіанти поведінки, доводиться змінювати клієнтський код. Патерн Factory Method усуває цю проблему шляхом інкапсуляції процесу створення об’єктів у спеціалізованому методі.

У структурі патерну Factory Method зазвичай виділяють кілька основних елементів. Першим є абстрактний продукт або базовий інтерфейс, який визначає спільні операції для всіх створюваних об’єктів. Другим є конкретні продукти, що реалізують цей інтерфейс і містять специфічну поведінку. Третім елементом є творець, у якому оголошується фабричний метод. Четвертим – конкретні творці, які перевизначають фабричний метод і повертають екземпляри конкретних продуктів. Саме така організація дає змогу делегувати вибір конкретного класу підкласам.

Перевага патерну полягає у тому, що клієнтський код працює не з конкретними класами, а з абстракціями. Це відповідає принципу програмування через інтерфейси та зменшує зв’язаність між частинами системи. Якщо виникає потреба додати новий тип продукту, достатньо створити новий конкретний клас продукту та новий конкретний клас творця без зміни існуючого коду, що вже працює з абстрактним інтерфейсом. Таким чином забезпечується відкритість системи до розширення.

Factory Method часто використовується у фреймворках, бібліотеках та застосунках, де потрібно створювати об’єкти різних типів залежно від вхідних параметрів, конфігурації або середовища виконання. Наприклад, у графічних застосунках цей патерн може використовуватися для створення різних елементів інтерфейсу. У системах оброблення документів він дає змогу створювати об’єкти для роботи з файлами різних форматів. У вебзастосунках аналогічний підхід використовується для створення обробників запитів, сервісів або джерел даних.

Під час розроблення застосунку на основі патерну Factory Method важливо правильно визначити спільний інтерфейс продуктів. Саме від нього залежить можливість уніфікованого використання різних об’єктів у клієнтському коді. Також необхідно забезпечити логічне розмежування між кодом створення об’єкта та кодом його використання. Якщо це розмежування виконано правильно, програма стає більш структурованою та зрозумілою.

Разом із перевагами патерн Factory Method має і певні обмеження. Його використання призводить до збільшення кількості класів, оскільки для кожного нового типу продукту часто потрібно створювати окремий клас творця. У простих програмах це може ускладнювати структуру без суттєвої користі. Тому застосування цього патерну повинно бути обґрунтованим і доцільним насамперед у тих випадках, коли очікується розширення системи або існує необхідність приховати деталі створення об’єктів.

У межах практичної роботи розробка застосунку на основі патерну Factory Method дає змогу засвоїти принципи породжувального проєктування, навчитися виділяти спільні властивості об’єктів, реалізовувати інтерфейси та організовувати код так, щоб створення об’єктів не було жорстко пов’язане з їх використанням. Це формує у здобувачів важливі навички побудови розширюваних програмних систем.

Практичне завдання

1. Ознайомитися з призначенням та структурою патерну Factory Method.
2. Визначити предметну область майбутнього застосунку.
3. Виділити спільний інтерфейс або базовий клас для групи об’єктів, що будуть створюватися у програмі.
4. Розробити не менше двох конкретних класів продуктів із різною поведінкою.
5. Створити базовий клас творця з фабричним методом.
6. Реалізувати конкретні класи творців, що створюють відповідні продукти.
7. Розробити клієнтську частину застосунку, що використовує об’єкти через спільний інтерфейс.
8. Продемонструвати роботу програми для різних типів створюваних об’єктів.
9. Проаналізувати переваги використання патерну Factory Method у розробленому застосунку.
10. Сформулювати висновки за результатами виконання роботи.

Методичні вказівки до виконання практичного завдання

Перед початком виконання практичної роботи необхідно повторити теоретичний матеріал щодо породжувальних патернів проєктування та детально ознайомитися зі структурою патерну Factory Method. Особливу увагу слід приділити відмінності між безпосереднім створенням об’єктів у коді та використанням фабричного методу як механізму інкапсуляції процесу створення.

На першому етапі потрібно обрати просту предметну область, у межах якої буде реалізовано застосунок. Це може бути система створення документів, повідомлень, транспортних засобів, графічних фігур, типів звітів або інших сутностей, для яких існує декілька варіантів реалізації. Вибрана предметна область повинна давати змогу чітко виділити спільний інтерфейс продуктів та кілька конкретних класів, що реалізують різні варіанти поведінки.

Після цього необхідно визначити абстрактний продукт. Він може бути поданий у вигляді інтерфейсу або базового класу. У ньому слід оголосити операції, спільні для всіх продуктів. Далі потрібно створити конкретні продукти, які реалізують цей інтерфейс і відрізняються між собою внутрішньою логікою або результатами роботи. На цьому етапі

важливо забезпечити логічну узгодженість між спільною структурою та індивідуальними особливостями кожного продукту.

Наступним етапом є створення класу творця. У ньому оголошується фабричний метод, який повертає об’єкт абстрактного продукту. За потреби у класі творця може міститися додаткова логіка, що працює із продуктом через спільний інтерфейс. Після цього слід створити конкретні класи творців, кожен із яких перевизначає фабричний метод та повертає екземпляр відповідного конкретного продукту.

Під час реалізації клієнтської частини програми необхідно забезпечити роботу через абстракції. Клієнтський код не повинен створювати конкретні продукти безпосередньо. Його завдання полягає у використанні фабричних методів та подальшій роботі зі створеними об’єктами через спільний інтерфейс. Саме цей підхід демонструє основну перевагу патерну Factory Method – відокремлення створення об’єктів від їх використання.

Після завершення програмної реалізації слід протестувати застосунок для різних варіантів створюваних об’єктів. Необхідно перевірити, чи коректно працює фабричний метод, чи створюються об’єкти потрібного типу та чи однаково зручно опрацьовуються вони у клієнтському коді. Доцільно також проаналізувати, наскільки просто можна додати новий тип продукту без змін у вже наявній логіці програми.

У пояснювальній частині звіту потрібно описати структуру програми, призначення основних класів і взаємодію між ними. Окремо слід показати, який саме елемент реалізації відповідає абстрактному продукту, конкретним продуктам, творцю та конкретним творцям. Після цього необхідно оцінити переваги й можливі недоліки застосування патерну у межах розробленого застосунку.

Під час виконання роботи слід дотримуватися єдиного стилю оформлення коду, використовувати зрозумілі імена класів і методів, а також забезпечити логічну відповідність між теоретичною моделлю патерну та її програмною реалізацією.

Контрольні питання

1. До якої групи патернів належить Factory Method?
2. Яке призначення патерну Factory Method?
3. У чому полягає відмінність між прямим створенням об’єктів і використанням фабричного методу?
4. Яку роль виконує абстрактний продукт у структурі патерну?
5. Яку роль виконують конкретні продукти?
6. Яке призначення класу творця?
7. Чому клієнтський код повинен працювати через інтерфейси, а не через конкретні класи?
8. Які переваги дає використання патерну Factory Method?
9. Які недоліки або обмеження має цей патерн?
10. У яких випадках доцільно використовувати Factory Method у програмних системах?

Зміст звіту

Звіт з практичної роботи повинен містити такі структурні елементи:

– назву практичної роботи;

– мету виконання практичної роботи;
– короткі теоретичні відомості про патерн Factory Method;
– опис предметної області застосунку;
– характеристику абстрактного продукту та конкретних продуктів;
– опис класу творця та конкретних класів творців;
– текст програми або основні фрагменти програмної реалізації;
– результати виконання та тестування програми;
– аналіз переваг використання патерну у розробленому застосунку;
– висновки за результатами виконання практичної роботи;
– перелік використаних джерел інформації.

Практична робота 10. Розробка застосунку на основі патерну Singleton

Ключові положення

Патерн Singleton належить до породжувальних патернів проєктування та використовується для забезпечення існування лише одного екземпляра певного класу у межах програмної системи. Основною ідеєю цього патерну є централізація доступу до об’єкта, який повинен бути унікальним, та контроль процесу його створення. Це дає змогу уникнути дублювання ресурсів, забезпечити узгодженість даних та спростити керування станом системи.

У звичайних умовах створення об’єктів виконується через виклик конструктора класу, що дозволяє створювати довільну кількість екземплярів. У випадку Singleton така поведінка обмежується: клас сам контролює процес створення об’єкта і гарантує, що у програмі існує лише один його екземпляр. Доступ до цього екземпляра здійснюється через спеціальний статичний метод, який повертає вже створений об’єкт або створює його при першому зверненні.

Структура патерну Singleton передбачає наявність закритого конструктора, що унеможливлює створення об’єкта поза межами класу, статичного поля для зберігання єдиного екземпляра та статичного методу доступу до нього. Така організація дає змогу централізувати керування об’єктом і забезпечити його доступність у будь-якій частині програми.

Патерн Singleton часто використовується у випадках, коли необхідно мати єдиний об’єкт для роботи з глобальними ресурсами. Прикладами можуть бути менеджери конфігурації, системи логування, керування підключенням до бази даних, кешування даних або управління станом застосунку. У таких випадках використання декількох екземплярів може призвести до некоректної роботи системи або втрати узгодженості даних.

Однією з переваг патерну є простота доступу до об’єкта. Оскільки екземпляр є глобально доступним, немає потреби передавати його через параметри функцій або зберігати у багатьох частинах програми. Це спрощує структуру коду та зменшує кількість залежностей між компонентами системи.

Разом із тим, використання Singleton має і певні недоліки. Зокрема, він може ускладнювати тестування програмного забезпечення, оскільки створює глобальний стан, який важко ізолювати. Крім того, надмірне використання цього патерну може призвести до порушення принципів об’єктно-орієнтованого програмування, зокрема принципу єдиної відповідальності.

Важливим аспектом реалізації Singleton є забезпечення коректної роботи у багатопотоковому середовищі. У таких випадках необхідно передбачити механізми синхронізації доступу до об’єкта, щоб уникнути створення декількох екземплярів при одночасному зверненні з різних потоків.

Під час розроблення застосунку на основі патерну Singleton необхідно чітко визначити, чи дійсно потрібен єдиний екземпляр об’єкта. Використання цього патерну повинно бути обґрунтованим, оскільки неправильне застосування може призвести до ускладнення структури програми.

У межах практичної роботи реалізація патерну Singleton дає змогу засвоїти принципи керування життєвим циклом об’єктів, навчитися обмежувати створення екземплярів класу та організовувати централізований доступ до ресурсів у програмній системі.

Практичне завдання

1. Ознайомитися з призначенням та структурою патерну Singleton.
2. Обрати предметну область застосунку, у якій доцільно використовувати єдиний екземпляр об’єкта.
3. Розробити клас, що реалізує патерн Singleton.
4. Забезпечити обмеження створення об’єктів цього класу.
5. Реалізувати метод доступу до єдиного екземпляра.
6. Створити клієнтську частину програми, що використовує Singleton.
7. Продемонструвати, що у програмі використовується лише один екземпляр об’єкта.
8. За потреби реалізувати потокобезпечний варіант Singleton.
9. Проаналізувати переваги та недоліки використання патерну у створеному застосунку.
10. Сформулювати висновки за результатами виконання роботи.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно ознайомитися з теоретичними відомостями щодо породжувальних патернів та особливостями патерну Singleton. Особливу увагу слід приділити механізму обмеження створення об’єктів та способам організації доступу до єдиного екземпляра.

На першому етапі необхідно визначити предметну область застосунку. Доцільно обрати задачу, у якій природно існує єдиний об’єкт, наприклад менеджер конфігурації, журнал подій або керування підключенням до бази даних. Вибір повинен бути обґрунтованим з точки зору доцільності використання Singleton.

Далі потрібно реалізувати клас Singleton. У цьому класі необхідно оголосити закритий конструктор, статичне поле для зберігання екземпляра та статичний метод доступу. Важливо забезпечити, щоб створення об’єкта відбувалося лише один раз.

Після цього слід розробити клієнтську частину програми. Клієнт повинен отримувати доступ до об’єкта через статичний метод та виконувати з ним певні дії. Доцільно продемонструвати використання Singleton у різних частинах програми, щоб показати, що всі звернення відбуваються до одного й того самого екземпляра.

На наступному етапі необхідно перевірити правильність реалізації. Для цього можна додати виведення інформації про створення об’єкта або використати ідентифікатори, що дозволяють переконатися у відсутності дублювання екземплярів.

За потреби можна реалізувати потокобезпечний варіант Singleton. Це передбачає використання механізмів синхронізації або інших підходів, що гарантують коректну роботу у багатопотоковому середовищі.

Після завершення програмної реалізації необхідно виконати аналіз отриманого результату. Слід оцінити, наскільки доцільним є використання Singleton у вибраній задачі, які переваги він надає та які обмеження має.

У звіті необхідно чітко описати структуру реалізації патерну, показати взаємодію між компонентами та зробити обґрунтовані висновки щодо ефективності використання цього підходу.

Контрольні питання

1. До якої групи патернів належить Singleton?
2. Яке призначення патерну Singleton?
3. Які основні елементи містить структура Singleton?
4. Чому конструктор класу Singleton повинен бути закритим?
5. Як здійснюється доступ до єдиного екземпляра об’єкта?
6. У яких випадках доцільно використовувати Singleton?
7. Які переваги має цей патерн?
8. Які недоліки або обмеження має Singleton?
9. Які проблеми можуть виникати у багатопотоковому середовищі?
10. Як забезпечити потокобезпечність Singleton?

Зміст звіту

Звіт з практичної роботи повинен містити такі структурні елементи:
– назву практичної роботи;
– мету виконання практичної роботи;
– короткі теоретичні відомості про патерн Singleton;
– опис предметної області застосунку;
– опис реалізації класу Singleton;
– текст програми або основні фрагменти коду;
– результати виконання та тестування програми;
– аналіз переваг та недоліків використання патерну;
– висновки за результатами виконання практичної роботи;
– перелік використаних джерел інформації.

ТЕМА 4. СТРУКТУРНІ ПАТЕРНИ

Практична робота 11. Розробка застосунку на основі патерну Adapter

Ключові положення

Патерн Adapter належить до структурних патернів проєктування та використовується для забезпечення сумісності між об’єктами з несумісними інтерфейсами. Його основне призначення полягає у перетворенні інтерфейсу одного класу в інший інтерфейс, який очікує клієнт. Це дає змогу використовувати існуючі класи без необхідності змінювати їх внутрішню реалізацію.

У процесі розроблення програмного забезпечення часто виникають ситуації, коли необхідно інтегрувати компоненти, що мають різні інтерфейси. Наприклад, це може бути використання сторонніх бібліотек, застарілих модулів або компонентів, розроблених різними командами. У таких випадках безпосереднє використання об’єктів є неможливим або ускладненим. Патерн Adapter дає змогу вирішити цю проблему шляхом створення проміжного класу, який узгоджує інтерфейси.

Структура патерну Adapter передбачає наявність кількох основних елементів. Першим є цільовий інтерфейс, який визначає очікуваний спосіб взаємодії клієнта з об’єктом. Другим є клас, що потребує адаптації, інтерфейс якого не відповідає вимогам клієнта. Третім елементом є адаптер, який реалізує цільовий інтерфейс і містить посилання на об’єкт, що адаптується. Саме адаптер виконує перетворення викликів між інтерфейсами.

Існує два основних підходи до реалізації патерну Adapter: на основі композиції та на основі спадкування. У першому випадку адаптер містить об’єкт, що адаптується, і делегує йому виклики, змінюючи їх формат. У другому випадку адаптер наслідує клас, що адаптується, та перевизначає необхідні методи. Найбільш поширеним є підхід на основі композиції, оскільки він забезпечує більшу гнучкість.

