***Кафедра комп’ютерної та програмної інженерії***

**ЛАБОРАТОРНА РОБОТА № 1**

**з курсу**

**«Архітектура програмної платформи .NET»**

**Студент: Бондарєва Є.Ю.**

**Група: № 32 ІПЗ**

**Факультет: Математики, інформатики**

**та фізики**

**Перевірив: Січкаренко В.О.**

Тема: Підготовка необхідних робочих інструментів.

Мета: Ознайомитись та підготувати інструменти під власні потреби, які використовуються для розробки програмного забезпечення (ПЗ).

Загальні завдання

1. Ознайомитися з теоретичною частиною.

2. Створити обліковий запис в GitHub за посиланням вказаному в джерелі ;

3. Встановити систему контролю версій Git;

4. Створити обліковий запис в Figma за посиланням вказаному в джерелі;

5. Встановити (в залежності від використовуваної ОС);

5.1. Для Windows — IDE Visual Studio або Visual Code [11];

6. У випадку встановлення Visual Code, необхідно встановити розширення - C від Microsoft;

7. Встановити платформу .NET (в залежності від використовуваної ОС);

8. Підготувати звіт до лабораторної роботи.

Хід роботи:

1. Створити обліковий запис в GitHub. Оскільки в мене вже був створений обліковий запис за довго до цієї лабораторної роботи, це продемонстровано на рис.1.

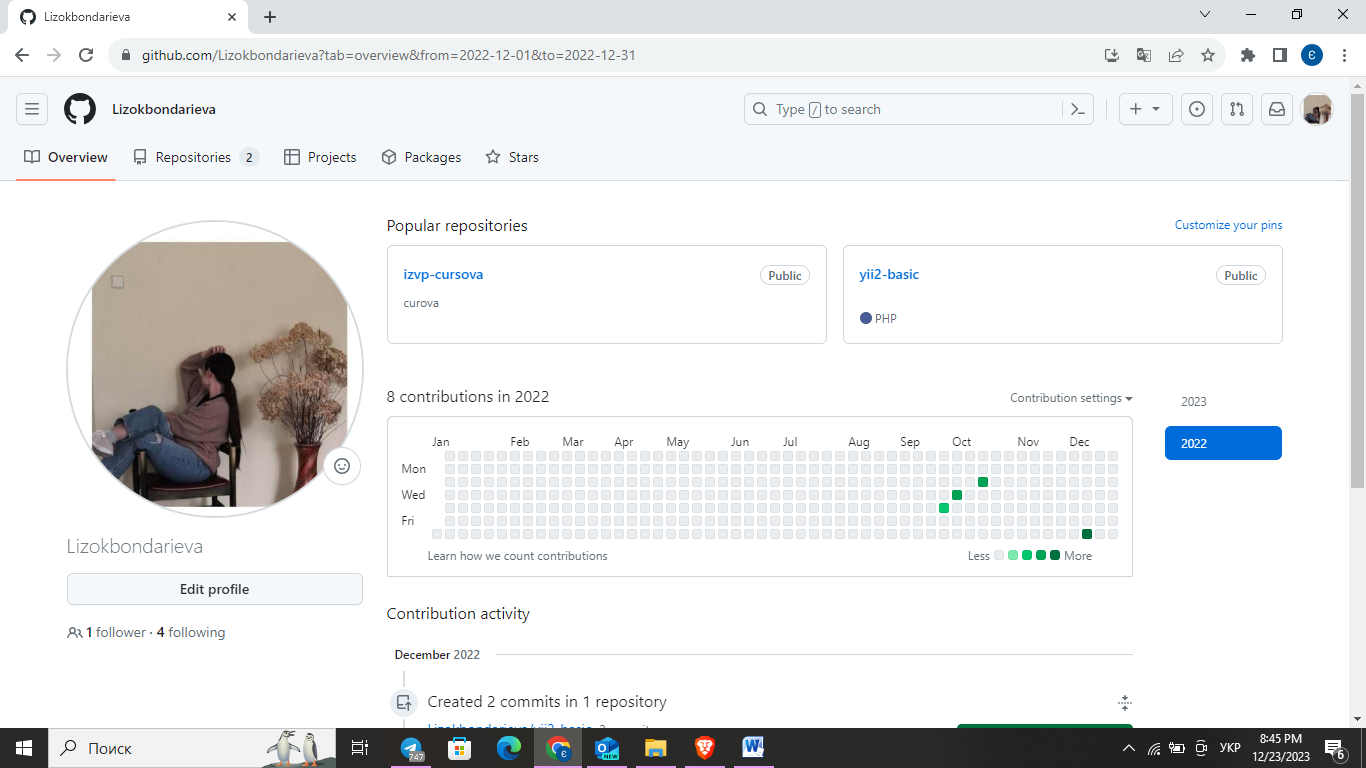


Рисунок 1 - Сторінка облікового запису в GitHub

2. Встановити систему контролю версій Git. Система контролю версій Git була встановлена разом з створенням облікового запису(див.рис.2-3).

