М. О. Данова, Ю. А. Кузнецова, М. О. Сьомочкін

ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»

М. О. Данова, Ю. А. Кузнецова, М. О. Сьомочкін

ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Навчальний посібник
до виконання лабораторних робіт

Харків «ХАІ» 2021

УДК 004.4
Д17

Рецензенти: д-р техн. наук, проф. М. О. Волк;
д-р техн. наук, проф. А. А. Коваленко

Данова, М. О.
Д75 Інформаційні технології розроблення програмного забезпечення. М. О. Данова, Ю. А. Кузнецова, М. О. Сьомочкін – Навч. посібник до виконання лабораторних робіт. – Харків: Нац. аерокосм. ун-т «Харк. авіац. ін-т », 2021. – 90 с.

Наведено основні відомості про системи контролю версій, подано необхідний словник термінів для його застосування під час командного розроблення програмного забезпечення й використання веб-сервісів для хостингу проектів і їхнього спільного розроблення, що основані на системі контролю версій. Показана підготовка коду для тестування та впровадження залежностей, а також необхідність роботи учасників проекту в окремих гілках. Наведено основні відомості про використання засобів протипування та Mock-фреймворки. Висвітлено основні відомості про делегати, анонімні методи, лямбда-вирази, а також переваги використання Moq-об'єктів. Описано перелік переваг безперервної інтеграції проекту і застосування серверів збирання, а також аналіз актуальності використання новітніх інформаційних технологій розроблення програмного забезпечення.

Посібник буде корисним для студентів спеціальності 121 «Інженерія програмного забезпечення». Також може бути цікавим для студентів, аспірантам і технічним фахівцям, які прагнуть одержати базові знання про командне розроблення промислового програмного забезпечення з використанням систем контролю версій.

Іл. 69. Табл. 2. Бібліогр.: 45 назв.

УДК 004.4

ЗМІСТ

ВСТУП

Ситуація, в якій електронний документ за час свого існування зазнає ряд змін, досить типова. При цьому часто буває важливо мати не тільки останню версію, але і кілька попередніх.

Іноді нова версія створюється непомітно для користувача (прозоро) або прикладною програмою, що має вбудовану підтримку такої функції, або за рахунок використання спеціальної файлової системи. У цьому випадку користувач просто працює з файлом, як завжди, і при збереженні файла автоматично створюється нова версія.

Кожна система управління версіями (VCS – Version Control System) має свої специфічні особливості в наборі команд, порядку роботи користувачів і адмініструванні. Проте, загальний порядок роботи для більшості VCS є абсолютно стереотипним. Тут передбачається, що проект, яким би він не був, вже існує, і на сервері розміщений його репозитарій, до якого розробник отримує доступ.

В результаті виконання лабораторної роботи №1 студенти:

- сформують проектну команду;
- навчаться адмініструвати систему контролю версій, розподіляти і відстежувати завдання за проектом;
- навчаться інтегрувати проект до системи контролю версій;
- отримають навички практичного використання клієнта TortoiseHG.

В результаті виконання лабораторної роботи №2 студенти:

- отримають практичні навички у створенні віх проекту;
- сформують технічне завдання на розробку системи контролю версій;
- зможуть ознайомитися з роллю архітектора проекту (тімліда або бізнес-аналітика).

В результаті виконання лабораторної роботи №3 студенти:

- отримають практичні навички щодо додавання завдань до віх проекту;
- навчаться призначати завдання членам команди;
- ознайомляться з життєвим циклом завдання з прив'язкою до кожної ролі;
- навчаться змінювати статуси завдань;
- продовжать розробку проекту в ролі програміста.

В результаті виконання лабораторної роботи №4 студенти:

– ознайомляться з шаблонами впровадження залежностей і реализують один з них (Dependency injection);

– навчаться писати код, зручний для тестування;

– отримають змогу створювати каркас модульних тестів;

– навчаться працювати в різних гілках проекту і одночасно підтримувати кілька версій програмного забезпечення.

В результаті виконання лабораторної роботи №5 студенти:

– навчаться розробляти макети класів, реалізуючи інтерфейси класів мок-об'єктами;

– навчаться використовувати прототипи класів у модульному тестуванні;

– зрозуміють роль модульного тестування в процесі безперервної інтеграції проекту.

В результаті виконання лабораторної роботи №6 студенти:

– навчаться створювати інсталятор для розробленого програмного продукту;

– зможуть розубратися в правилах версіонування й унікальних кодах продукту;

– отримають досвід у підписанні інсталятора самопідписним сертифікатом.