Перевагою використання Adapter є можливість повторного використання існуючого коду без його модифікації. Це особливо важливо при інтеграції сторонніх компонентів або при модернізації застарілих систем. Крім того, патерн дає змогу ізолювати код, що залежить від конкретних реалізацій, від клієнтської частини програми.

Разом із тим, використання Adapter може призводити до ускладнення структури програми за рахунок введення додаткових класів. У деяких випадках це може ускладнювати розуміння коду, особливо якщо використовується велика кількість адаптерів. Тому застосування цього патерну повинно бути обґрунтованим і доцільним.

Патерн Adapter широко використовується у сучасних програмних системах. Наприклад, у веброзробці він може застосовуватися для інтеграції різних API, у графічних застосунках – для узгодження різних форматів даних, у системах оброблення даних – для перетворення форматів введення та виведення.

Розробка застосунку на основі патерну Adapter дає змогу засвоїти принципи інтеграції компонентів, навчитися працювати з різними інтерфейсами та забезпечувати їх сумісність без зміни існуючого коду. Це є важливим аспектом підготовки фахівців у галузі інженерії програмного забезпечення.

Практичне завдання

1. Ознайомитися з призначенням та структурою патерну Adapter.
2. Обрати предметну область застосунку, у якій існує необхідність узгодження різних інтерфейсів.
3. Визначити цільовий інтерфейс, який буде використовувати клієнт.
4. Обрати або створити клас із несумісним інтерфейсом, що потребує адаптації.
5. Розробити клас адаптера, який забезпечує узгодження інтерфейсів.
6. Реалізувати клієнтську частину застосунку, що працює через цільовий інтерфейс.
7. Продемонструвати коректну роботу програми після використання адаптера.
8. Проаналізувати переваги використання патерну Adapter у створеному застосунку.
9. Визначити можливі недоліки або обмеження застосування патерну.
10. Сформулювати висновки за результатами виконання роботи.

Методичні вказівки до виконання практичного завдання

Перед початком виконання практичної роботи необхідно ознайомитися з теоретичними відомостями щодо структурних патернів проєктування та особливостями патерну Adapter. Особливу увагу слід приділити розумінню ситуацій, у яких виникає потреба у використанні цього патерну.

На першому етапі необхідно обрати предметну область застосунку. Доцільно використовувати приклади, пов’язані з інтеграцією різних форматів даних, роботою з різними інтерфейсами або використанням сторонніх бібліотек. Наприклад, це може бути система оброблення повідомлень різних форматів, перетворення одиниць вимірювання або адаптація різних джерел даних.

Далі потрібно визначити цільовий інтерфейс, який буде використовувати клієнт. Цей інтерфейс повинен містити методи, що забезпечують необхідну функціональність для роботи програми. Після цього слід визначити клас, який має несумісний інтерфейс і не може бути використаний безпосередньо.

Наступним етапом є розробка класу адаптера. У цьому класі необхідно реалізувати цільовий інтерфейс та забезпечити перетворення викликів до методів класу, що адаптується. Якщо використовується підхід на основі композиції, адаптер повинен містити посилання на об’єкт, що адаптується, і делегувати йому виконання операцій.

Після цього слід реалізувати клієнтську частину програми. Клієнт повинен працювати через цільовий інтерфейс і не залежати від конкретної реалізації класу, що адаптується. Це демонструє основну перевагу патерну – ізоляцію клієнтського коду від деталей реалізації.

На наступному етапі необхідно перевірити правильність роботи програми. Слід переконатися, що адаптер коректно перетворює виклики та забезпечує сумісність між інтерфейсами. Доцільно протестувати програму для різних варіантів вхідних даних.

Після завершення реалізації необхідно виконати аналіз отриманого результату. Слід оцінити, наскільки ефективно патерн Adapter вирішує проблему несумісності інтерфейсів, які переваги він надає та які обмеження має.

У звіті необхідно чітко описати структуру застосунку, показати взаємодію між компонентами та обґрунтувати доцільність використання патерну.

Контрольні питання

1. До якої групи патернів належить Adapter?
2. Яке призначення патерну Adapter?
3. У яких випадках доцільно використовувати Adapter?
4. Які основні елементи містить структура патерну?
5. Що таке цільовий інтерфейс?
6. Яку роль виконує клас, що адаптується?
7. Яке призначення класу адаптера?
8. Які існують способи реалізації Adapter?
9. Які переваги дає використання цього патерну?
10. Які недоліки або обмеження має Adapter?

Зміст звіту

Звіт з практичної роботи повинен містити такі структурні елементи:
– назву практичної роботи;
– мету виконання практичної роботи;
– короткі теоретичні відомості про патерн Adapter;
– опис предметної області застосунку;
– характеристику цільового інтерфейсу та класу, що адаптується;
– опис реалізації класу адаптера;
– текст програми або основні фрагменти коду;
– результати виконання та тестування програми;
– аналіз переваг та недоліків використання патерну;
– висновки за результатами виконання практичної роботи;
– перелік використаних джерел інформації.

Практична робота 12. Розробка застосунку на основі патерну Decorator

Ключові положення

Патерн Decorator належить до структурних патернів проєктування та використовується для динамічного розширення функціональності об’єктів без зміни їхнього початкового коду. Основна ідея цього патерну полягає у «обгортанні» об’єкта іншим об’єктом, який реалізує той самий інтерфейс і додає нову поведінку. Такий підхід дає змогу гнучко комбінувати функціональність та уникати створення великої кількості підкласів.

У традиційному підході розширення функціональності часто виконується шляхом наслідування. Це призводить до збільшення кількості класів і ускладнення ієрархії, особливо якщо потрібно підтримувати різні комбінації можливостей. Патерн Decorator дозволяє вирішити цю проблему, оскільки функціональність додається шляхом композиції, а не наслідування.

Структура патерну Decorator містить кілька основних елементів. Першим є компонент – інтерфейс або абстрактний клас, який визначає базову функціональність. Другим є конкретний компонент, що реалізує базову поведінку. Третім є базовий декоратор, який

також реалізує інтерфейс компонента і містить посилання на об’єкт компонента. Четвертим елементом є конкретні декоратори, що розширюють поведінку компонента, додаючи нову функціональність до базових операцій.

Основною особливістю Decorator є те, що об’єкти можна вкладати один в один, утворюючи ланцюжок декораторів. Кожен декоратор виконує свою частину роботи та передає виклик далі. Це дає змогу формувати різні комбінації поведінки під час виконання програми, що є важливою перевагою цього підходу.

Перевагами патерну є гнучкість, можливість динамічного розширення функціональності, відсутність жорсткої залежності від ієрархії класів та відповідність принципу відкритості/закритості. Клієнтський код працює через інтерфейс компонента і не залежить від конкретних реалізацій або кількості застосованих декораторів.

Разом із тим, використання Decorator може ускладнювати структуру програми через велику кількість дрібних класів. Крім того, у складних системах може бути важко відстежити послідовність виконання декораторів. Тому важливо правильно організовувати структуру програми та використовувати зрозумілі назви класів.

Патерн Decorator широко застосовується у сучасних програмних системах. Наприклад, у бібліотеках введення-виведення він використовується для додавання буферизації, шифрування або форматування даних. У графічних застосунках – для додавання ефектів до об’єктів інтерфейсу. У веброзробці – для розширення функціональності обробників запитів або компонентів інтерфейсу.

Розробка застосунку на основі патерну Decorator дає змогу засвоїти принципи композиції об’єктів, навчитися гнучко розширювати функціональність та будувати масштабовані програмні системи.

Практичне завдання

1. Ознайомитися з призначенням та структурою патерну Decorator.
2. Обрати предметну область застосунку, у якій необхідно динамічно розширювати функціональність об’єктів.
3. Визначити базовий інтерфейс компонента.
4. Реалізувати конкретний компонент із базовою поведінкою.
5. Створити базовий клас декоратора.
6. Розробити не менше двох конкретних декораторів, що додають нову функціональність.
7. Реалізувати клієнтську частину застосунку, що використовує різні комбінації декораторів.
8. Продемонструвати роботу програми для різних варіантів розширення функціональності.
9. Проаналізувати переваги використання патерну Decorator у створеному застосунку.
10. Сформулювати висновки за результатами виконання роботи.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно ознайомитися з теоретичними відомостями щодо структурних патернів проєктування та принципами роботи патерну

Decorator. Особливу увагу слід приділити розумінню відмінності між наслідуванням і композицією як способами розширення функціональності.

На першому етапі необхідно обрати предметну область застосунку. Доцільно використовувати приклади, у яких об’єкти можуть мати додаткові властивості або поведінку. Наприклад, це може бути система оброблення тексту, напої з додатковими інгредієнтами, оброблення сигналів або графічні об’єкти з ефектами.

Далі потрібно визначити базовий інтерфейс компонента. У цьому інтерфейсі слід оголосити методи, що забезпечують основну функціональність. Після цього необхідно реалізувати конкретний компонент, який виконує базову операцію без додаткових розширень.

Наступним етапом є створення базового класу декоратора. Він повинен реалізовувати інтерфейс компонента та містити посилання на об’єкт, який декорується. Це дає змогу передавати виклики до базового об’єкта та розширювати його поведінку.

Після цього потрібно розробити конкретні декоратори. Кожен декоратор повинен додавати окрему функціональність до базового об’єкта. Наприклад, це може бути додавання нових властивостей, модифікація результату або виконання додаткових операцій перед або після виклику базового методу.

На наступному етапі необхідно реалізувати клієнтську частину застосунку. Клієнт повинен створювати базовий об’єкт і обгортати його одним або кількома декораторами. Важливо продемонструвати можливість комбінування декораторів та отримання різних результатів залежно від їх послідовності.

Після реалізації програми необхідно перевірити її роботу. Слід протестувати різні комбінації декораторів і переконатися, що функціональність розширюється коректно. Доцільно також проаналізувати, наскільки просто додати новий декоратор без зміни існуючого коду.

У звіті необхідно описати структуру застосунку, показати взаємодію між компонентом і декораторами та пояснити принцип розширення функціональності. Окремо слід оцінити переваги й обмеження використання патерну Decorator у межах розробленого застосунку.

Контрольні питання

1. До якої групи патернів належить Decorator?
2. Яке призначення патерну Decorator?
3. У чому полягає основна ідея цього патерну?
4. Які основні елементи містить структура Decorator?
5. Чим відрізняється декоратор від базового компонента?
6. Яку роль відіграє базовий клас декоратора?
7. Які переваги дає використання Decorator?
8. Які недоліки або обмеження має цей патерн?
9. У чому перевага композиції над наслідуванням у цьому контексті?
10. У яких випадках доцільно використовувати патерн Decorator?

Зміст звіту

Звіт з практичної роботи повинен містити такі структурні елементи:
– назву практичної роботи;
– мету виконання практичної роботи;
– короткі теоретичні відомості про патерн Decorator;
– опис предметної області застосунку;
– характеристику компонента та декораторів;
– текст програми або основні фрагменти коду;
– результати виконання та тестування програми;
– аналіз переваг та недоліків використання патерну;
– висновки за результатами виконання практичної роботи;
– перелік використаних джерел інформації.

ТЕМА 5. ПОВЕДІНКОВІ ПАТЕРНИ

Практична робота 13. Розробка застосунку на основі патерну Observer

Ключові положення

Патерн Observer належить до поведінкових патернів проєктування та використовується для організації залежності типу «один до багатьох» між об'єктами. У межах цього підходу один об'єкт, який називається суб'єктом, зберігає список залежних об'єктів – спостерігачів – і автоматично повідомляє їх про зміну свого стану. Це дає змогу забезпечити узгодженість даних між компонентами системи без жорсткої прив'язки між ними.

Основною ідеєю патерну є розділення відповідальності між об'єктом, що генерує події, та об'єктами, що реагують на ці події. Суб'єкт не повинен знати деталі реалізації спостерігачів, він лише викликає їхні методи сповіщення. Такий підхід забезпечує слабке зв'язування між компонентами системи та дає змогу легко додавати або видаляти спостерігачів без зміни коду суб'єкта.

Структура патерну Observer містить кілька основних елементів. Першим є інтерфейс суб'єкта, що визначає методи для підписки, відписки та сповіщення спостерігачів. Другим є конкретний суб'єкт, який зберігає стан і виконує сповіщення при його зміні. Третім елементом є інтерфейс спостерігача, що містить метод для отримання повідомлень. Четвертим – конкретні спостерігачі, які реалізують цей інтерфейс і реагують на зміни стану суб'єкта.

Патерн Observer широко використовується у системах із подієвою моделлю. Наприклад, у графічних інтерфейсах користувача зміна стану елементів інтерфейсу призводить до автоматичного оновлення інших компонентів. У вебзастосунках цей підхід використовується для організації реакції на події, зміни даних або повідомлення між компонентами системи.

Перевагами патерну є слабке зв'язування між об'єктами, можливість динамічного керування списком спостерігачів та зручність розширення системи. Клієнт може додавати нові спостерігачі без зміни існуючого коду, що відповідає принципу відкритості/закритості.

Разом із тим, використання Observer має і певні недоліки. Зокрема, у складних системах може виникати велика кількість повідомлень між об'єктами, що ускладнює відстеження логіки виконання програми. Крім того, неправильна організація підписки може призводити до витоків пам'яті або небажаних залежностей.

Під час розроблення застосунку на основі патерну Observer важливо правильно визначити, які об'єкти повинні бути суб'єктами, а які – спостерігачами. Необхідно також забезпечити ефективний механізм сповіщення та уникати надмірної кількості залежностей між компонентами.

У межах практичної роботи реалізація патерну Observer дає змогу засвоїти принципи подієвої взаємодії, навчитися організовувати обмін інформацією між об'єктами та будувати гнучкі програмні системи з низьким рівнем зв'язаності.

Практичне завдання

1. Ознайомитися з призначенням та структурою патерну Observer.
2. Обрати предметну область застосунку, у якій необхідно реалізувати механізм сповіщення.
3. Визначити інтерфейс суб’єкта.
4. Реалізувати клас суб’єкта, що зберігає стан і повідомляє спостерігачів.
5. Визначити інтерфейс спостерігача.
6. Розробити не менше двох конкретних спостерігачів.
7. Реалізувати механізм підписки та відписки спостерігачів.
8. Продемонструвати сповіщення спостерігачів при зміні стану суб’єкта.
9. Проаналізувати переваги використання патерну Observer у створеному застосунку.
10. Сформулювати висновки за результатами виконання роботи.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно ознайомитися з теоретичними відомостями щодо поведінкових патернів проєктування та принципами роботи патерну Observer. Особливу увагу слід приділити механізму підписки та сповіщення об’єктів.

На першому етапі необхідно обрати предметну область застосунку. Доцільно використовувати приклади, у яких відбувається зміна стану об’єкта та необхідно інформувати інші компоненти. Наприклад, це може бути система моніторингу температури, оновлення інтерфейсу користувача, система повідомлень або оброблення подій.

Далі потрібно визначити інтерфейс суб’єкта. У ньому слід передбачити методи для додавання, видалення та сповіщення спостерігачів. Після цього необхідно реалізувати конкретний клас суб’єкта, який містить стан і виконує сповіщення при його зміні.

Наступним етапом є визначення інтерфейсу спостерігача. Він повинен містити метод, який викликається при отриманні повідомлення від суб’єкта. Після цього потрібно реалізувати конкретні класи спостерігачів, що виконують певні дії у відповідь на зміну стану.

На етапі реалізації необхідно забезпечити можливість динамічного керування списком спостерігачів. Клієнтська частина програми повинна мати змогу додавати та видаляти спостерігачів у процесі виконання.

Після цього слід реалізувати механізм сповіщення. При зміні стану суб’єкта необхідно викликати метод сповіщення, який передає інформацію всім підписаним спостерігачам. Доцільно продемонструвати роботу програми для різних сценаріїв, зокрема при додаванні та видаленні спостерігачів.