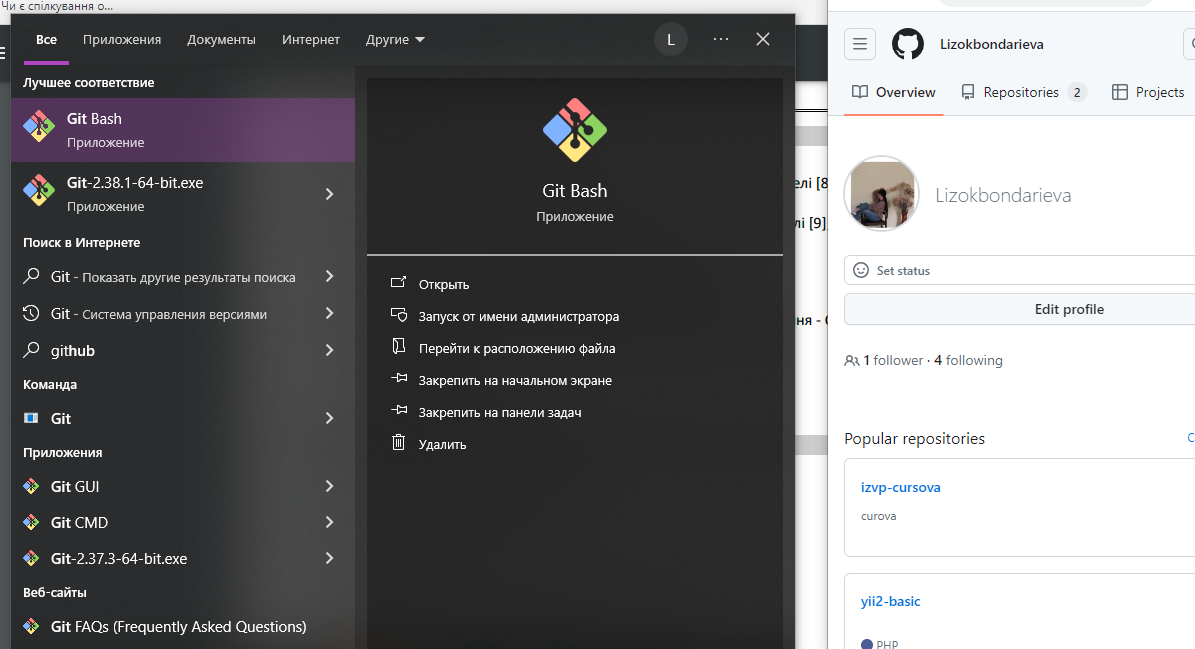


Рисунок 2 - Встановлена система контролю версій Git

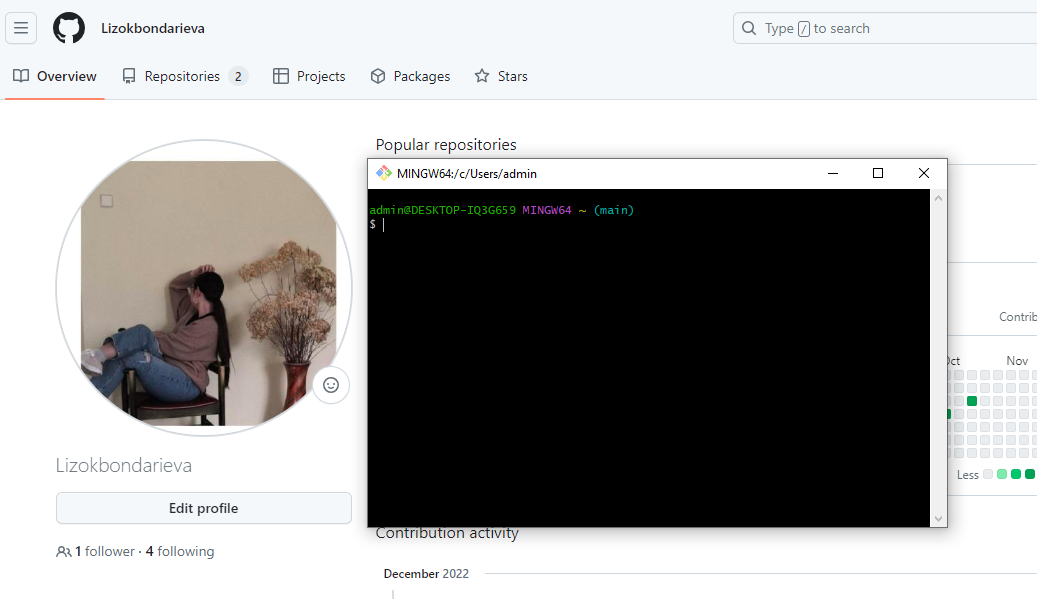


Рисунок 3 – Вікно встановленої системи контролю версій Git

3. Створити обліковий запис в Figma. Оскільки обліковий запис в Figma був створений за довго до виконання даної лабораторної роботи, його продемонстровано на рисунку 4.