Після завершення програмної реалізації необхідно протестувати застосунок. Слід переконатися, що всі спостерігачі отримують повідомлення та коректно реагують на зміну стану суб’єкта.

У звіті необхідно описати структуру застосунку, показати взаємодію між суб’єктом і спостерігачами та обґрунтувати доцільність використання патерну. Окремо слід проаналізувати переваги й можливі обмеження цього підходу.

Контрольні питання

1. До якої групи патернів належить Observer?
2. Яке призначення патерну Observer?
3. У чому полягає основна ідея цього патерну?
4. Які основні елементи містить структура Observer?
5. Яку роль виконує суб’єкт?
6. Яку роль виконують спостерігачі?
7. Як реалізується механізм підписки та відписки?
8. Які переваги дає використання Observer?
9. Які недоліки або обмеження має цей патерн?
10. У яких випадках доцільно використовувати Observer?

Зміст звіту

Звіт з практичної роботи повинен містити такі структурні елементи:
– назву практичної роботи;
– мету виконання практичної роботи;
– короткі теоретичні відомості про патерн Observer;
– опис предметної області застосунку;
– характеристику суб’єкта та спостерігачів;
– текст програми або основні фрагменти коду;
– результати виконання та тестування програми;
– аналіз переваг та недоліків використання патерну;
– висновки за результатами виконання практичної роботи;
– перелік використаних джерел інформації.

Практична робота 14. Розробка застосунку на основі патерну Strategy

Ключові положення

Патерн Strategy належить до поведінкових патернів проєктування та використовується для визначення сімейства алгоритмів, інкапсуляції кожного з них та забезпечення можливості їх взаємозаміни під час виконання програми. Основна ідея цього патерну полягає у винесенні алгоритмів у окремі класи та організації роботи клієнта через спільний інтерфейс. Це дає змогу змінювати поведінку об’єкта без зміни його коду.

У традиційному підході вибір алгоритму часто реалізується за допомогою умовних операторів, що призводить до ускладнення коду та зниження його гнучкості. При додаванні нових алгоритмів необхідно змінювати існуючий код, що порушує принцип відкритості/закритості. Патерн Strategy дозволяє уникнути цієї проблеми, оскільки кожен алгоритм реалізується як окремий клас, а вибір алгоритму здійснюється через інтерфейс.

Структура патерну Strategy містить кілька основних елементів. Першим є інтерфейс стратегії, що визначає загальний набір операцій для всіх алгоритмів. Другим є конкретні стратегії, які реалізують цей інтерфейс і містять різні варіанти алгоритмів. Третім елементом

є контекст – клас, який використовує об’єкт стратегії для виконання певної операції. Контекст зберігає посилання на поточну стратегію та делегує їй виконання алгоритму.

Основною перевагою патерну Strategy є можливість динамічної зміни алгоритму під час виконання програми. Це дає змогу адаптувати поведінку системи до різних умов без зміни її структури. Крім того, використання цього патерну сприяє зменшенню зв’язаності між компонентами та підвищує повторне використання коду.

Патерн Strategy широко застосовується у програмних системах, де необхідно реалізувати різні варіанти оброблення даних. Наприклад, це може бути сортування різними методами, обчислення вартості залежно від умов, вибір алгоритму стиснення даних або реалізація різних режимів роботи системи.

Разом із перевагами патерн має і певні недоліки. Зокрема, використання Strategy призводить до збільшення кількості класів, що може ускладнювати структуру програми. Крім того, клієнт повинен знати про існування різних стратегій і вибирати відповідну, що може ускладнювати логіку використання.

Під час розроблення застосунку на основі патерну Strategy важливо правильно визначити сімейство алгоритмів та забезпечити їхню взаємозамінність через спільний інтерфейс. Контекст повинен бути незалежним від конкретних реалізацій стратегій і працювати лише через абстракцію.

У межах практичної роботи реалізація патерну Strategy дає змогу засвоїти принципи інкапсуляції алгоритмів, навчитися будувати гнучкі програмні системи та ефективно організовувати змінювану поведінку об’єктів.

Практичне завдання

1. Ознайомитися з призначенням та структурою патерну Strategy.
2. Обрати предметну область застосунку, у якій існує декілька варіантів алгоритмів.
3. Визначити інтерфейс стратегії.
4. Реалізувати не менше двох конкретних стратегій.
5. Створити клас контексту, що використовує стратегію.
6. Реалізувати можливість зміни стратегії під час виконання програми.
7. Розробити клієнтську частину застосунку.
8. Продемонструвати роботу програми з різними стратегіями.
9. Проаналізувати переваги використання патерну Strategy у створеному застосунку.
10. Сформулювати висновки за результатами виконання роботи.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно ознайомитися з теоретичними відомостями щодо поведінкових патернів проєктування та принципами роботи патерну Strategy. Особливу увагу слід приділити розумінню відокремлення алгоритмів від контексту їх використання.

На першому етапі необхідно обрати предметну область застосунку. Доцільно використовувати приклади, у яких існує декілька способів виконання однієї і тієї ж операції. Наприклад, це може бути система сортування даних, розрахунок вартості послуг, оброблення сигналів або вибір алгоритму обчислень.

Далі потрібно визначити інтерфейс стратегії. У цьому інтерфейсі слід оголосити метод, що реалізує алгоритм. Після цього необхідно створити конкретні класи стратегій, кожен з яких реалізує власний варіант алгоритму.

Наступним етапом є розробка класу контексту. Контекст повинен містити посилання на об’єкт стратегії та викликати її методи для виконання операцій. Важливо забезпечити можливість зміни стратегії під час виконання програми.

Після цього слід реалізувати клієнтську частину застосунку. Клієнт повинен мати змогу обирати необхідну стратегію та передавати її до контексту. Доцільно продемонструвати зміну стратегії у процесі виконання програми.

На етапі тестування необхідно перевірити роботу програми для різних варіантів стратегій. Слід переконатися, що зміна стратегії не впливає на структуру програми і виконується коректно.

Після завершення реалізації необхідно виконати аналіз отриманого результату. Слід оцінити, наскільки ефективно патерн Strategy дозволяє змінювати поведінку системи, які переваги він надає та які обмеження має.

У звіті необхідно описати структуру застосунку, показати взаємодію між контекстом і стратегіями та обґрунтувати доцільність використання патерну.

Контрольні питання

1. До якої групи патернів належить Strategy?
2. Яке призначення патерну Strategy?
3. У чому полягає основна ідея цього патерну?
4. Які основні елементи містить структура Strategy?
5. Яку роль виконує інтерфейс стратегії?
6. Яку роль виконують конкретні стратегії?
7. Що таке контекст у патерні Strategy?
8. Які переваги дає використання цього патерну?
9. Які недоліки або обмеження має Strategy?
10. У яких випадках доцільно використовувати цей патерн?

Зміст звіту

Звіт з практичної роботи повинен містити такі структурні елементи:
– назву практичної роботи;
– мету виконання практичної роботи;
– короткі теоретичні відомості про патерн Strategy;
– опис предметної області застосунку;
– характеристику стратегій і контексту;
– текст програми або основні фрагменти коду;
– результати виконання та тестування програми;
– аналіз переваг та недоліків використання патерну;
– висновки за результатами виконання практичної роботи;
– перелік використаних джерел інформації.

ЧАСТИНА 2

ФРЕЙМВОРКИ ДЛЯ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ТЕМА 6. ВИКОРИСТАННЯ ФРЕЙМВОРКІВ ДЛЯ СТВОРЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Практична робота 15. Аналіз поняття фреймворку та його ролі у процесі розробки програмного забезпечення

Ключові положення

Фреймворк у програмному забезпеченні є програмною платформою, що визначає структуру застосунку, надає набір готових компонентів та встановлює правила взаємодії між ними. На відміну від бібліотек, які використовуються вибірково, фреймворк визначає загальну архітектуру програмної системи та керує потоком виконання програми. Це явище часто описується як принцип інверсії керування, коли не програма викликає функції фреймворку, а фреймворк визначає, коли і як викликається код розробника.

Фреймворки використовуються для спрощення процесу розроблення програмного забезпечення, підвищення продуктивності праці розробників та забезпечення стандартизації підходів до створення програмних систем. Вони містять готові рішення для типових задач, таких як оброблення запитів, робота з базами даних, управління станом застосунку, організація інтерфейсу користувача та забезпечення безпеки.

Важливою характеристикою фреймворків є їх орієнтація на певний тип застосунків або технологічну платформу. Існують вебфреймворки, фреймворки для розроблення мобільних застосунків, настільних програм, ігрових систем та вбудованих пристроїв. Наприклад, Flask і Django використовуються для створення вебзастосунків, Qt – для настільних програм, Flutter – для мобільних застосунків, а .NET є універсальною платформою для розроблення різних типів програмного забезпечення.

Фреймворки часто реалізують архітектурні патерни, зокрема Model–View–Controller або багаторівневу архітектуру. Це дає змогу забезпечити єдину структуру програмного коду, спростити його супроводження та підвищити якість програмного забезпечення. Використання фреймворків сприяє дотриманню принципів проєктування, таких як розподіл відповідальності, модульність і повторне використання коду.

Перевагами використання фреймворків є скорочення часу розроблення, наявність готових компонентів, стандартизація структури проєкту та підтримка спільнотою розробників. Завдяки цьому розробники можуть зосередитися на реалізації бізнес-логіки, не витрачаючи час на створення базової інфраструктури застосунку.

Разом із тим, використання фреймворків має і певні обмеження. Зокрема, розробник повинен дотримуватися встановлених правил і структури, що може обмежувати гнучкість. Крім того, фреймворки можуть мати високу складність освоєння, а також накладати додаткові вимоги до ресурсів системи. У деяких випадках використання фреймворку може бути надмірним для простих задач.

Роль фреймворків у процесі розроблення програмного забезпечення є визначальною у сучасних умовах. Вони забезпечують основу для побудови складних систем, сприяють впровадженню найкращих практик розроблення та дозволяють створювати масштабовані й підтримувані програмні продукти.

Аналіз поняття фреймворку та його ролі дає змогу зрозуміти сучасні підходи до розроблення програмного забезпечення, оцінити переваги використання готових платформ та визначити доцільність їх застосування у різних проєктах.

Практичне завдання

1. Ознайомитися з поняттям фреймворку у програмному забезпеченні.
2. Визначити основні характеристики фреймворків.
3. Порівняти фреймворки з бібліотеками програмного забезпечення.
4. Розглянути приклади сучасних фреймворків.
5. Проаналізувати роль фреймворків у процесі розроблення програмного забезпечення.
6. Визначити переваги використання фреймворків.
7. Проаналізувати недоліки та обмеження використання фреймворків.
8. Розглянути приклади застосування фреймворків у різних типах програмних систем.
9. Сформулювати висновки щодо доцільності використання фреймворків.

Методичні вказівки до виконання практичного завдання

Перед виконанням роботи необхідно ознайомитися з теоретичним матеріалом щодо сучасних підходів до розроблення програмного забезпечення та ролі фреймворків у цих процесах. Особливу увагу слід приділити розумінню принципу інверсії керування та відмінності між фреймворками і бібліотеками.

На першому етапі необхідно сформулювати визначення поняття фреймворку. Доцільно розглянути декілька джерел інформації та виділити основні характеристики, що визначають цей термін.

Далі слід виконати порівняння фреймворків із бібліотеками. Необхідно визначити відмінності у способах використання, рівні впливу на структуру програми та ролі у процесі розроблення.

На наступному етапі потрібно розглянути приклади сучасних фреймворків. Доцільно обрати декілька фреймворків різного призначення, наприклад для веброзробки, настільних застосунків або мобільних систем, та коротко охарактеризувати їх.

Особливу увагу слід приділити аналізу ролі фреймворків у процесі розроблення програмного забезпечення. Необхідно визначити, які задачі вони дозволяють вирішувати, як впливають на продуктивність розроблення та які вимоги висувають до розробника.

Після цього необхідно проаналізувати переваги та недоліки використання фреймворків. Важливо оцінити як позитивні аспекти, так і можливі обмеження, що можуть виникати при їх застосуванні.

На завершальному етапі слід сформулювати висновки щодо доцільності використання фреймворків у різних типах програмних систем. У висновках доцільно узагальнити отримані результати та визначити основні фактори, що впливають на вибір фреймворку.

Контрольні питання

1. Що таке фреймворк у програмному забезпеченні?
2. Які основні характеристики фреймворків?

3. У чому полягає принцип інверсії керування?
4. Чим відрізняється фреймворк від бібліотеки?
5. Які типи фреймворків існують?
6. Які приклади сучасних фреймворків можна навести?
7. Які переваги дає використання фреймворків?
8. Які недоліки мають фреймворки?
9. У яких випадках використання фреймворку є доцільним?
10. Яку роль відіграють фреймворки у сучасній інженерії програмного забезпечення?

Зміст звіту

Звіт з практичної роботи повинен містити такі структурні елементи:
– назву практичної роботи;
– мету виконання практичної роботи;
– визначення поняття фреймворку;
– порівняння фреймворків і бібліотек;
– приклади сучасних фреймворків;
– аналіз ролі фреймворків у процесі розроблення;
– переваги та недоліки використання;
– висновки за результатами виконання практичної роботи;
– перелік використаних джерел інформації.

Практична робота 16. Порівняння фреймворків, бібліотек та програмних платформ

Ключові положення

Фреймворки, бібліотеки та програмні платформи є ключовими складовими сучасного процесу розроблення програмного забезпечення. Вони забезпечують повторне використання коду, прискорюють розроблення та сприяють стандартизації підходів до створення програмних систем. Разом із тим, ці поняття мають суттєві відмінності, які необхідно враховувати при виборі інструментів для розроблення.

Бібліотека є набором готових функцій, класів або модулів, які розробник може використовувати у власній програмі для реалізації окремих задач. Основною характеристикою бібліотеки є те, що вона не визначає структуру програми і не керує її виконанням. Розробник сам вирішує, коли і як викликати функції бібліотеки. Таким чином, бібліотека є допоміжним інструментом, що розширює можливості мови програмування.

Фреймворк, на відміну від бібліотеки, визначає загальну структуру програмної системи та керує потоком виконання. У цьому випадку реалізується принцип інверсії керування: не розробник викликає функції фреймворку, а фреймворк викликає код розробника у визначених точках. Фреймворк задає архітектуру застосунку, правила організації коду та механізми взаємодії компонентів.

Програмна платформа є більш широким поняттям, що охоплює середовище виконання, інструменти розроблення, бібліотеки та фреймворки. Платформа визначає технологічну основу для створення та виконання програмного забезпечення. Вона може

містити віртуальні машини, компілятори, системи керування пакетами, засоби взаємодії з операційною системою та інші компоненти.

Основна відмінність між цими трьома поняттями полягає у рівні абстракції та ступені впливу на програмну систему. Бібліотека є найменш впливовим елементом, оскільки використовується локально для виконання окремих задач. Фреймворк має більший вплив, оскільки визначає структуру програми та керує її виконанням. Програмна платформа є найвищим рівнем абстракції, оскільки визначає загальне середовище розроблення та виконання програмного забезпечення.

Прикладами бібліотек є стандартні бібліотеки мов програмування, бібліотеки для роботи з графікою або математичні пакети. Прикладами фреймворків є Django, Flask, Qt, Angular та інші системи, що визначають структуру застосунків. Програмними платформами є .NET, Java Platform, Android, які забезпечують повний набір інструментів для створення та виконання програм.

Порівняння цих підходів дає змогу зрозуміти їх роль у процесі розроблення. Бібліотеки доцільно використовувати для вирішення окремих задач, фреймворки – для побудови структурованих застосунків, а платформи – як основу для створення програмного забезпечення певного типу.

Перевагою бібліотек є їх простота та гнучкість використання. Фреймворки забезпечують швидке розроблення та стандартизацію структури проєкту. Програмні платформи забезпечують комплексне середовище для створення та виконання програм.

Разом із тим, кожен із цих підходів має свої обмеження. Бібліотеки не забезпечують цілісної архітектури системи. Фреймворки можуть обмежувати гнучкість через встановлені правила. Платформи можуть вимагати значних ресурсів та часу для освоєння.

Аналіз і порівняння фреймворків, бібліотек та програмних платформ дає змогу сформувати системне уявлення про сучасні інструменти розроблення та обрати оптимальні рішення для конкретних задач.

Практичне завдання

1. Ознайомитися з поняттями бібліотеки, фреймворку та програмної платформи.
2. Визначити основні характеристики кожного з підходів.
3. Навести приклади бібліотек, фреймворків та платформ.
4. Проаналізувати відмінності між бібліотеками та фреймворками.
5. Проаналізувати відмінності між фреймворками та програмними платформами.
6. Визначити рівні абстракції для кожного підходу.
7. Порівняти їх за критеріями гнучкості, складності використання та впливу на структуру програми.
8. Проаналізувати переваги та недоліки кожного підходу.
9. Визначити, у яких випадках доцільно використовувати кожен із підходів.
10. Сформулювати висновки за результатами порівняння.

Методичні вказівки до виконання практичного завдання

Перед виконанням роботи необхідно ознайомитися з теоретичними відомостями щодо інструментів розроблення програмного забезпечення. Особливу увагу слід приділити

розумінню рівнів абстракції та ролі кожного підходу у процесі створення програмних систем.

На першому етапі необхідно сформулювати визначення понять бібліотеки, фреймворку та програмної платформи. Доцільно виділити їх основні характеристики та особливості використання.

Далі слід навести приклади кожного з підходів. Важливо використовувати реальні та сучасні приклади, що застосовуються у практиці розроблення програмного забезпечення.

Наступним етапом є виконання порівняння. Необхідно визначити критерії, за якими буде здійснюватися аналіз, та послідовно порівняти кожен підхід. Доцільно використовувати структурований опис або таблицю.

Особливу увагу слід приділити аналізу відмінностей між підходами. Необхідно пояснити, чим саме вони відрізняються за способом використання, впливом на структуру програми та рівнем абстракції.

Після цього необхідно проаналізувати переваги та недоліки кожного підходу. Важливо оцінити, у яких умовах їх використання є доцільним.

На завершальному етапі необхідно сформулювати висновки. У висновках слід узагальнити результати порівняння та визначити роль кожного підходу у сучасній інженерії програмного забезпечення.

Контрольні питання

1. Що таке бібліотека у програмному забезпеченні?
2. Що таке фреймворк?
3. Що таке програмна платформа?
4. У чому полягає принцип інверсії керування?
5. Чим відрізняється бібліотека від фреймворку?
6. Чим відрізняється фреймворк від програмної платформи?
7. Які приклади бібліотек можна навести?
8. Які приклади фреймворків можна навести?
9. Які приклади програмних платформ можна навести?
10. У яких випадках доцільно використовувати кожен із підходів?

Зміст звіту

Звіт з практичної роботи повинен містити такі структурні елементи:
– назву практичної роботи;
– мету виконання практичної роботи;
– визначення понять бібліотеки, фреймворку та програмної платформи;
– приклади кожного підходу;
– результати порівняння за обраними критеріями;
– аналіз переваг та недоліків;
– висновки за результатами виконання практичної роботи;
– перелік використаних джерел інформації.

Практична робота 17. Аналіз архітектурних особливостей сучасних фреймворків

Ключові положення

Сучасні фреймворки є складними програмними системами, що реалізують усталені архітектурні підходи та забезпечують основу для створення програмних застосунків. Їх архітектурні особливості визначають структуру програмного коду, механізми взаємодії компонентів, спосіб оброблення даних і загальну організацію процесу розроблення. Аналіз цих особливостей дає змогу зрозуміти принципи роботи фреймворків та ефективно використовувати їх у практиці.

Однією з ключових характеристик сучасних фреймворків є використання архітектурних патернів. Найбільш поширеним є підхід Model–View–Controller, що забезпечує розподіл відповідальності між компонентами системи. У межах цього підходу фреймворк визначає чітку структуру застосунку, де модель відповідає за дані, представлення – за відображення, а контролер – за оброблення запитів і керування взаємодією.

Іншою важливою особливістю є використання багаторівневої архітектури. Фреймворки часто реалізують поділ на рівень представлення, рівень бізнес-логіки та рівень доступу до даних. Такий підхід дає змогу забезпечити модульність, зменшити зв’язаність між компонентами та спростити тестування і супроводження програмного забезпечення.

Сучасні фреймворки активно використовують принцип інверсії керування та механізм впровадження залежностей. Це означає, що фреймворк керує життєвим циклом об’єктів і визначає, коли та як вони створюються і взаємодіють між собою. Впровадження залежностей дає змогу зменшити жорсткі зв’язки між компонентами та підвищити гнучкість системи.

Ще однією важливою архітектурною особливістю є модульність. Фреймворки будуються як набір взаємопов’язаних модулів, кожен з яких відповідає за певну функціональність. Це дає змогу використовувати лише необхідні компоненти та розширювати систему шляхом підключення додаткових модулів.

Фреймворки також забезпечують стандартизовану організацію проєкту. Вони визначають структуру каталогів, правила іменування файлів, способи розміщення коду та конфігурацій. Це сприяє уніфікації підходів до розроблення та полегшує роботу у команді.

У сучасних фреймворках широко застосовуються механізми маршрутизації, оброблення запитів, шаблонізації та взаємодії з базами даних. Маршрутизація визначає, як запити користувача обробляються та які компоненти відповідають за їх виконання. Шаблонізація дає змогу відокремити логіку відображення від бізнес-логіки. Інтеграція з базами даних забезпечується через спеціалізовані модулі або об’єктно-реляційні засоби доступу до даних.

Перевагами сучасних фреймворків є висока продуктивність розроблення, наявність готових архітектурних рішень, підтримка спільнотою та можливість створення масштабованих програмних систем. Разом із тим, вони можуть мати складну внутрішню структуру, що вимагає часу для освоєння, а також накладати обмеження на спосіб реалізації окремих компонентів.

Аналіз архітектурних особливостей сучасних фреймворків дає змогу сформувати системне уявлення про їх внутрішню організацію, зрозуміти принципи їх роботи та ефективно використовувати їх у процесі розроблення програмного забезпечення.

Практичне завдання

1. Ознайомитися з архітектурними підходами, що використовуються у сучасних фреймворках.
2. Обрати один або декілька фреймворків для аналізу.
3. Проаналізувати використання архітектурних патернів у вибраних фреймворках.
4. Визначити структуру програмного застосунку, що задається фреймворком.
5. Проаналізувати використання принципу інверсії керування та впровадження залежностей.
6. Дослідити модульну структуру фреймворку.
7. Описати механізми маршрутизації та оброблення запитів.
8. Проаналізувати засоби роботи з даними та базами даних.
9. Визначити переваги та недоліки архітектури фреймворку.
10. Сформулювати висновки щодо ефективності використання фреймворків.

Методичні вказівки до виконання практичного завдання

Перед виконанням роботи необхідно повторити теоретичний матеріал щодо архітектурних патернів, принципів побудови програмних систем та ролі фреймворків у процесі розроблення.

На першому етапі необхідно обрати один або декілька сучасних фреймворків. Доцільно використовувати ті фреймворки, що застосовуються у курсі, наприклад для веброзробки або створення настільних застосунків.

Далі потрібно проаналізувати архітектурні особливості обраного фреймворку. Слід визначити, які архітектурні патерни він використовує, як організована структура застосунку та які принципи покладені в основу його роботи.

Особливу увагу слід приділити аналізу інверсії керування та впровадження залежностей. Необхідно пояснити, яким чином фреймворк керує виконанням програми та як організовується взаємодія між компонентами.

На наступному етапі слід дослідити модульну структуру фреймворку. Необхідно визначити основні компоненти, їх призначення та спосіб взаємодії.

Далі потрібно проаналізувати механізми маршрутизації, оброблення запитів та роботи з даними. Важливо показати, як ці механізми реалізовані у межах обраного фреймворку.

Після цього необхідно оцінити переваги та недоліки архітектури фреймворку. Слід визначити, у яких випадках його використання є доцільним, а у яких – може бути обмеженим.

На завершальному етапі необхідно сформулювати висновки. У висновках слід узагальнити результати аналізу та визначити роль архітектурних особливостей фреймворків у сучасному програмному забезпеченні.

Контрольні питання

1. Які архітектурні підходи використовуються у сучасних фреймворках?
2. Що таке інверсія керування?
3. Що таке впровадження залежностей?
4. Які переваги дає використання архітектурних патернів у фреймворках?

5. Як організована структура застосунку у фреймворках?
6. Що таке модульна архітектура фреймворку?
7. Яку роль відіграє маршрутизація у вебфреймворках?
8. Як фреймворки забезпечують роботу з базами даних?
9. Які переваги мають сучасні фреймворки?
10. Які недоліки можуть мати фреймворки?

Зміст звіту

Звіт з практичної роботи повинен містити такі структурні елементи:
– назву практичної роботи;
– мету виконання практичної роботи;
– опис обраного фреймворку;
– аналіз його архітектурних особливостей;
– опис використаних архітектурних патернів;
– аналіз принципів інверсії керування та впровадження залежностей;
– оцінку переваг та недоліків;
– висновки за результатами виконання практичної роботи;
– перелік використаних джерел інформації.

Практична робота 18. Аналіз прикладів використання фреймворків у програмних системах

Ключові положення

Фреймворки є основою більшості сучасних програмних систем і визначають підходи до їх побудови, організації коду та взаємодії компонентів. Аналіз прикладів використання фреймворків дає змогу зрозуміти, як теоретичні принципи реалізуються у практичних рішеннях, а також оцінити ефективність застосування різних технологій у конкретних умовах.

У сучасних програмних системах фреймворки застосовуються для розроблення вебзастосунків, настільних програм, мобільних застосунків і розподілених систем. Кожен фреймворк орієнтований на певний тип задач і реалізує відповідні архітектурні підходи. Наприклад, вебфреймворки використовують підхід Model–View–Controller або подібні архітектури для організації оброблення запитів і відображення даних.

Фреймворки для веброзробки, такі як Django та Flask, забезпечують засоби маршрутизації, оброблення HTTP-запитів, роботу з базами даних і формування представлення. У таких системах фреймворк визначає структуру застосунку, а розробник реалізує бізнес-логіку відповідно до заданих правил. Це дає змогу швидко створювати масштабовані вебзастосунки.

Фреймворки для настільних застосунків, зокрема Qt, забезпечують засоби створення графічного інтерфейсу користувача, оброблення подій і взаємодії з операційною системою. У таких системах широко використовується подієва модель, що має ознаки патерну Observer. Це дає змогу організувати взаємодію між компонентами інтерфейсу та логікою програми.

Мобільні фреймворки, такі як Flutter, забезпечують створення інтерфейсу користувача, управління станом застосунку та взаємодію з апаратними ресурсами пристрою. Вони використовують компонентний підхід до побудови інтерфейсу та забезпечують кросплатформеність застосунків.

У корпоративних системах та серверних застосунках часто використовується платформа .NET, яка містить фреймворки для розроблення різних типів програмного забезпечення. Вона забезпечує засоби роботи з базами даних, мережеву взаємодію, безпеку та інші функціональні можливості.

Аналіз прикладів використання фреймворків показує, що вони реалізують типові архітектурні підходи та патерни проєктування. Це дозволяє розробникам використовувати готові рішення для організації програмних систем та зосереджуватися на реалізації функціональних вимог.

Перевагами використання фреймворків у програмних системах є скорочення часу розроблення, наявність готових компонентів, стандартизація структури коду та підтримка спільнотою. Разом із тим, вони можуть накладати обмеження на реалізацію та вимагати додаткового часу для освоєння.

Аналіз практичних прикладів використання фреймворків дає змогу сформувати у здобувачів навички вибору відповідних технологій, оцінки їх можливостей та ефективного застосування у процесі розроблення програмного забезпечення.

Практичне завдання

1. Ознайомитися з прикладами використання фреймворків у програмних системах.
2. Обрати один або декілька фреймворків для аналізу.
3. Описати призначення обраного фреймворку.
4. Проаналізувати архітектуру програмної системи, побудованої на його основі.
5. Визначити, які функціональні можливості забезпечує фреймворк.
6. Розглянути приклади реалізації основних компонентів системи.
7. Проаналізувати переваги використання фреймворку у конкретному випадку.
8. Визначити можливі недоліки або обмеження використання.
9. Порівняти декілька фреймворків (за потреби).
10. Сформулювати висновки щодо ефективності використання фреймворків.

Методичні вказівки до виконання практичного завдання

Перед виконанням роботи необхідно повторити теоретичний матеріал щодо фреймворків, їх архітектурних особливостей та ролі у процесі розроблення програмного забезпечення.

На першому етапі необхідно обрати фреймворк або декілька фреймворків для аналізу. Доцільно використовувати ті, що застосовуються у курсі, наприклад для веброзробки, настільних або мобільних застосунків.

Далі потрібно визначити призначення обраного фреймворку. Слід пояснити, для яких типів задач він використовується та які проблеми дозволяє вирішувати.

На наступному етапі необхідно проаналізувати приклад програмної системи, побудованої на основі цього фреймворку. Це може бути навчальний проєкт,

демонстраційний приклад або реальна система. Важливо встановити відповідність між компонентами системи та можливостями фреймворку.

Особливу увагу слід приділити аналізу архітектури системи. Необхідно визначити, які архітектурні підходи використовуються та як фреймворк впливає на організацію коду.

Далі потрібно проаналізувати функціональні можливості фреймворку. Слід визначити, які інструменти він надає для реалізації основних задач та як вони використовуються у системі.

Після цього необхідно оцінити переваги та недоліки використання фреймворку. Слід визначити, у яких умовах його застосування є доцільним, а у яких – може бути обмеженим.

На завершальному етапі необхідно сформулювати висновки. У висновках слід узагальнити результати аналізу та визначити роль фреймворків у сучасних програмних системах.

Контрольні питання

1. У яких програмних системах використовуються фреймворки?
2. Які приклади сучасних фреймворків можна навести?
3. Які задачі вирішують вебфреймворки?
4. Які задачі вирішують фреймворки для настільних застосунків?
5. Які задачі вирішують мобільні фреймворки?
6. Яку роль відіграє архітектура у фреймворках?
7. Які переваги дає використання фреймворків?
8. Які недоліки можуть мати фреймворки?
9. Як обрати фреймворк для конкретного проєкту?
10. Чому важливим є аналіз прикладів використання фреймворків?

Зміст звіту

Звіт з практичної роботи повинен містити такі структурні елементи:
– назву практичної роботи;
– мету виконання практичної роботи;
– опис обраного фреймворку;
– аналіз прикладу програмної системи;
– характеристику функціональних можливостей фреймворку;
– оцінку переваг та недоліків використання;
– висновки за результатами виконання практичної роботи;
– перелік використаних джерел інформації.

ТЕМА 7. ТИПИ ФРЕЙМВОРКІВ ДЛЯ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Практична робота 19. Класифікація фреймворків за сферою застосування

Ключові положення

Фреймворки відіграють ключову роль у сучасному програмному забезпеченні та застосовуються у різних галузях розроблення. Класифікація фреймворків за сферою застосування дає змогу систематизувати їх різноманіття, визначити їх призначення та обрати найбільш доцільний інструмент для вирішення конкретної задачі.