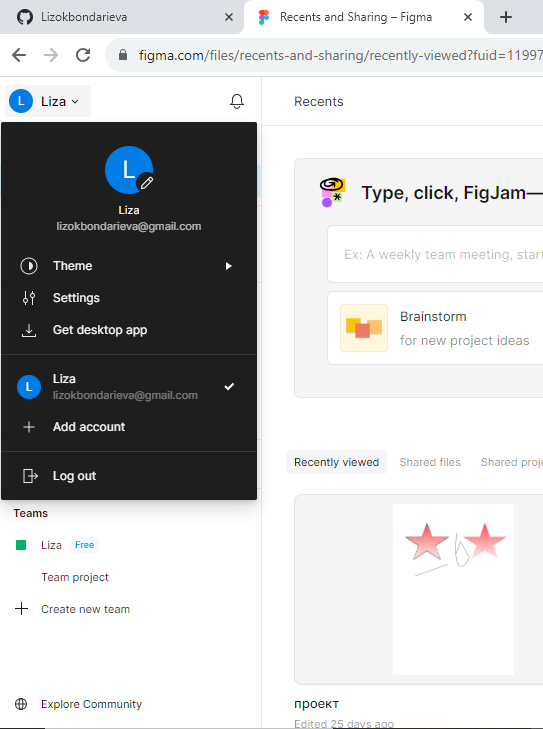


Рисунок 4 - Створений обліковий запис в Figma

4. Встановити IDE Visual Studio або Visual Code. Оскільки обидві IDE Visual Studio або Visual Code були встановлені на моєму девайсі вже давно це проілюстровано на рисунках 5-6.

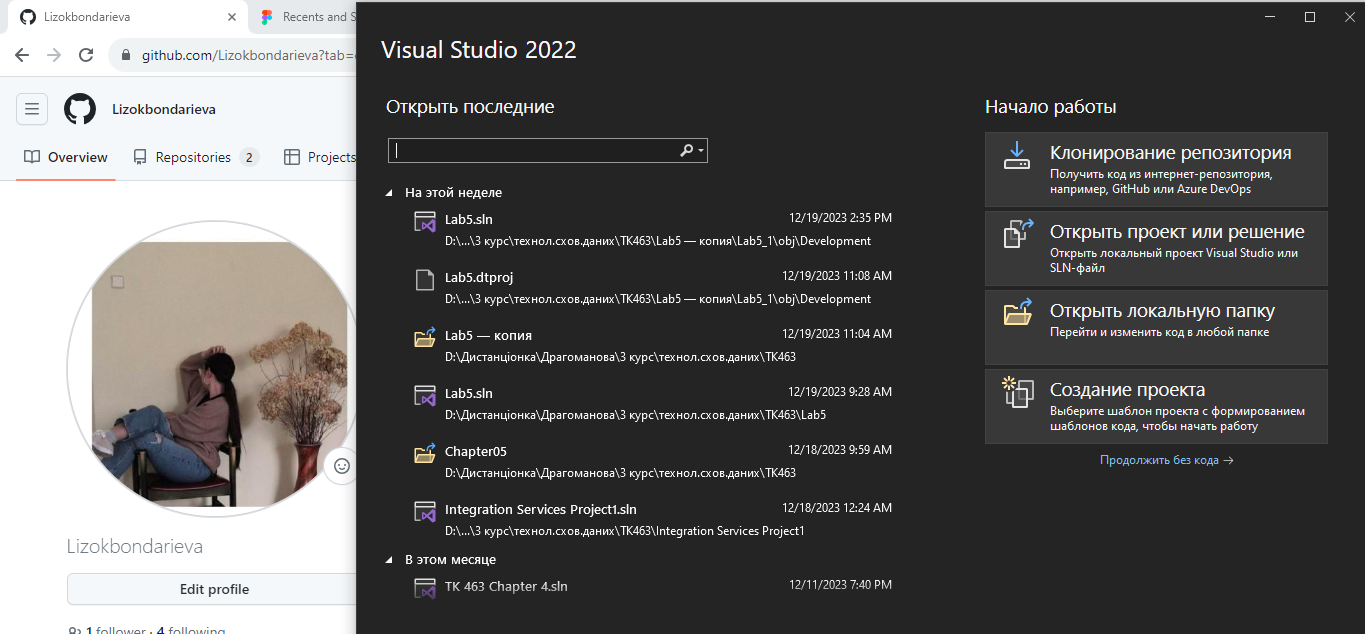


Рисунок 5 – Вікно встановленого Visual Studio

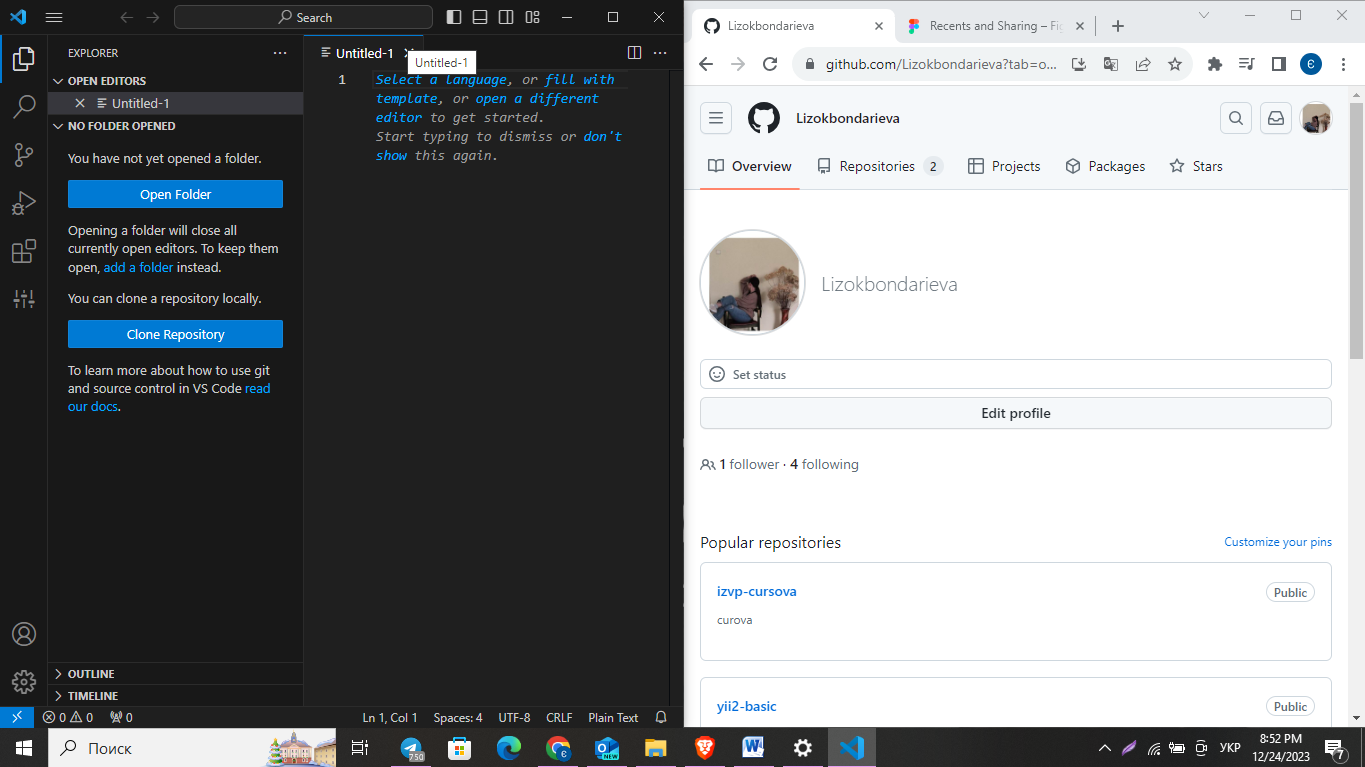


Рисунок 6 – Вікно встановленого Visual Studio Code

5. Встановити платформу .NET. Дана платформа була встановлена всередині Visual Studio (див.рис.7).

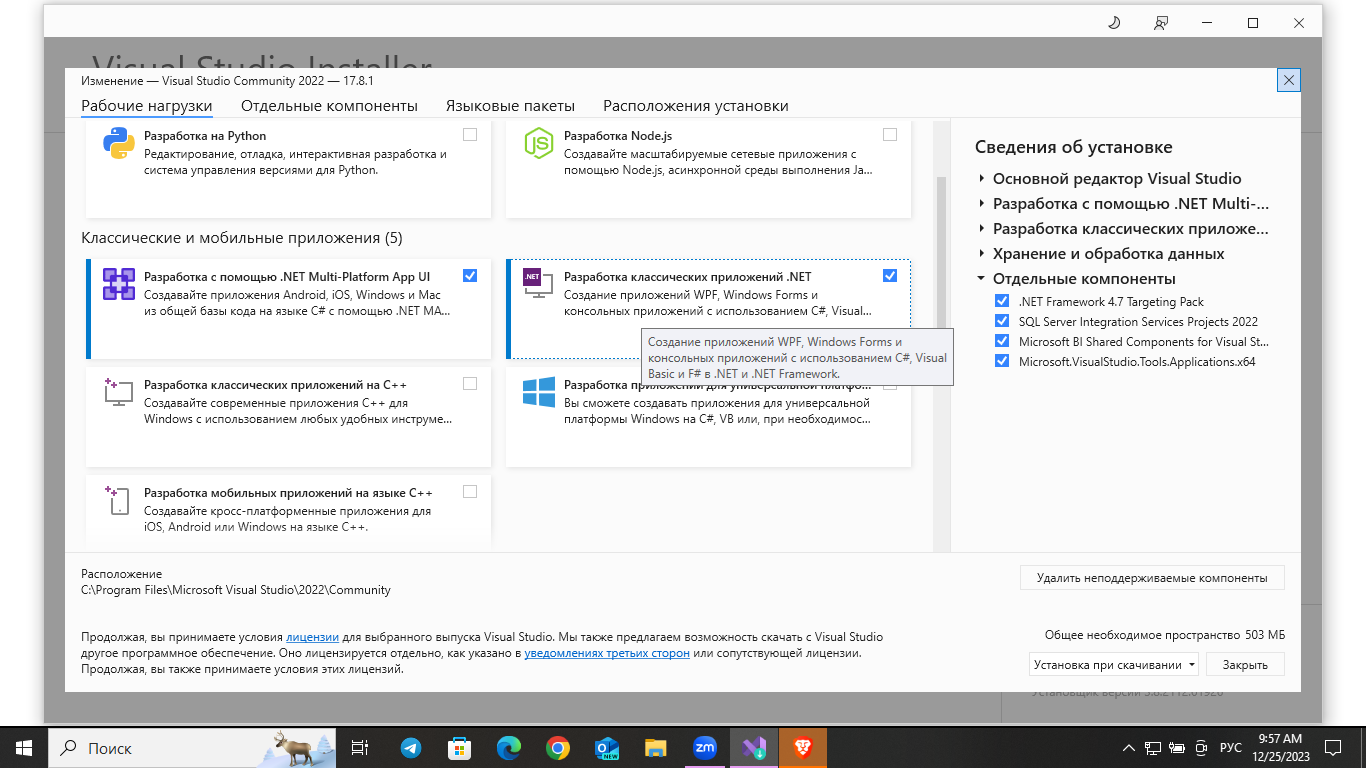


Рисунок 7 – Вікно встановленого.NET в середовищі Visual Studio

**Відповіді на контрольні запитання**

1. Що таке CASE-системи та їх призначення?