Однією з основних ознак класифікації фреймворків є сфера їх використання. За цим критерієм виділяють вебфреймворки, фреймворки для настільних застосунків, мобільні фреймворки, фреймворки для ігрових систем, фреймворки для оброблення даних та фреймворки для вбудованих систем. Кожна з цих груп має власні особливості та орієнтована на вирішення специфічних задач.

Вебфреймворки призначені для створення вебзастосунків та сервісів. Вони забезпечують оброблення HTTP-запитів, маршрутизацію, роботу з базами даних, формування представлення та інші функції, необхідні для розроблення вебсистем. Такі фреймворки часто використовують архітектурні підходи, зокрема Model–View–Controller або подібні до нього.

Фреймворки для настільних застосунків забезпечують створення графічного інтерфейсу користувача, оброблення подій та взаємодію з операційною системою. Вони надають інструменти для розроблення віконних програм, роботи з файлами, мережевими ресурсами та іншими компонентами системи.

Мобільні фреймворки орієнтовані на розроблення застосунків для мобільних пристроїв. Вони забезпечують доступ до апаратних можливостей пристрою, управління інтерфейсом користувача та підтримку різних платформ. Багато з них реалізують кросплатформений підхід, що дає змогу створювати застосунки для різних операційних систем на основі єдиного коду.

Фреймворки для ігрових систем використовуються для створення комп’ютерних ігор. Вони містять засоби роботи з графікою, фізикою, звуком, обробленням подій та іншими аспектами ігрового процесу. Такі фреймворки забезпечують основу для розроблення інтерактивних систем у реальному часі.

Фреймворки для оброблення даних застосовуються у задачах аналізу, оброблення та зберігання великих обсягів інформації. Вони забезпечують засоби роботи з масивами даних, виконання обчислень та інтеграцію з різними джерелами інформації.

Фреймворки для вбудованих систем використовуються для розроблення програмного забезпечення для мікроконтролерів та інших спеціалізованих пристроїв. Вони забезпечують доступ до апаратних ресурсів, управління пристроями та реалізацію алгоритмів керування.

Класифікація фреймворків за сферою застосування дає змогу зрозуміти їх призначення та визначити, який інструмент є найбільш доцільним для конкретного проєкту. Вибір фреймворку повинен враховувати вимоги до функціональності, продуктивності, масштабованості та умов використання програмної системи.

Таким чином, систематизація фреймворків за сферою застосування є важливим елементом підготовки фахівців у галузі інженерії програмного забезпечення, оскільки дозволяє орієнтуватися у сучасних технологіях та ефективно використовувати їх у практичній діяльності.

Практичне завдання

1. Ознайомитися з поняттям фреймворку та його роллю у програмному забезпеченні.
2. Визначити критерії класифікації фреймворків.
3. Описати основні групи фреймворків за сферою застосування.
4. Навести приклади вебфреймворків.
5. Навести приклади фреймворків для настільних застосунків.
6. Навести приклади мобільних фреймворків.
7. Навести приклади фреймворків для ігрових систем.
8. Навести приклади фреймворків для оброблення даних.
9. Навести приклади фреймворків для вбудованих систем.
10. Сформулювати висновки щодо вибору фреймворку залежно від сфери застосування.

Методичні вказівки до виконання практичного завдання

Перед виконанням роботи необхідно повторити теоретичний матеріал щодо поняття фреймворків, їх класифікації та ролі у процесі розроблення програмного забезпечення.

На першому етапі необхідно визначити критерії класифікації фреймворків. Основну увагу слід приділити сфері застосування як ключовому критерію систематизації.

Далі потрібно описати основні групи фреймворків. Для кожної групи слід визначити її призначення, основні функції та особливості використання.

На наступному етапі необхідно навести приклади фреймворків для кожної групи. Важливо використовувати сучасні та поширені технології, що застосовуються у практиці розроблення програмного забезпечення.

Після цього необхідно виконати аналіз отриманої класифікації. Слід визначити, у яких умовах використання певного типу фреймворку є доцільним, а також які фактори впливають на вибір інструменту.

На завершальному етапі необхідно сформулювати висновки. У висновках слід узагальнити результати роботи та визначити значення класифікації фреймворків для практичної діяльності розробника.

Контрольні питання

1. Що таке фреймворк у програмному забезпеченні?
2. За якими критеріями можна класифікувати фреймворки?
3. Які основні групи фреймворків за сферою застосування існують?
4. Які особливості вебфреймворків?
5. Які особливості фреймворків для настільних застосунків?
6. Які особливості мобільних фреймворків?
7. Які особливості фреймворків для ігрових систем?

8. Які особливості фреймворків для оброблення даних?
9. Які особливості фреймворків для вбудованих систем?
10. Як обрати фреймворк залежно від сфери застосування?

Зміст звіту

Звіт з практичної роботи повинен містити такі структурні елементи:
– назву практичної роботи;
– мету виконання практичної роботи;
– критерії класифікації фреймворків;
– опис груп фреймворків за сферою застосування;
– приклади фреймворків;
– аналіз доцільності використання;
– висновки за результатами виконання практичної роботи;
– перелік використаних джерел інформації.

Практична робота 20. Аналіз фреймворків для розробки застосунків для персональних комп'ютерів

Ключові положення

Фреймворки для розробки застосунків для персональних комп'ютерів є програмними платформами, що забезпечують створення настільних програм із графічним інтерфейсом користувача, обробленням подій та взаємодією з операційною системою. Вони надають набір інструментів для розроблення віконних застосунків, управління ресурсами системи та реалізації логіки роботи програмного забезпечення.

Однією з основних характеристик таких фреймворків є підтримка графічного інтерфейсу користувача. Вони містять готові компоненти, такі як вікна, кнопки, текстові поля, списки, таблиці та інші елементи інтерфейсу. Це дає змогу швидко створювати зручні та функціональні застосунки без необхідності реалізації базових елементів з нуля.

Фреймворки для настільних застосунків зазвичай реалізують подієву модель взаємодії. Це означає, що програма реагує на дії користувача, такі як натискання кнопок, введення тексту або переміщення курсора. Такий підхід має ознаки патерну Observer, оскільки об'єкти інтерфейсу реагують на події та змінюють свій стан відповідно до них.

Серед найбільш поширених фреймворків для розроблення застосунків для персональних комп'ютерів можна виділити Qt, .NET (зокрема Windows Forms та WPF), JavaFX та інші. Кожен із них має власні особливості, мови програмування та підходи до організації програмного коду.

Фреймворк Qt є кросплатформеним рішенням, що дозволяє створювати застосунки для різних операційних систем. Він підтримує мову програмування C++ та використовує механізм сигналів і слотів для організації взаємодії між компонентами. Це дає змогу реалізувати гнучку та ефективну подієву модель.

Платформа .NET забезпечує створення настільних застосунків із використанням мов програмування C# та інших мов. Вона містить фреймворки Windows Forms і WPF, які надають засоби для створення графічного інтерфейсу та організації логіки програми. WPF,

зокрема, використовує декларативний підхід до опису інтерфейсу та підтримує сучасні можливості візуалізації.

JavaFX є фреймворком для створення настільних застосунків на мові Java. Він забезпечує створення графічного інтерфейсу, підтримує анімацію, оброблення подій та інтеграцію з іншими компонентами Java-платформи.

Фреймворки для настільних застосунків можуть використовувати різні архітектурні підходи, зокрема Model–View–Controller або Model–View–ViewModel. Це дає змогу забезпечити розподіл відповідальності між компонентами системи та підвищити якість програмного забезпечення.

Перевагами використання таких фреймворків є наявність готових компонентів інтерфейсу, підтримка подієвої моделі, можливість створення кросплатформених застосунків та інтеграція з операційною системою. Разом із тим, вони можуть мати складну структуру, вимагати часу для освоєння та накладати обмеження на реалізацію окремих функцій.

Аналіз фреймворків для розроблення застосунків для персональних комп'ютерів дає змогу зрозуміти їх можливості, визначити їх переваги та обмеження, а також обрати найбільш доцільний інструмент для створення програмного забезпечення.

Практичне завдання

1. Ознайомитися з фреймворками для розроблення настільних застосунків.
2. Обрати один або декілька фреймворків для аналізу.
3. Описати призначення обраного фреймворку.
4. Проаналізувати засоби створення графічного інтерфейсу користувача.
5. Дослідити механізми оброблення подій.
6. Визначити архітектурні підходи, що використовуються у фреймворку.
7. Проаналізувати можливості інтеграції з операційною системою.
8. Оцінити переваги використання фреймворку.
9. Визначити недоліки або обмеження.
10. Сформулювати висновки щодо ефективності використання фреймворків.

Методичні вказівки до виконання практичного завдання

Перед виконанням роботи необхідно ознайомитися з теоретичними відомостями щодо фреймворків для розроблення настільних застосунків та їх ролі у програмному забезпеченні.

На першому етапі необхідно обрати фреймворк або декілька фреймворків для аналізу. Доцільно використовувати ті, що застосовуються у курсі, наприклад Qt або .NET.

Далі потрібно визначити призначення обраного фреймворку. Слід пояснити, для яких задач він використовується та які можливості надає.

На наступному етапі необхідно проаналізувати засоби створення графічного інтерфейсу користувача. Слід визначити, які компоненти доступні та як вони використовуються у програмі.

Після цього потрібно дослідити механізми оброблення подій. Необхідно пояснити, як система реагує на дії користувача та як організовується взаємодія між компонентами.

Далі слід проаналізувати архітектурні підходи, що використовуються у фреймворку. Важливо визначити, як організовано розподіл відповідальності між компонентами системи.

На наступному етапі необхідно оцінити можливості інтеграції з операційною системою. Слід визначити, які ресурси системи доступні та як вони використовуються.

Після цього потрібно проаналізувати переваги та недоліки фреймворку. Важливо оцінити його ефективність у різних умовах використання.

На завершальному етапі необхідно сформулювати висновки. У висновках слід узагальнити результати аналізу та визначити доцільність використання фреймворку для розроблення застосунків для персональних комп'ютерів.

Контрольні питання

1. Які фреймворки використовуються для розроблення настільних застосунків?
2. Які можливості надають фреймворки для створення інтерфейсу користувача?
3. Що таке подієва модель у настільних застосунках?
4. Які архітектурні підходи використовуються у таких фреймворках?
5. Які особливості фреймворку Qt?
6. Які особливості платформи .NET для настільних застосунків?
7. Які можливості надає JavaFX?
8. Які переваги мають фреймворки для настільних застосунків?
9. Які недоліки або обмеження мають ці фреймворки?
10. Як обрати фреймворк для розроблення настільного застосунку?

Зміст звіту

Звіт з практичної роботи повинен містити такі структурні елементи:
– назву практичної роботи;
– мету виконання практичної роботи;
– опис обраного фреймворку;
– аналіз його можливостей;
– характеристику інтерфейсу користувача та оброблення подій;
– оцінку переваг та недоліків;
– висновки за результатами виконання практичної роботи;
– перелік використаних джерел інформації.

Практична робота 21. Аналіз фреймворків для розробки мобільних застосунків

Ключові положення

Фреймворки для розробки мобільних застосунків є програмними платформами, що забезпечують створення програмного забезпечення для мобільних пристроїв, таких як смартфони та планшети. Вони надають інструменти для побудови інтерфейсу користувача, управління станом застосунку, взаємодії з апаратними можливостями пристрою та оброблення подій. Використання таких фреймворків дає змогу значно спростити процес розроблення та забезпечити кросплатформеність програмних продуктів.

Сучасні мобільні фреймворки можна умовно поділити на нативні та кросплатформені. Нативні фреймворки орієнтовані на розроблення застосунків для конкретної операційної

системи, наприклад Android або iOS, і забезпечують максимальну продуктивність та доступ до всіх можливостей платформи. Кросплатформені фреймворки дають змогу створювати застосунки для декількох платформ на основі єдиного коду, що суттєво скорочує час розроблення.

Одним із найбільш поширених кросплатформених фреймворків є Flutter, який використовує мову програмування Dart і реалізує компонентний підхід до побудови інтерфейсу користувача. У Flutter інтерфейс формується із набору віджетів, що можуть комбінуватися між собою для створення складних структур. Це дає змогу забезпечити гнучкість та повторне використання компонентів.

Іншим популярним підходом є використання фреймворків, що базуються на вебтехнологіях, наприклад React Native. У цьому випадку інтерфейс користувача створюється із використанням JavaScript і компонентної моделі, а взаємодія з платформою здійснюється через спеціальні механізми інтеграції.

Мобільні фреймворки реалізують подієву модель взаємодії, у якій застосунок реагує на дії користувача, такі як натискання елементів інтерфейсу, введення даних або зміна стану системи. Такий підхід має ознаки патерну Observer і забезпечує гнучке управління поведінкою застосунку.

Важливою архітектурною особливістю мобільних фреймворків є управління станом застосунку. Це включає збереження даних, їх оновлення та синхронізацію між різними компонентами інтерфейсу. Для цього використовуються спеціальні механізми, що дозволяють забезпечити узгодженість даних та ефективне оновлення інтерфейсу.

Мобільні фреймворки також забезпечують доступ до апаратних можливостей пристрою, таких як камера, сенсори, мережеві інтерфейси та система зберігання даних. Це дає змогу створювати функціональні застосунки, що використовують можливості сучасних мобільних пристроїв.

Перевагами використання мобільних фреймворків є скорочення часу розроблення, можливість створення кросплатформених застосунків, наявність готових компонентів інтерфейсу та підтримка сучасних архітектурних підходів. Разом із тим, вони можуть мати обмеження щодо продуктивності, доступу до окремих функцій платформи та вимагати додаткових знань для ефективного використання.

Аналіз фреймворків для розробки мобільних застосунків дає змогу зрозуміти їх можливості, оцінити їх переваги та обмеження, а також обрати найбільш доцільний інструмент для створення програмного забезпечення.

Практичне завдання

1. Ознайомитися з фреймворками для розроблення мобільних застосунків.
2. Обрати один або декілька фреймворків для аналізу.
3. Описати призначення обраного фреймворку.
4. Проаналізувати підхід до побудови інтерфейсу користувача.
5. Дослідити механізми оброблення подій.
6. Визначити способи управління станом застосунку.
7. Проаналізувати можливості взаємодії з апаратними ресурсами пристрою.
8. Оцінити переваги використання фреймворку.
9. Визначити недоліки або обмеження.
10. Сформулювати висновки щодо ефективності використання фреймворків.

Методичні вказівки до виконання практичного завдання

Перед виконанням роботи необхідно ознайомитися з теоретичними відомостями щодо мобільних фреймворків та їх ролі у сучасному програмному забезпеченні.

На першому етапі необхідно обрати фреймворк або декілька фреймворків для аналізу. Доцільно використовувати сучасні та поширені рішення, що застосовуються у практиці розроблення мобільних застосунків.

Далі потрібно визначити призначення обраного фреймворку. Слід пояснити, для яких задач він використовується та які можливості надає.

На наступному етапі необхідно проаналізувати підхід до побудови інтерфейсу користувача. Слід визначити, які компоненти використовуються та як організовується їх взаємодія.

Після цього потрібно дослідити механізми оброблення подій. Необхідно пояснити, як застосунок реагує на дії користувача та як організовується взаємодія між компонентами.

Далі слід проаналізувати способи управління станом застосунку. Важливо визначити, як забезпечується узгодженість даних та їх оновлення.

На наступному етапі необхідно оцінити можливості взаємодії з апаратними ресурсами пристрою. Слід визначити, які функції доступні та як вони використовуються.

Після цього потрібно проаналізувати переваги та недоліки фреймворку. Важливо оцінити його ефективність у різних умовах використання.