CASE (Computer-Aided Software Engineering) системи — це набір програмних інструментів та методів, які використовуються для підтримки розробки програмного забезпечення. Їхнє призначення полягає в автоматизації різних етапів життєвого циклу розробки програмного забезпечення, включаючи аналіз, проектування, реалізацію, тестування та управління проектом.

Основні функції CASE-систем включають:

1. Моделювання: CASE-інструменти дозволяють розробникам створювати моделі програми або системи перед тим, як розпочати активну розробку. Це може включати в себе діаграми потоку даних, діаграми класів, діаграми взаємодії та інші.

2. Генерація коду: Деякі CASE-системи можуть автоматично генерувати вихідний код або певні його частини на основі розроблених моделей. Це може значно спростити та прискорити процес розробки.

3. Управління версіями: CASE-інструменти часто надають можливості зберігання та відстеження змін в програмному коді, щоб полегшити спільну роботу команди розробників.

4. Аналіз та тестування: CASE-системи можуть підтримувати процеси аналізу вимог, тестування та валідації програмного забезпечення.

5. Управління проектом: Деякі CASE-інструменти дозволяють вести облік завдань, ресурсів та графіків в рамках проекту, що полегшує управління та контроль над розробкою.

CASE-системи є корисними для розробників, оскільки вони сприяють підвищенню продуктивності, покращенню якості програмного забезпечення та зменшенню ризиків помилок під час розробки.

1. Призначення систем керування проектами

Системи керування проектами (Project Management Systems) призначені для ефективного планування, виконання та контролю за проектами. Їх основні функції і призначення включають:

1. Планування проекту: Дозволяють визначити обсяг робіт, ресурси, терміни та інші аспекти проекту. Це може включати створення графіків, розподіл завдань та визначення залежностей між ними.

2. Управління завданнями: Дозволяють розподіляти завдання між командою, встановлювати пріоритети та відстежувати стан виконання кожного завдання.

3. Відстеження ресурсів: Допомагають управляти ресурсами, включаючи людські, фінансові та матеріальні ресурси. Забезпечують можливість відстежувати витрати та забезпечувати ефективне використання ресурсів.

4. Контроль стану проекту: Надають інструменти для відстеження прогресу, ризиків та виконання графіку. Забезпечують можливість оперативно реагувати на зміни та проблеми в процесі виконання проекту.

5. Комунікація та співпраця: Забезпечують засоби для комунікації та співпраці між членами команди, клієнтами та іншими зацікавленими сторонами.

6. Документування: Надають можливість зберігати та обмінюватися документацією проекту, такою як плани, звіти та інші документи.

7. Оцінка ризиків: Дозволяють визначити та відстежувати можливі ризики проекту, а також розробляти стратегії їх управління.

8. Звітність: Надають можливість створювати різноманітні звіти та аналізи щодо прогресу та результатів проекту.

Застосування систем керування проектами сприяє підвищенню ефективності роботи, зменшенню ризиків затримок та перевищення бюджету, а також полегшує співпрацю всіх учасників проекту.

1. Призначення хостингів проектів

Хостинг проектів відноситься до надання інфраструктурних ресурсів та сервісів для зберігання, керування та розгортання програмного коду, файлів та інших елементів проекту в Інтернеті. Основне призначення хостингу проектів включає в себе:

1. Зберігання файлів та даних: Хостинг проектів надає місце для зберігання різноманітних файлів, таких як вихідний код програм, документація, графіка, бази даних та інші компоненти проекту.

2. Керування версіями: Багато хостинг-платформ надають системи керування версіями, які дозволяють розробникам відстежувати та керувати змінами у вихідному коді. Це полегшує співпрацю між розробниками та ведення історії змін.

3. Розгортання (deployment) проекту: Хостинг дозволяє розміщувати ваш проект у відкритому доступі, щоб інші користувачі або члени команди могли перевіряти його роботу, а також використовувати вирішення в реальному часі.

4. Керування завданнями та проектами: Деякі хостинг-платформи також надають інструменти для керування завданнями, призначенням ролей та іншими аспектами управління проектом.

5. Інтеграція з іншими сервісами: Хостинг проектів може бути інтегрований з іншими сервісами, такими як сервіси автоматичної збірки, тестування, аналізу коду, спільної роботи та інші.

6. Забезпечення доступу: Забезпечення доступу до проекту для всіх членів команди, клієнтів або співпрацівників, навіть якщо вони знаходяться в різних місцях.

7. Забезпечення безпеки: Захист від несанкціонованого доступу, резервне копіювання та інші заходи безпеки для зберігання та збереження даних проекту.

Хостинг проектів дозволяє розробникам та командам зосередитися на роботі над своїми проектами, не турбуючись про інфраструктурні аспекти зберігання та розгортання.

1. Призначення та використання IDE

IDE (Integrated Development Environment) — це інтегроване середовище розробки, яке надає розробникам зручний інтерфейс та набір інструментів для створення, редагування, компіляції, налагодження та керування програмами. Основне призначення та використання IDE включають:

1. Кодування та Редагування Коду:

- Підсвічування синтаксису: IDE виділяє ключові елементи коду різними кольорами, полегшуючи сприйняття.

- Автодоповнення: Пропонує варіанти завершення коду та автоматично вставляє шаблони коду.

- Перевірка помилок: Сповіщення про синтаксичні та логічні помилки надають можливість виправити їх на ранніх етапах розробки.

2. Керування Проектами:

- Створення та організація проектів: Інструменти для створення, відкриття та управління проектами забезпечують легку навігацію у великих кодових базах.

- Версійний контроль: Інтеграція з системами контролю версій для збереження та відстеження змін в коді.

3. Компіляція та Виконання Коду:

- Автоматична компіляція: IDE може автоматично компілювати код під час редагування та підкреслювати можливі проблеми.

- Запуск та налагодження програм: Можливість виконувати та налагоджувати програми безпосередньо з інтерфейсу IDE.

4. Відстеження та Аналіз Коду:

- Аналіз коду: Інструменти для виявлення потенційних проблем, оптимізації та стильового аналізу коду.

- Вбудовані тестувальні інструменти: Підтримка тестування та автоматичної перевірки коду.

5. Налагодження та Профілювання:

- Відлагодження коду: Можливість встановлення точок зупинки, відстеження значень змінних та аналіз стеку викликів.

- Профілювання коду: Інструменти для виявлення та усунення проблем з продуктивністю програм.

6. Документація та Співпраця:

- Підтримка документації: Можливість додавати коментарі, генерувати документацію та використовувати підказки для розробників.

- Колективна робота: Інтеграція з системами спільної роботи та комунікації.

Ідеально підібрана IDE значно полегшує роботу розробників та забезпечує ефективність у розробці програмного забезпечення.

1. Приклади IDE

Існує багато IDE для різних мов програмування та технологій. Нижче подано кілька прикладів популярних IDE:

1. Eclipse:

- Мови: Java, C/C++, PHP, Python, та інші.

- Використання: Веб-розробка, мобільна розробка, корпоративні застосунки.

2. IntelliJ IDEA:

- Мови: Java, Kotlin.

- Використання: Розробка Java-застосунків, Android-розробка.

3. Visual Studio:

- Мови: C, C++, Visual Basic, F, та інші.

- Використання: Розробка Windows-застосунків, веб-розробка, мобільна розробка, хмарові послуги.

4. PyCharm:

- Мови: Python.

- Використання: Розробка веб-застосунків, аналіз даних, наукові дослідження.

5. Android Studio:

- Мови: Java, Kotlin.

- Використання: Розробка мобільних додатків для Android.

6. Xcode:

- Мови: Swift, Objective-C.

- Використання: Розробка мобільних додатків для iOS та macOS.

7. NetBeans:

- Мови: Java, PHP, HTML5, JavaScript, та інші.

- Використання: Різноманітні типи розробок, включаючи веб-розробку та Java-застосунки.

8. Sublime Text:

- Мови: Підтримує багато мов програмування.

- Використання: Загальне використання, редагування текстових файлів, програмування.

9. Atom:

- Мови: Підтримує багато мов програмування.

- Використання: Редагування коду, веб-розробка.

10. Visual Studio Code:

- Мови: Підтримує багато мов програмування.

- Використання: Загальне використання, редагування коду, веб-розробка.

Ці IDE відомі своєю функціональністю, широкою підтримкою мов та різноманіттю інструментів для зручної розробки програмного забезпечення. Розробники вибирають IDE відповідно до своїх потреб та вподобань.

1. Як додати в поточний проект сторонню бібліотеку коду?

Додавання сторонньої бібліотеки коду до поточного проекту зазвичай включає кілька кроків. Нижче подано загальний опис цього процесу:

1. Вибір бібліотеки:

- Визначте, яку бібліотеку ви хочете додати до свого проекту. Зазвичай це буде бібліотека, яка вирішує конкретну задачу або надає певний функціонал.

2. Завантаження бібліотеки:

- Завантажте бібліотеку або використовуйте інструменти управління пакетами (наприклад, npm для JavaScript, pip для Python, Maven або Gradle для Java), якщо такі є, для автоматизації завантаження.

3. Інтеграція з проектом:

- Додаємо бібліотеку до проекту, використовуючи один із наступних методів:

- Ручне додавання файлів:

- Скопіюйте файли бібліотеки в папку вашого проекту та додайте їх до системи контролю версій (якщо ви його використовуєте).

- Використання інструментів збірки (Build Tools):

- Використовуйте систему управління пакетами або залежностями вашої мови програмування. Наприклад, для Java це може бути Maven або Gradle, для JavaScript - npm або Yarn.

4. Конфігурація проекту:

- Додайте посилання на бібліотеку у конфігураційних файлах вашого проекту. Це може включати в себе налаштування шляху до бібліотеки, версії, ініціалізації та інші параметри, які можуть бути потрібні.

5. Перевірка та Тестування:

- Переконайтеся, що ваш проект вдало інтегрується з новою бібліотекою. Виконайте тести, переконайтеся, що немає конфліктів і що бібліотека працює, як очікується.