На завершальному етапі необхідно сформулювати висновки. У висновках слід узагальнити результати аналізу та визначити доцільність використання фреймворку для розроблення мобільних застосунків.

Контрольні питання

1. Які фреймворки використовуються для розроблення мобільних застосунків?
2. Які відмінності між нативними та кросплатформеними фреймворками?
3. Які особливості фреймворку Flutter?
4. Які особливості React Native?
5. Як організовується інтерфейс користувача у мобільних фреймворках?
6. Що таке подієва модель у мобільних застосунках?
7. Як здійснюється управління станом застосунку?
8. Які можливості взаємодії з апаратними ресурсами надають фреймворки?
9. Які переваги мають мобільні фреймворки?
10. Які недоліки або обмеження мають ці фреймворки?

Зміст звіту

Звіт з практичної роботи повинен містити такі структурні елементи:
– назву практичної роботи;
– мету виконання практичної роботи;
– опис обраного фреймворку;
– аналіз його архітектурних особливостей;
– характеристику інтерфейсу користувача та оброблення подій;

– аналіз управління станом застосунку;
– оцінку переваг та недоліків;
– висновки за результатами виконання практичної роботи;
– перелік використаних джерел інформації.

Практична робота 22. Аналіз фреймворків для розробки вебзастосунків

Ключові положення

Фреймворки для розробки вебзастосунків є основою сучасних інформаційних систем, що функціонують у мережі Інтернет. Вони забезпечують створення серверної та клієнтської частин вебзастосунків, організовують оброблення запитів, взаємодію з базами даних та формування інтерфейсу користувача. Використання таких фреймворків дає змогу стандартизувати процес розроблення та значно скоротити час створення програмного продукту.

Сучасні вебфреймворки можна поділити на серверні та клієнтські. Серверні фреймворки відповідають за оброблення HTTP-запитів, реалізацію бізнес-логіки та взаємодію з базами даних. Клієнтські фреймворки забезпечують побудову інтерфейсу користувача та управління станом вебзастосунку у браузері.

До серверних фреймворків належать Django, Flask та інші. Django є повнофункціональним фреймворком, що містить засоби маршрутизації, роботи з базами даних, аутентифікації та адміністративного управління. Flask є більш легким фреймворком, що надає базові можливості та дозволяє розширювати функціональність за допомогою додаткових компонентів.

До клієнтських фреймворків належать Angular, React та Vue. Вони реалізують компонентний підхід до побудови інтерфейсу користувача та забезпечують динамічне оновлення сторінки без повного перезавантаження. Це дозволяє створювати інтерактивні вебзастосунки з високим рівнем зручності для користувача.

Вебфреймворки зазвичай реалізують архітектурні підходи, такі як Model–View–Controller або його варіації. Це забезпечує розподіл відповідальності між компонентами системи, що спрощує супроводження та розвиток програмного забезпечення.

Важливою складовою вебфреймворків є механізм маршрутизації, який визначає, як обробляються запити користувача та які компоненти відповідають за їх виконання. Також важливими є засоби шаблонізації, що дозволяють відокремити логіку відображення від бізнес-логіки, та механізми роботи з базами даних.

Перевагами вебфреймворків є швидкість розроблення, наявність готових компонентів, підтримка сучасних стандартів та можливість створення масштабованих систем. Разом із тим, вони можуть накладати обмеження на структуру застосунку та вимагати значних зусиль для освоєння.

Аналіз вебфреймворків дає змогу зрозуміти принципи побудови вебзастосунків, оцінити можливості різних підходів та обрати оптимальні технології для реалізації програмних систем.

Практичне завдання

1. Ознайомитися з вебфреймворками та їх призначенням.
2. Обрати один або декілька фреймворків для аналізу.
3. Описати призначення обраного фреймворку.
4. Проаналізувати архітектуру вебзастосунку, що використовується у фреймворку.
5. Дослідити механізми маршрутизації та оброблення запитів.
6. Проаналізувати засоби роботи з базами даних.
7. Описати підхід до побудови інтерфейсу користувача.
8. Оцінити переваги використання фреймворку.
9. Визначити недоліки або обмеження.
10. Сформулювати висновки щодо ефективності використання вебфреймворків.

Методичні вказівки до виконання практичного завдання

Перед виконанням роботи необхідно повторити теоретичний матеріал щодо вебзастосунків, архітектурних підходів та ролі фреймворків у їх створенні.

На першому етапі необхідно обрати фреймворк або декілька фреймворків для аналізу. Доцільно використовувати ті, що застосовуються у курсі, наприклад Django, Flask, Angular або React.

Далі потрібно визначити призначення обраного фреймворку. Слід пояснити, для яких задач він використовується та які можливості надає.

На наступному етапі необхідно проаналізувати архітектуру вебзастосунку. Слід визначити, які компоненти входять до його складу та як вони взаємодіють між собою.

Після цього потрібно дослідити механізми маршрутизації. Необхідно пояснити, як обробляються запити користувача та як визначається відповідний обробник.

Далі слід проаналізувати засоби роботи з базами даних. Важливо визначити, як організовано доступ до даних та їх оброблення.

На наступному етапі необхідно описати підхід до побудови інтерфейсу користувача. Слід визначити, які інструменти використовуються для формування представлення та взаємодії з користувачем.

Після цього потрібно оцінити переваги та недоліки фреймворку. Важливо визначити, у яких умовах його використання є доцільним.

На завершальному етапі необхідно сформулювати висновки. У висновках слід узагальнити результати аналізу та визначити роль вебфреймворків у сучасному програмному забезпеченні.

Контрольні питання

1. Які фреймворки використовуються для розроблення вебзастосунків?
2. Які відмінності між серверними та клієнтськими фреймворками?
3. Які особливості фреймворку Django?
4. Які особливості Flask?
5. Які особливості Angular?
6. Які особливості React?
7. Що таке маршрутизація у вебзастосунках?

8. Як організовується робота з базами даних у вебфреймворках?
9. Які переваги мають вебфреймворки?
10. Які недоліки або обмеження мають вебфреймворки?

Зміст звіту

Звіт з практичної роботи повинен містити такі структурні елементи:
– назву практичної роботи;
– мету виконання практичної роботи;
– опис обраного фреймворку;
– аналіз його архітектурних особливостей;
– характеристику механізмів маршрутизації та оброблення запитів;
– аналіз роботи з базами даних;
– оцінку переваг та недоліків;
– висновки за результатами виконання практичної роботи;
– перелік використаних джерел інформації.

ТЕМА 8. ФРЕЙМВОРКИ ДЛЯ РОЗРОБКИ ЗАСТОСУНКІВ ДЛЯ ПЕРСОНАЛЬНИХ КОМП'ЮТЕРІВ

Практична робота 23. Розробка застосунку за допомогою фреймворку Qt

Ключові положення

Фреймворк Qt є кросплатформеним інструментом для розроблення застосунків із графічним інтерфейсом користувача. Він підтримує створення програм для різних операційних систем, зокрема Windows, Linux та macOS, і забезпечує єдину модель розроблення незалежно від платформи. Основною мовою програмування у Qt є C++, проте також можливе використання інших мов через відповідні засоби інтеграції.

Однією з ключових особливостей Qt є наявність великої кількості готових компонентів інтерфейсу користувача. До них належать вікна, кнопки, текстові поля, списки, таблиці та інші елементи, що дають змогу створювати повноцінні графічні застосунки. Це дозволяє значно скоротити час розроблення та зосередитися на реалізації функціональності програми.

Qt використовує подієву модель програмування, у якій застосунок реагує на дії користувача та системні події. Для організації взаємодії між компонентами застосовується механізм сигналів і слотів. Сигнали генеруються об'єктами при настанні певних подій, а слоти є методами, що обробляють ці події. Такий підхід забезпечує слабке зв'язування між компонентами та підвищує гнучкість системи.

Фреймворк Qt підтримує різні архітектурні підходи, зокрема Model–View–Controller та Model–View–ViewModel. Це дає змогу забезпечити розподіл відповідальності між компонентами застосунку, що спрощує його супроводження та розвиток.

Ще однією важливою особливістю Qt є підтримка роботи з різними ресурсами системи, такими як файлові системи, мережеві з'єднання, бази даних та мультимедійні компоненти. Це дозволяє створювати функціонально насичені застосунки для різних сфер використання.

Перевагами Qt є кросплатформеність, наявність розвиненої бібліотеки компонентів, підтримка об'єктно-орієнтованого програмування та ефективна подієва модель. Разом із тим, освоєння Qt може вимагати значних зусиль, особливо через специфічні механізми, такі як сигнали і слоти, а також використання додаткових інструментів для збірки проєктів.

Розробка застосунку за допомогою Qt дає змогу засвоїти принципи створення графічних інтерфейсів, організації подієвої взаємодії та використання сучасних фреймворків у процесі розроблення програмного забезпечення.

Практичне завдання

1. Ознайомитися з основними можливостями фреймворку Qt.
2. Встановити середовище розробки Qt Creator або використати Visual Studio Code з відповідними інструментами.
3. Створити новий проєкт застосунку.
4. Реалізувати графічний інтерфейс користувача із використанням базових компонентів.

5. Додати оброблення подій за допомогою механізму сигналів і слотів.
6. Реалізувати просту функціональність застосунку.
7. Виконати тестування роботи програми.
8. Проаналізувати структуру створеного застосунку.
9. Оцінити переваги використання фреймворку Qt.
10. Сформулювати висновки за результатами виконання роботи.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно встановити середовище розробки Qt Creator або налаштувати інші інструменти для роботи з фреймворком Qt. Необхідно переконатися у коректності встановлення компілятора та бібліотек Qt.

На першому етапі потрібно створити новий проєкт застосунку. Доцільно використати шаблон застосунку з графічним інтерфейсом. Після створення проєкту слід ознайомитися зі структурою файлів та основними компонентами програми.

Далі необхідно реалізувати графічний інтерфейс користувача. Це можна зробити за допомогою візуального редактора або шляхом написання коду. Слід додати основні елементи інтерфейсу, такі як кнопки, текстові поля або інші компоненти, що відповідають поставленій задачі.

Наступним етапом є реалізація оброблення подій. Для цього потрібно використати механізм сигналів і слотів. Необхідно пов’язати події елементів інтерфейсу з відповідними методами, що реалізують логіку програми.

Після цього слід реалізувати функціональність застосунку. Це може бути виконання обчислень, оброблення введених даних або інші дії, що демонструють роботу програми.

На етапі тестування необхідно перевірити коректність роботи застосунку. Слід переконатися, що всі елементи інтерфейсу працюють належним чином, а оброблення подій виконується правильно.

Після завершення реалізації необхідно проаналізувати структуру застосунку. Слід визначити, як організовано код, які компоненти використовуються та як вони взаємодіють між собою.

У звіті необхідно описати процес створення застосунку, показати основні фрагменти коду та зробити висновки щодо ефективності використання фреймворку Qt.

Контрольні питання

1. Що таке фреймворк Qt?
2. Які основні можливості надає Qt?
3. Що таке подієва модель програмування?
4. Що таке сигнали і слоти у Qt?
5. Які компоненти інтерфейсу доступні у Qt?
6. Які архітектурні підходи підтримує Qt?
7. Які переваги має використання Qt?
8. Які недоліки має цей фреймворк?
9. Як створити новий проєкт у Qt?
10. Як реалізувати оброблення подій у Qt?

Зміст звіту

Звіт з практичної роботи повинен містити такі структурні елементи:
– назву практичної роботи;
– мету виконання практичної роботи;
– опис фреймворку Qt;
– опис створеного застосунку;
– основні фрагменти програмної реалізації;
– результати тестування;
– аналіз переваг та недоліків використання Qt;
– висновки за результатами виконання практичної роботи;
– перелік використаних джерел інформації.

Практична робота 24. Розробка застосунку за допомогою фреймворку .NET

Ключові положення

Фреймворк .NET є програмною платформою, що забезпечує створення різних типів застосунків, зокрема настільних, вебзастосунків і сервісів. Він підтримує кілька мов програмування, серед яких найбільш поширеною є C#. Платформа містить середовище виконання, бібліотеки класів і засоби для розроблення, налагодження та тестування програмного забезпечення.

Однією з ключових особливостей .NET є наявність спільного середовища виконання, яке забезпечує керування пам’яттю, оброблення винятків, безпеку та виконання програмного коду. Це дозволяє розробникам зосередитися на реалізації логіки застосунку без необхідності керування низькорівневими аспектами роботи системи.

Для розроблення настільних застосунків у .NET використовуються технології Windows Forms та Windows Presentation Foundation. Windows Forms забезпечує швидке створення графічного інтерфейсу користувача із використанням готових компонентів. Windows Presentation Foundation надає більш сучасні можливості для створення інтерфейсу, зокрема підтримку декларативного опису інтерфейсу та розширені засоби візуалізації.

Фреймворк .NET підтримує подієву модель програмування, у якій застосунок реагує на дії користувача та системні події. Оброблення подій реалізується за допомогою делегатів і подій, що забезпечує гнучку взаємодію між компонентами програми.

У .NET широко використовується архітектурний підхід Model–View–Controller або Model–View–ViewModel. Це дає змогу розділити логіку застосунку, інтерфейс користувача та оброблення даних, що спрощує супроводження та розвиток програмного забезпечення.

Платформа .NET забезпечує засоби роботи з файлами, мережевими ресурсами, базами даних, багатопотоковістю та іншими компонентами системи. Це дозволяє створювати функціонально насичені застосунки для різних сфер використання.

Перевагами .NET є висока продуктивність, наявність великої бібліотеки класів, підтримка сучасних технологій розроблення та інтеграція з операційною системою. Разом із тим, використання цієї платформи може вимагати значних ресурсів та часу для освоєння.

Розробка застосунку за допомогою .NET дає змогу засвоїти принципи створення графічних інтерфейсів, організації подієвої взаємодії та використання сучасних програмних платформ у процесі розроблення програмного забезпечення.

Практичне завдання

1. Ознайомитися з основними можливостями платформи .NET.
2. Встановити середовище розробки (наприклад, Visual Studio або Visual Studio Code).
3. Створити новий проєкт настільного застосунку.
4. Реалізувати графічний інтерфейс користувача із використанням базових компонентів.
5. Додати оброблення подій у застосунку.
6. Реалізувати функціональність програми.
7. Виконати тестування роботи застосунку.
8. Проаналізувати структуру створеного застосунку.
9. Оцінити переваги використання платформи .NET.
10. Сформулювати висновки за результатами виконання роботи.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно встановити середовище розробки для роботи з платформою .NET. Доцільно використовувати Visual Studio або Visual Studio Code із відповідними розширеннями. Необхідно переконатися у коректності встановлення компілятора та бібліотек.

На першому етапі потрібно створити новий проєкт застосунку. Доцільно обрати шаблон настільного застосунку, наприклад Windows Forms або WPF. Після створення проєкту слід ознайомитися зі структурою файлів і основними компонентами програми.

Далі необхідно реалізувати графічний інтерфейс користувача. Слід використати стандартні елементи інтерфейсу, такі як кнопки, текстові поля, списки та інші компоненти. Можна використовувати візуальний редактор або створювати інтерфейс програмним шляхом.

Наступним етапом є реалізація оброблення подій. Необхідно визначити події, що будуть оброблятися, та реалізувати відповідні обробники. Наприклад, це може бути реакція на натискання кнопки або введення даних.

Після цього слід реалізувати функціональність застосунку. Це може бути виконання обчислень, оброблення даних або інші дії, що демонструють роботу програми.

На етапі тестування необхідно перевірити коректність роботи застосунку. Слід переконатися, що всі елементи інтерфейсу працюють належним чином, а оброблення подій виконується правильно.

Після завершення реалізації необхідно проаналізувати структуру застосунку. Слід визначити, як організовано код, які компоненти використовуються та як вони взаємодіють між собою.