6. Оновлення та Підтримка:

- Відстежуйте нові версії бібліотеки і вчасно оновлюйте її в вашому проекті, щоб користуватися всіма нововведеннями та виправленнями помилок.

Зазначте, що процедура може трошки відрізнятися залежно від використовуваної мови програмування та інструментів збірки. Важливо дотримуватися рекомендацій та документації, які надає розробник бібліотеки, щоб уникнути можливих проблем.

1. Призначення систем контролю версій

Системи контролю версій (Version Control Systems, VCS) призначені для відстеження змін у файлах та деревах каталогів в процесі розробки програмного забезпечення. Основні цілі та призначення систем контролю версій включають:

1. Відстеження Змін:

- Спостереження за змінами: Системи контролю версій відслідковують, які файли змінено, коли і ким.

2. Версіонування:

- Створення версій: Зберігання копій коду та ресурсів на різних етапах розвитку проекту, щоб забезпечити можливість відновлення попередніх станів.

3. Співпраця та Робота В Команді:

- Паралельна робота: Дозвіл одночасно працювати над проектом багатьом розробникам, при цьому автоматично об'єднується їхні зміни.

4. Виявлення та Виправлення Помилок:

- Виявлення помилок: Здатність визначати, коли та де виникають помилки, і легко відновлювати код до попередніх станів.

5. Гілкування та Злиття (Branching and Merging):

- Гілкування: Створення окремих ліній розвитку (гілок) для вирішення певних завдань чи функцій.

- Злиття: Об'єднання різних гілок для об'єднання роботи розробників.

6. Історія та Авторство:

- Історія змін: Зберігання повної історії всіх змін, включаючи коментарі та відомості про автора.

- Визначення авторства: Вказання того, хто та коли вніс певні зміни.

7. Робота Офлайн:

- Локальне збереження: Зберігання повної копії проекту на робочій машині розробника, що дозволяє працювати офлайн.

8. Відгалуження для Випуску (Release Branching):

- Створення гілки для випуску: Розділення роботи для підготовки випуску програмного продукту.

9. Безпека та Відновлення:

- Забезпечення безпеки: Захист від втрати даних або випадкового видалення файлів.

- Відновлення до попередніх станів: Можливість повернутися до певної версії коду у випадку серйозних проблем.

Системи контролю версій є необхідним інструментом у сфері розробки програмного забезпечення, забезпечуючи розробникам управління та відстеження змін для ефективної та спільної роботи.

1. Типи тестування

Тестування програмного забезпечення — це важлива частина процесу розробки, спрямована на визначення помилок, перевірку правильності та забезпечення високої якості продукту. Існує багато різних видів тестування, які можна класифікувати за різними критеріями. Основні типи тестування включають:

1. Функціональне тестування (Functional Testing):

- Unit Testing (Тестування одиниць): Перевірка окремих функцій або модулів програми.

- Integration Testing (Інтеграційне тестування): Тестування взаємодії між різними компонентами або модулями.

- System Testing (Системне тестування): Перевірка всієї системи як єдиної юніті.

2. Невідновлювальне тестування (Non-functional Testing):

- Performance Testing (Тестування продуктивності): Визначення швидкодії, завантаження та ефективності системи.

- Security Testing (Тестування безпеки): Виявлення потенційних загроз та заходів забезпечення безпеки.

- Usability Testing (Тестування зручності використання): Оцінка зручності використання та інтерфейсу користувача.

3. Ручне та Автоматизоване Тестування:

- Manual Testing (Ручне тестування): Тестування, яке виконується без використання автоматизованих інструментів.

- Automated Testing (Автоматизоване тестування): Використання скриптів та інструментів для автоматизації процесу тестування.

4. Регресійне Тестування (Regression Testing):

- Перевірка того, чи впливають нові зміни на вже існуючий функціонал.

5. Тестування Прийняття (Acceptance Testing):

- User Acceptance Testing (UAT) (Тестування користувацької прийняття): Тестування, що виконується користувачами для підтвердження, що система відповідає їхнім вимогам.

6. Тестування Сумісності (Compatibility Testing):

- Визначення того, як програма або система працює на різних платформах, браузерах, операційних системах тощо.

7. Тестування Білих та Чорних Скринь (Black Box Testing & White Box Testing):

- Black Box Testing (Тестування чорної скриньки): Тестування, при якому тестувальник не має інформації про внутрішню реалізацію системи.

- White Box Testing (Тестування білої скриньки): Тестування, при якому тестувальник має знання про внутрішню реалізацію системи.

Ці типи тестування є лише деякими з можливих. Вибір конкретного типу залежить від вимог та характеристик конкретного проекту.

1. Призначення емуляторів та віртуальних машин

Емулятори та віртуальні машини — це інструменти, які використовуються в області інформаційних технологій для різних цілей. Їх призначення може змінюватися в залежності від конкретного контексту, але основні функції такі:

Емулятори:

1. Моделювання апаратного забезпечення:

- Приладні емулятори: Моделювання різних пристроїв (наприклад, смартфонів, планшетів) для розробки та тестування програмного забезпечення.

2. Тестування та Навчання:

- Емуляція середовища: Використання емуляторів для створення тестових середовищ або для тренування без реальних витрат на обладнання.