У звіті необхідно описати процес створення застосунку, показати основні фрагменти коду та зробити висновки щодо ефективності використання платформи .NET.

Контрольні питання

1. Що таке платформа .NET?
2. Які можливості надає .NET?
3. Що таке спільне середовище виконання?

4. Які технології використовуються для створення настільних застосунків у .NET?
5. Що таке подієва модель програмування?
6. Як реалізується оброблення подій у .NET?
7. Які архітектурні підходи використовуються у .NET?
8. Які переваги має використання платформи .NET?
9. Які недоліки має цей підхід?
10. Як створити застосунок у середовищі .NET?

Зміст звіту

Звіт з практичної роботи повинен містити такі структурні елементи:
– назву практичної роботи;
– мету виконання практичної роботи;
– опис платформи .NET;
– опис створеного застосунку;
– основні фрагменти програмної реалізації;
– результати тестування;
– аналіз переваг та недоліків використання .NET;
– висновки за результатами виконання практичної роботи;
– перелік використаних джерел інформації.

ТЕМА 9. ФРЕЙМВОРКИ ДЛЯ РОЗРОБКИ МОБІЛЬНИХ ЗАСТОСУНКІВ

Практична робота 25. Розробка застосунку за допомогою фреймворку Flutter

Ключові положення

Фреймворк Flutter є сучасним засобом для розроблення кросплатформених застосунків, що дає змогу створювати програми для мобільних пристроїв, персональних комп'ютерів і вебсередовища на основі єдиної кодової бази. Він розроблений компанією Google та використовує мову програмування Dart. Основною особливістю Flutter є компонентний підхід до побудови інтерфейсу користувача, при якому весь інтерфейс формується з окремих віджетів.

У Flutter кожен елемент інтерфейсу є віджетом. Це можуть бути як прості елементи, наприклад текст, кнопка або поле введення, так і складні компоненти, що поєднують кілька інших віджетів. Такий підхід дає змогу будувати інтерфейс застосунку у вигляді ієрархічної структури, що забезпечує гнучкість, повторне використання коду та зручність модифікації окремих частин програми.

Однією з важливих переваг Flutter є можливість створення кросплатформених застосунків без необхідності окремої реалізації для кожної операційної системи. Розробник створює одну програму, яка може бути адаптована для різних платформ. Це скорочує час розроблення, спрощує супроводження програмного забезпечення та дає змогу повторно використовувати значну частину коду.

Flutter використовує власний механізм візуалізації інтерфейсу, що забезпечує однаковий вигляд елементів на різних платформах. На відміну від підходів, де застосунок використовує нативні компоненти операційної системи, Flutter самостійно відтворює інтерфейс за допомогою графічного рушія. Це дозволяє досягти високої узгодженості вигляд та поведінки застосунку.

Важливою складовою Flutter є підтримка подієвої моделі програмування. Застосунок реагує на дії користувача, зокрема натискання кнопок, введення тексту, переміщення по екранах або зміну стану елементів інтерфейсу. Для побудови логіки програми у Flutter застосовуються механізми керування станом, що забезпечують оновлення інтерфейсу при зміні даних.

У Flutter виділяють статичні та динамічні віджети. StatelessWidget використовується для елементів, стан яких не змінюється після створення, а StatefulWidget – для елементів, що можуть змінювати свій стан у процесі виконання програми. Такий поділ дає змогу чітко організувати структуру застосунку та визначити, які компоненти потребують оновлення інтерфейсу.

Flutter містить велику кількість готових компонентів для побудови інтерфейсу, організації навігації між екранами, роботи з формами, списками, зображеннями, мережевими запитами та локальним збереженням даних. Це дає змогу швидко реалізувати повноцінний застосунок без необхідності створення базових елементів з нуля.

Ще однією важливою перевагою Flutter є підтримка гарячого перезавантаження. Цей механізм дає змогу швидко переглядати результати змін у коді без повного перезапуску застосунку. У навчальному процесі це є особливо корисним, оскільки спрощує експериментування з інтерфейсом і логікою програми.

Разом із перевагами Flutter має і певні обмеження. Зокрема, для ефективної роботи з ним необхідно освоїти мову Dart, зрозуміти підходи до побудови інтерфейсу на основі віджетів та опанувати механізми керування станом. Крім того, у складних проєктах важливо раціонально організовувати структуру віджетів, щоб уникати надмірної складності коду.

Розробка застосунку за допомогою Flutter дає змогу засвоїти сучасні підходи до побудови кросплатформених програм, навчитися створювати інтерфейс користувача з використанням віджетів, реалізовувати подієву взаємодію та організовувати логіку програмного застосунку відповідно до сучасних вимог розроблення програмного забезпечення.

Практичне завдання

1. Ознайомитися з основними можливостями фреймворку Flutter.
2. Встановити Flutter SDK та налаштувати середовище розробки.
3. Створити новий проєкт застосунку.
4. Реалізувати графічний інтерфейс користувача із використанням базових віджетів.
5. Додати оброблення подій у застосунку.
6. Реалізувати просту функціональність програми.
7. Використати механізм оновлення стану інтерфейсу.
8. Виконати тестування роботи застосунку.
9. Проаналізувати структуру створеного застосунку.
10. Сформулювати висновки за результатами виконання роботи.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно встановити Flutter SDK, а також налаштувати середовище розробки, наприклад Visual Studio Code або Android Studio, із відповідними розширеннями для мови Dart і фреймворку Flutter. Потрібно переконатися у правильності встановлення інструментів і можливості запуску тестового застосунку.

На першому етапі слід створити новий проєкт Flutter. Після цього необхідно ознайомитися зі структурою проєкту, зокрема з основним файлом програми, каталогами ресурсів і службовими файлами конфігурації. Особливу увагу потрібно приділити файлу main.dart, оскільки саме у ньому зазвичай визначається точка входу до застосунку та початкова структура інтерфейсу.

Далі необхідно реалізувати графічний інтерфейс користувача. Доцільно створити простий застосунок, що містить базові елементи, наприклад текстові написи, кнопки, поля введення або області відображення результатів. При цьому слід використовувати стандартні віджети Flutter і забезпечити логічну структуру розміщення елементів на екрані.

Наступним етапом є реалізація подієвої взаємодії. Для цього необхідно визначити, які дії користувача буде обробляти програма, та реалізувати відповідні обробники подій. Наприклад, це може бути натискання кнопки, введення тексту або вибір значення. Усі ці дії повинні змінювати стан застосунку або викликати певну функціональність.

Після цього слід реалізувати оновлення стану інтерфейсу. Якщо програма містить дані, що змінюються під час виконання, доцільно використати StatefulWidget і механізм setState для оновлення екрана. Необхідно показати, як зміна внутрішніх даних відображається у візуальному інтерфейсі застосунку.

На етапі реалізації функціональності потрібно забезпечити виконання певного практичного завдання. Це може бути обчислення значень, відображення повідомлень, робота зі списком елементів або інша нескладна задача, що демонструє можливості Flutter щодо побудови інтерфейсу та організації логіки програми.

Після завершення програмної реалізації необхідно виконати тестування застосунку. Слід перевірити правильність відображення інтерфейсу, коректність оброблення подій, оновлення стану та загальну працездатність програми. За можливості доцільно перевірити роботу застосунку на емуляторі або реальному пристрої.

Після цього потрібно проаналізувати структуру створеного застосунку. У звіті слід визначити, які віджети були використані, як організовано логіку програми, яким чином реалізовано оброблення подій і оновлення стану. Також необхідно оцінити переваги використання Flutter для створення застосунків та зазначити можливі труднощі, що виникли під час виконання роботи.

Під час виконання практичної роботи слід дотримуватися зрозумілого стилю оформлення коду, використовувати змістовні назви змінних і віджетів, а також забезпечити логічну відповідність між структурою інтерфейсу та функціональністю програми.

Контрольні питання

1. Що таке фреймворк Flutter?
2. Яку мову програмування використовують у Flutter?
3. Що таке віджет у Flutter?
4. Чим відрізняються StatelessWidget і StatefulWidget?
5. Що таке подієва модель у Flutter?
6. Для чого використовується механізм setState?
7. Які переваги має кросплатформений підхід Flutter?
8. Які основні компоненти містить проєкт Flutter?
9. Які переваги дає гаряче перезавантаження?
10. Які труднощі можуть виникати при розробленні застосунків за допомогою Flutter?

Зміст звіту

Звіт з практичної роботи повинен містити такі структурні елементи:
– назву практичної роботи;
– мету виконання практичної роботи;
– короткі теоретичні відомості про фреймворк Flutter;
– опис створеного застосунку;
– характеристику використаних віджетів;
– основні фрагменти програмної реалізації;
– результати тестування;
– аналіз особливостей реалізації інтерфейсу та оброблення подій;
– висновки за результатами виконання практичної роботи;
– перелік використаних джерел інформації.

Практична робота 26. Розробка застосунку за допомогою фреймворку React Native

Ключові положення

Фреймворк React Native є сучасним засобом для розроблення мобільних застосунків, що дає змогу створювати програмне забезпечення для платформ Android та iOS на основі спільної кодової бази. Він побудований на основі бібліотеки React та використовує мову програмування JavaScript або TypeScript. Основною ідеєю React Native є компонентний підхід до побудови інтерфейсу користувача, за якого застосунок формується із сукупності незалежних і повторно використовуваних компонентів.

На відміну від звичайних вебзастосунків, у React Native для побудови інтерфейсу використовуються не HTML-елементи, а спеціальні компоненти, що відображаються як нативні елементи мобільної операційної системи. Це дає змогу створювати застосунки, які мають вигляд і поведінку, наближені до нативних програм. Завдяки цьому React Native поєднує переваги кросплатформеного підходу з можливістю створення зручного мобільного інтерфейсу.

Основою побудови застосунків у React Native є компоненти. Компонент є окремою частиною інтерфейсу, що може містити власну структуру, властивості та логіку роботи. Компоненти можуть бути простими, наприклад текстовий напис або кнопка, або складними, коли вони поєднують декілька інших компонентів і реалізують окремий функціональний фрагмент застосунку. Такий підхід дає змогу будувати інтерфейс як ієрархію взаємопов’язаних елементів.

Важливою особливістю React Native є використання механізму стану та властивостей. Властивості застосовуються для передавання даних до компонента, а стан використовується для зберігання змінних даних, які можуть змінюватися під час роботи програми. При зміні стану компонент автоматично оновлюється, що забезпечує динамічну взаємодію з користувачем та коректне відображення актуальних даних у застосунку.

React Native підтримує подієву модель програмування, у межах якої застосунок реагує на дії користувача, зокрема натискання кнопок, введення тексту, прокручування вмісту, перемикання екранів та інші події. Для цього використовуються обробники подій, які зв’язують інтерфейсні компоненти з програмною логікою. Такий підхід дає змогу реалізувати інтерактивні застосунки із гнучкою поведінкою.

Ще однією важливою складовою React Native є механізм навігації між екранами. Оскільки мобільний застосунок зазвичай містить декілька екранів, необхідно організувати перехід між ними та передавання даних. Для цього можуть використовуватися спеціальні бібліотеки навігації, які дають змогу реалізувати стекову, вкладену або комбіновану структуру переходів між екранами.

React Native забезпечує доступ до багатьох можливостей мобільного пристрою, таких як робота з мережею, збереження локальних даних, використання камери, геолокації, файлової системи та інших ресурсів. У базових навчальних прикладах зазвичай використовуються прості засоби взаємодії, проте сама платформа дозволяє створювати функціонально насичені мобільні застосунки.

Перевагами React Native є можливість повторного використання коду для різних платформ, компонентний підхід до побудови інтерфейсу, активна підтримка спільнотою та зручність інтеграції з сучасними інструментами JavaScript-розробки. Крім того, фреймворк

дає змогу швидко створювати прототипи та навчальні застосунки, що є важливим у межах освітнього процесу.

Разом із перевагами React Native має і певні обмеження. Для повноцінної розробки необхідно добре розуміти принципи роботи React, зокрема компоненти, властивості, стан та життєвий цикл. У складних проєктах також може виникати потреба у використанні додаткових бібліотек для навігації, керування станом та інтеграції з нативними можливостями пристрою. Проте для навчальних і прикладних задач цей фреймворк є цілком придатним засобом розроблення мобільних застосунків.

Розробка застосунку за допомогою React Native дає змогу засвоїти сучасні підходи до створення мобільного програмного забезпечення, навчитися будувати інтерфейс із компонентів, організовувати подієву взаємодію, працювати зі станом застосунку та реалізовувати базову логіку мобільної програми на основі кросплатформеного фреймворку.

Практичне завдання

1. Ознайомитися з основними можливостями фреймворку React Native.
2. Встановити середовище розробки та налаштувати інструменти для створення застосунку.
3. Створити новий проєкт React Native.
4. Реалізувати інтерфейс користувача із використанням базових компонентів.
5. Додати оброблення подій у застосунку.
6. Реалізувати просту функціональність програми.
7. Використати механізм стану для оновлення інтерфейсу.
8. За потреби реалізувати перехід між екранами.
9. Виконати тестування роботи застосунку.
10. Сформулювати висновки за результатами виконання роботи.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно встановити середовище розробки для React Native. Доцільно використовувати Visual Studio Code та відповідні інструменти для роботи з Node.js, npm або yarn, а також емулятор мобільного пристрою чи засоби запуску застосунку на реальному смартфоні. Необхідно переконатися у правильності налаштування середовища та можливості створення тестового проєкту.

На першому етапі потрібно створити новий проєкт React Native. Після цього слід ознайомитися зі структурою каталогу проєкту, файлами конфігурації та основним файлом застосунку. Особливу увагу потрібно приділити головному компоненту програми, оскільки саме в ньому зазвичай реалізується початкова структура інтерфейсу.

Далі необхідно реалізувати інтерфейс користувача. Для цього слід використати базові компоненти React Native, наприклад View, Text, TextInput, Button, Pressable або інші елементи, що відповідають змісту завдання. Інтерфейс повинен бути логічно організованим і достатнім для демонстрації основних можливостей застосунку.

Наступним етапом є реалізація подієвої взаємодії. Необхідно визначити, які дії користувача оброблятиме програма, наприклад натискання кнопки, введення тексту або вибір одного з елементів. Для кожної події слід створити відповідний обробник, який змінює стан програми або виконує необхідні дії.

Після цього потрібно реалізувати механізм стану. Якщо застосунок відображає дані, що змінюються під час роботи, необхідно використати засоби React для збереження стану компонента. Зміна стану повинна приводити до автоматичного оновлення інтерфейсу, що демонструє динамічний характер роботи програми.

За потреби у межах практичної роботи можна реалізувати перехід між двома або більше екранами. Це дає змогу показати, як у React Native організовується навігація у мобільному застосунку. У простішому варіанті робота може бути виконана в межах одного екрана без додаткової навігації.

На етапі реалізації функціональності слід забезпечити виконання нескладного практичного завдання. Це може бути застосунок для введення і відображення даних, калькулятор, список елементів, форма з перевіркою введення або інша програма, що дає змогу продемонструвати взаємодію компонентів, подій і стану.

Після завершення програмної реалізації необхідно протестувати застосунок. Слід перевірити правильність відображення інтерфейсу, коректність реакції на дії користувача, зміну стану та працездатність усіх основних функцій. За можливості доцільно перевірити роботу програми як на емуляторі, так і на реальному мобільному пристрої.

Після цього у звіті потрібно проаналізувати структуру створеного застосунку. Необхідно визначити, які компоненти були використані, як організовано подієву взаємодію, яким чином реалізовано зберігання стану та які особливості React Native проявилися під час виконання роботи. Також слід оцінити переваги використання цього фреймворку та зазначити труднощі, що виникли під час розроблення.