3. Відтворення Умов:

- Емуляція мережі: Створення умов мережі для тестування різних сценаріїв та умов для програм та систем.

4. Забезпечення Сумісності:

- Емуляція операційних систем: Тестування програмного забезпечення на різних операційних системах, не використовуючи фізичні пристрої.

Віртуальні машини (VM):

1. Ізоляція та Сегментація:

- Ізоляція ресурсів: Забезпечення ізоляції від інших віртуальних машин, що дозволяє запускати різні операційні системи на одному фізичному сервері.

2. Тестування та Відновлення:

- Тестування на різних конфігураціях: Віртуальні машини дозволяють легко тестувати програмне забезпечення на різних конфігураціях обладнання.

3. Відновлення Системи:

- Створення знімків: Віртуальні машини можуть створювати знімки стану, що дозволяє відновлювати систему до попередніх станів.

4. Розгортання Систем:

- Легке розгортання: Швидке розгортання стандартних віртуальних машин для швидкого старту нових систем.

5. Ізоляція Середовища Розробки:

- Ізоляція розробки: Розділення середовищ розробки від основної системи, щоб уникнути конфліктів.

6. Обчислення у Хмарі:

- Хмарні віртуальні машини: Використання віртуальних машин у хмарних обчисленнях для забезпечення масштабованості та ефективного використання ресурсів.

Якщо коротко, емулятори частіше використовуються для моделювання реальних пристроїв чи середовищ, тоді як віртуальні машини забезпечують ізольоване виконання операційних систем та додатків на фізичному обладнанні. Обидва підходи дозволяють розробникам та адміністраторам масштабувати та управляти ресурсами більш ефективно.

1. Відмінність між контейнером та віртуальною машиною

Контейнери та віртуальні машини (VMs) - це два різні підходи до віртуалізації та ізоляції середовищ в обчислювальних системах. Основні відмінності між ними включають:

Віртуальні машини (VMs):

1. Операційна Система:

-VM: Кожна віртуальна машина включає в себе повністю функціональну операційну систему.

- Контейнер: Контейнери використовують операційну систему господаря та ділять ядро операційної системи з господарям.

2. Ресурси:

- VM: Віртуальні машини використовують більше ресурсів, оскільки кожна з них має свою власну операційну систему та віртуальний стек.

- Контейнер: Контейнери використовують менше ресурсів, оскільки вони спільно використовують ядро операційної системи та ресурси господаря.

3. Запуск:

- VM: Запуск віртуальних машин може займати більше часу, оскільки потрібно завантажити та запустити повну операційну систему.

- Контейнер: Контейнери запускаються швидше, оскільки вони використовують операційну систему господаря та мають менше об'єму системних ресурсів.

4. Ізоляція:

- VM: Віртуальні машини ізольовані одна від одної, оскільки кожна має власну операційну систему.

- Контейнер: Хоча контейнери також ізольовані, вони використовують механізми просторів імен та контролю груп для ізоляції процесів.

Контейнери:

1. Розмір:

- VM:Зазвичай більші в розмірі, оскільки включають в себе операційну систему та додаткові компоненти.

- Контейнер: Зазвичай менші в розмірі, оскільки вони використовують операційну систему господаря та спільні бібліотеки.

2. Масштабованість:

- VM: Важко швидко масштабувати, оскільки створення та запуск нових віртуальних машин вимагає багато ресурсів.

- Контейнер: Легко масштабується, оскільки можна швидко створювати та запускати нові контейнери.

3. Мобільність:

- VM: Важко переносити між різними гіпервізорами через різні формати віртуальних машин.

- Контейнер: Легко переносити, оскільки вони мають стандартизований формат та використовують загальні компоненти.

4. Інструменти:

-VM: Вимагає гіпервізора для запуску, такого як VMware, VirtualBox, або Hyper-V.

- Контейнер:Вимагає контейнерного двигуна, такого як Docker.

Обидва підходи мають свої переваги та обмеження, і вибір між ними залежить від конкретних вимог та умов конкретного проекту. Контейнери широко використовуються в сучасних додатках через свою легкість та швидкодію, а віртуальні машини можуть бути використані там, де важлива повна ізоляція та незалежність операційних систем.

Висновки

В ході виконання лабораторної роботи 1 з дисципліни патерни програмування та проектування мною була виконана підготовка необхідних робочих інструментів.

Досягнуто мети, а саме я: ознайомилась та підготувала інструменти під власні потреби, які використовуються для розробки програмного забезпечення (ПЗ). Ознайомилася з теоретичною частиною.

Створила (поновила) обліковий запис в GitHub;знайшла та продемонструвала встановлену систему контролю версій Git;створила (поновила) обліковий запис в Figma; продемонструвала встановлені IDE Visual Studio або Visual Code ;встановити платформу .NET.

Під час виконання даної лабораторної роботи помилок та труднощів не виникло, поставлена мета була досягнута.

Список використаних джерел:

1. <https://moodle.fmif.udu.edu.ua/mod/assign/view.php?id=33035>
2. https://moodle.fmif.udu.edu.ua/pluginfile.php?file=%2F76464%2Fmod\_assign%2Fintroattachment%2F0%2F%D0%9B.%20%D1%80.%20%E2%84%961%20-%20%D0%86%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B8%20%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B8.pdf&amp;forcedownload=