Під час виконання практичної роботи необхідно дотримуватися зрозумілого стилю оформлення коду, використовувати змістовні назви компонентів, змінних та функцій, а також забезпечувати логічну відповідність між побудовою інтерфейсу та функціональністю застосунку.

Контрольні питання

1. Що таке фреймворк React Native?
2. Яку мову програмування використовують у React Native?
3. У чому полягає компонентний підхід у React Native?
4. Чим відрізняються властивості від стану компонента?
5. Які базові компоненти інтерфейсу використовуються у React Native?
6. Що таке подієва модель у React Native?
7. Як реалізується оновлення інтерфейсу при зміні стану?
8. Для чого використовується навігація між екранами?
9. Які переваги має React Native як кросплатформений фреймворк?
10. Які труднощі можуть виникати під час розроблення застосунків за допомогою React Native?

Зміст звіту

Звіт з практичної роботи повинен містити такі структурні елементи:

– назву практичної роботи;

– мету виконання практичної роботи;

– короткі теоретичні відомості про фреймворк React Native;

– опис створеного застосунку;
– характеристику використаних компонентів;
– основні фрагменти програмної реалізації;
– результати тестування;
– аналіз особливостей реалізації інтерфейсу, подій і стану;
– висновки за результатами виконання практичної роботи;
– перелік використаних джерел інформації.

ТЕМА 10. ФРЕЙМВОРКИ ДЛЯ РОЗРОБКИ ВЕБЗАСТОСУНКІВ

Практична робота 27. Розробка застосунку за допомогою фреймворку Flask

Ключові положення

Фреймворк Flask є легким вебфреймворком для мови програмування Python, що використовується для створення серверної частини вебзастосунків. Він належить до так званих мікрофреймворків, оскільки містить лише базовий набір функціональності, необхідний для оброблення HTTP-запитів, маршрутизації та формування відповіді. Додаткові можливості, такі як робота з базами даних або аутентифікація, можуть підключатися за допомогою зовнішніх модулів.

Основою роботи Flask є оброблення HTTP-запитів клієнта. Користувач взаємодіє із застосунком через веббраузер, надсилаючи запити до сервера. Flask приймає ці запити, визначає відповідний обробник за допомогою механізму маршрутизації та формує відповідь, яка повертається клієнту. Такий підхід лежить в основі роботи більшості вебзастосунків.

Маршрутизація у Flask реалізується за допомогою декораторів, які зв'язують URL-адреси з функціями оброблення. Кожна така функція виконує певну логіку, наприклад обробляє введені дані, виконує обчислення або взаємодіє з базою даних, і повертає результат у вигляді HTML-сторінки або інших форматів даних.

Flask підтримує використання шаблонів для формування інтерфейсу користувача. Для цього застосовується шаблонізатор Jinja2, що дає змогу відокремити логіку програми від представлення даних. Шаблони дозволяють створювати динамічні вебсторінки, у яких вміст формується на основі даних, отриманих від сервера.

Важливою особливістю Flask є його простота та гнучкість. Розробник має можливість самостійно визначати структуру застосунку, підключати лише необхідні компоненти та реалізовувати логіку програми відповідно до вимог проєкту. Це робить Flask зручним для навчальних задач, прототипування та створення невеликих вебзастосунків.

Разом із тим, Flask не нав'язує жорсткої архітектури, що може бути як перевагою, так і недоліком. У великих проєктах відсутність стандартизованої структури може ускладнювати організацію коду та його супроводження. Тому у таких випадках необхідно самостійно впроваджувати архітектурні підходи, зокрема багаторівневу архітектуру або поділ на модулі.

Flask забезпечує можливості для роботи з формами, оброблення запитів різних типів, передачі даних між клієнтом і сервером, а також інтеграції з базами даних за допомогою додаткових бібліотек. Це дає змогу створювати повноцінні вебзастосунки, що відповідають сучасним вимогам.

Розробка застосунку за допомогою Flask дає змогу засвоїти основи створення вебзастосунків, навчитися працювати з HTTP-запитами, реалізовувати маршрутизацію, формувати динамічні сторінки та організовувати взаємодію між клієнтською та серверною частинами програмної системи.

Практичне завдання

1. Ознайомитися з основними можливостями фреймворку Flask.
2. Встановити Python та необхідні бібліотеки для роботи з Flask.

3. Створити новий вебзастосунок.
4. Реалізувати маршрутизацію для оброблення запитів.
5. Створити шаблон HTML-сторінки для відображення даних.
6. Реалізувати оброблення введених користувачем даних.
7. Додати просту функціональність застосунку.
8. Виконати тестування роботи вебзастосунку.
9. Проаналізувати структуру створеного застосунку.
10. Сформулювати висновки за результатами виконання роботи.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно встановити Python та пакет Flask за допомогою менеджера пакетів. Доцільно використовувати віртуальне середовище для ізоляції залежностей проєкту. Після встановлення необхідно перевірити працездатність середовища, створивши простий тестовий застосунок.

На першому етапі потрібно створити базовий файл застосунку. У ньому необхідно ініціалізувати об’єкт застосунку Flask та визначити перший маршрут, який буде відповідати за оброблення запитів до головної сторінки.

Далі необхідно реалізувати маршрутизацію. Для цього слід створити декілька маршрутів, кожен з яких виконує певну функцію. Наприклад, один маршрут може відображати головну сторінку, інший – обробляти дані, введені користувачем.

Наступним етапом є створення шаблонів HTML. Для цього потрібно створити окрему папку з шаблонами та використати шаблонізатор для формування динамічних сторінок. Необхідно показати, як передаються дані з сервера до шаблону та відображаються у браузері.

Після цього потрібно реалізувати оброблення даних, введених користувачем. Це може бути форма, у яку користувач вводить значення, що передаються на сервер і обробляються програмою. Результат оброблення повинен відображатися на сторінці.

На етапі реалізації функціональності доцільно створити простий вебзастосунок, наприклад калькулятор, форму оброблення даних або систему відображення інформації. Це дозволяє продемонструвати взаємодію між клієнтом і сервером.

Після завершення програмної реалізації необхідно протестувати застосунок. Слід перевірити правильність роботи маршрутів, коректність оброблення даних та відображення результатів у браузері.

Після цього у звіті потрібно проаналізувати структуру створеного застосунку. Необхідно описати основні компоненти, маршрути, шаблони та логіку оброблення запитів. Також слід оцінити переваги використання Flask та зазначити труднощі, що виникли під час виконання роботи.

Під час виконання практичної роботи необхідно дотримуватися зрозумілого стилю оформлення коду, використовувати змістовні назви змінних і функцій, а також забезпечувати логічну відповідність між структурою застосунку та його функціональністю.

Контрольні питання

1. Що таке фреймворк Flask?
2. Які особливості має Flask як мікрофреймворк?
3. Що таке маршрутизація у Flask?

4. Як реалізується оброблення HTTP-запитів?
5. Для чого використовуються шаблони у Flask?
6. Що таке Jinja2?
7. Як організовується оброблення даних користувача?
8. Які переваги має Flask?
9. Які недоліки має цей фреймворк?
10. У яких випадках доцільно використовувати Flask?

Зміст звіту

Звіт з практичної роботи повинен містити такі структурні елементи:
– назву практичної роботи;
– мету виконання практичної роботи;
– короткі теоретичні відомості про фреймворк Flask;
– опис створеного вебзастосунку;
– характеристику маршрутів і шаблонів;
– основні фрагменти програмної реалізації;
– результати тестування;
– аналіз особливостей реалізації;
– висновки за результатами виконання практичної роботи;
– перелік використаних джерел інформації.

Практична робота 28. Розробка застосунку за допомогою фреймворку Django

Ключові положення

Фреймворк Django є повнофункціональним вебфреймворком для мови програмування Python, що використовується для розроблення серверної частини вебзастосунків. На відміну від мікрофреймворків, Django надає широкий набір вбудованих засобів, що охоплюють маршрутизацію, роботу з базами даних, аутентифікацію, адміністративний інтерфейс та інші компоненти, необхідні для створення повноцінних вебсистем.

Основою архітектури Django є підхід Model–View–Template, що є варіацією архітектури Model–View–Controller. У цій моделі модель відповідає за роботу з даними та їх структуру, представлення визначає логіку оброблення запитів і формування відповіді, а шаблон відповідає за відображення даних у вигляді HTML-сторінок. Такий підхід дає змогу чітко розділити відповідальність між компонентами системи.

Django реалізує механізм маршрутизації, який визначає, як обробляються HTTP-запити. Для кожного URL-адресу визначається відповідна функція або клас, що обробляє запит і повертає результат. Це забезпечує гнучкість у побудові структури вебзастосунку та дає змогу легко організовувати взаємодію між клієнтом і сервером.

Важливою особливістю Django є наявність вбудованої системи роботи з базами даних. Фреймворк використовує об’єктно-реляційний механізм, що дозволяє працювати з даними у вигляді об’єктів, а не безпосередньо з SQL-запитами. Це спрощує розроблення та підвищує безпеку програмного забезпечення.

Django також містить потужну систему шаблонів, що дозволяє створювати динамічні вебсторінки. Шаблони дають змогу відокремити логіку відображення від бізнес-логіки, що спрощує супроводження та розвиток застосунку.

Ще однією важливою складовою Django є адміністративний інтерфейс, який генерується автоматично на основі моделей даних. Це дає змогу швидко створювати інструменти для керування даними без додаткового програмування.

Фреймворк Django забезпечує високий рівень безпеки, зокрема захист від поширених атак, таких як SQL-ін'єкції, міжсайтові запити та інші загрози. Це робить його придатним для створення надійних вебзастосунків.

Перевагами Django є наявність великої кількості вбудованих компонентів, стандартизована структура проєкту, висока продуктивність розроблення та підтримка спільнотою. Разом із тим, для невеликих проєктів цей фреймворк може бути надмірним через свою складність і обсяг.

Розробка застосунку за допомогою Django дає змогу засвоїти принципи створення вебзастосунків, навчитися працювати з базами даних, реалізовувати маршрутизацію, створювати динамічні сторінки та організовувати структуру програмного проєкту відповідно до сучасних підходів.

Практичне завдання

1. Ознайомитися з основними можливостями фреймворку Django.
2. Встановити Python та необхідні інструменти для роботи з Django.
3. Створити новий проєкт Django.
4. Створити додаток у межах проєкту.
5. Реалізувати модель даних.
6. Налаштувати маршрутизацію для оброблення запитів.
7. Створити шаблони для відображення даних.
8. Реалізувати оброблення запитів користувача.
9. Виконати тестування роботи вебзастосунку.
10. Сформулювати висновки за результатами виконання роботи.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно встановити Python та фреймворк Django. Доцільно використовувати віртуальне середовище для ізоляції залежностей проєкту. Після встановлення слід перевірити працездатність середовища, створивши тестовий проєкт.

На першому етапі потрібно створити новий проєкт Django за допомогою відповідної команди. Після цього необхідно ознайомитися зі структурою проєкту, зокрема з файлами налаштувань, маршрутизації та запуску застосунку.

Далі слід створити додаток у межах проєкту. Додаток є окремим модулем, що реалізує певну функціональність. Після створення додатка необхідно підключити його до основного проєкту.

Наступним етапом є реалізація моделі даних. Для цього потрібно визначити структуру даних, що буде використовуватися у застосунку, та описати її у вигляді класів. Після цього необхідно виконати міграції для створення відповідних таблиць у базі даних.

Після цього слід налаштувати маршрутизацію. Необхідно визначити URL-адреси, що будуть оброблятися застосунком, та зв’язати їх із відповідними функціями або класами.

Далі потрібно створити шаблони для відображення даних. Шаблони повинні містити HTML-структуру та спеціальні конструкції для вставлення даних. Необхідно показати, як дані передаються з сервера до шаблону та відображаються у браузері.

На наступному етапі необхідно реалізувати оброблення запитів користувача. Це може бути відображення списку об’єктів, оброблення форм або виконання інших дій, що демонструють роботу застосунку.

Після завершення програмної реалізації необхідно протестувати застосунок. Слід перевірити правильність роботи маршрутів, коректність відображення даних та загальну працездатність програми.

Після цього у звіті потрібно проаналізувати структуру створеного застосунку. Необхідно описати моделі, маршрути, представлення та шаблони, а також оцінити переваги використання Django.

Під час виконання практичної роботи необхідно дотримуватися зрозумілого стилю оформлення коду, використовувати змістовні назви класів і функцій, а також забезпечувати логічну відповідність між структурою застосунку та його функціональністю.

Контрольні питання

1. Що таке фреймворк Django?
2. Які основні можливості надає Django?
3. Що таке архітектура Model–View–Template?
4. Як реалізується маршрутизація у Django?
5. Що таке модель у Django?
6. Як організовується робота з базами даних?
7. Що таке шаблони у Django?
8. Які переваги має Django?
9. Які недоліки має цей фреймворк?
10. У яких випадках доцільно використовувати Django?

Зміст звіту

Звіт з практичної роботи повинен містити такі структурні елементи:
– назву практичної роботи;
– мету виконання практичної роботи;
– короткі теоретичні відомості про фреймворк Django;
– опис створеного вебзастосунку;
– характеристику моделей, маршрутів і шаблонів;
– основні фрагменти програмної реалізації;
– результати тестування;
– аналіз особливостей реалізації;
– висновки за результатами виконання практичної роботи;
– перелік використаних джерел інформації.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

Основна

1. Freeman E., Robson E. Head First Design Patterns: A Brain-Friendly Guide. 2nd ed. O'Reilly Media, 2020. 672 p. ISBN: 9781492077954, 149207795X.
2. Richards M., Ford N. Fundamentals of Software Architecture: An Engineering Approach. O'Reilly Media, 2020. 432 p. ISBN: 9781492043423, 1492043427.
3. Newman, S. Building Microservices. O'Reilly Media. 2021. 616 p. ISBN: 9781492033998, 1492033995
4. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2017. 432 p. ISBN: 9780134494326, 0134494326.
5. Kleppmann M. Designing Data-Intensive Applications. O'Reilly Media, 2017. 616 p. ISBN: 9781491903100, 1491903104.

Допоміжна

6. Fowler M. Refactoring: Improving the Design of Existing Code. 2nd ed. Addison-Wesley Professional, 2018. 448 p. ISBN: 9780134757704, 013475770X.
7. Hunt A., Thomas D. The Pragmatic Programmer: Your Journey to Mastery. 2nd ed. Addison-Wesley Professional, 2019. 352 p. ISBN: 9780135956915, 0135956919.
8. Vernon V. Domain-Driven Design. MTM, 2023. ISBN: 9789130090075, 9130090075.
9. Ford N., Richards M., Sadalage P., Dehghani Z. Software Architecture: The Hard Parts. O'Reilly Media, 2021. 462 p. ISBN: 9781492086864, 149208686X.
10. Burns B., Beda J., Hightower K. Kubernetes: Up and Running. 3rd ed. O'Reilly Media, 2022. 328 p. ISBN: 9781098110178, 109811017X.

Інформаційні ресурси

11. Gamma E. et al. Design Patterns Catalog. URL: https://refactoring.guru/design-patterns
12. Microsoft Developer Documentation. URL: https://learn.microsoft.com
13. Mozilla Developer Network (MDN Web Docs). URL: https://developer.mozilla.org
14. Qt Documentation. URL: https://doc.qt.io
15. Flutter Documentation. URL: https://docs.flutter.dev

Навчальне видання

Сергій Борисович Приходько

ПАТЕРНИ ТА ФРЕЙМВОРКИ

Методичні вказівки до виконання практичних робіт для здобувачів вищої освіти, які навчаються за спеціальностями галузі «Інформаційні технології»

Українською мовою