В результаті виконання розрахункової роботи студенти зможуть налаштувати сервер безперервної інтеграції TeamCity (або Jenkins).

Для цього необхідно виконати наступні кроки:

– установка TeamCity на локальну машину;

– установка BuildAgent;

– базова настройка TeamCity;

– підключення сховища Git;

– створення кроків збірки;

– створення тригера запуску збірки.

В результаті буде протестованороботу сервера безперервної інтеграції шляхом запуску кількох збірок і доповнень змін в репозиторій Git (або Mercurial).

МЕТОДИЧНІ ВКАЗІВКИ ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ

Лабораторна робота № 1.
Командна робота. Інтеграція проекту до системи контролю версій

Мета лабораторної роботи. Сформувати проектну команду. Навчитися адмініструвати систему контролю версій, розподіляти і відстежувати завдання за проектом. Навчитися інтегрувати проект до системи контролю версій. Отримати навички практичного використання клієнта TortoiseHG.

Порядок виконання роботи:

1. Прочитати теоретичний матеріал до лабораторної роботи.
2. Розбитися на команди по 3-5 чоловік.
3. Виділити в команді ролі (тімліда, програміста, тестувальника).
4. Розробити простий додаток, який містить в собі кілька класів, при цьому кожен клас повинен бути реалізований окремим програмістом, а інтерфейси для кожного класу описуються тімлідом.

Проект. Немає обмежень на мову програмування. Можна використовувати (доопрацювати) opensource-проект або вибрати будь-який інший проект, доступний на використовуваному Вами веб-хостингу. Існує можливість вибору проекту за бажанням за умови узгодження з викладачем.

5. *Виконати тестове завдання (кожному учаснику команди):* розробити звичайний / інженерний калькулятор, при цьому для базових операцій («складання», «віднімання», «множення», «поділ») використовувати інтерфейси (проектуються тімліда, реалізуються розробником, тестування може проводити будь-який учасник команди).
6. *Розробку необхідно вести з використанням різних гілок:* відгалуження додавати або для кожної ролі учасника проекту, або для розробки / доопрацювання робочого / неробочого функціоналу).
7. У процесі виконання лабораторної роботи необхідно зафіксувати («заскрінити») всі помилки і проблеми, що виникли при використанні

обраної СКВ. Якщо подібних ситуацій не виникло, то зімітувати помилкове виконань операцій (хоча б одне). Детальний опис додати в звіт.

5. Тімліду:

a. Зареєструватися на ресурсі Bitbucket (bitbucket.org).

b. Створити Репозиторій (який буде використаний для виконання першої лабораторної роботи).

c. Створити Групу користувачів.

d. Додати до групи користувачів членів команди і призначити їм повні права доступу.

e. Створити Солюшин з проектом, який містить опис інтерфейсів класів.

6. Програмістам:

a. Встановити TortoiseHG (http://tortoisesvn.net/)

b. Отримати копію сховища на локальний комп'ютер.

c. Додати в Солюшин проекти, які реалізують інтерфейс і вкомітіть його в репозиторій.

d. Описати класи, що реалізують описані інтерфейси.

e. Закоммітіть і зафіксувати всі зміни на сервері.

6. Тестувальникам:

a. Підключитися до сховища проекту.

b. Витягти з репозиторію на комп'ютер робочу копію проекту.

c. Перевірити функціональність і взаємодію інтерфейсів. Усунути баги.

d. Залити назад проект в SVN.

7. Скласти матрицю відповідальних за виконання проекту.

8. Оформити звіт, перевірити його у викладача, захистити лабораторну роботу.

9. Зберегти файл, який містить звіт з лабораторної роботи, та файл-архів проекту в системі Mentor (mentor.khai.edu).

Презентація.

Презентація проекту учасниками команди. Кожен з учасників команди готує доповідь про виконану роботу, а також про подальше планування своїх дій.

Зміст звіту:

1. Постановка завдання.
2. Опис проекту.
3. Обгрунтування вибору проекту: в чому його актуальність, на яку аудиторію користувачів розрахований (орієнтований).
4. Обгрунтування вибору системи контролю версій. Короткі теоретичні відомості про те, яким проектом більше підходить та чи інша СКВ. Чому? (Дати коротке пояснення). Критерії вибору (перерахувати).
5. Аналіз коментарів експертів щодо використовуваних систем контролю версій. У ролі експертів, як правило, виступають розробники програмного забезпечення великих IT-підприємств, які мають великий досвід у сфері створення програмного продукту.
6. Обгрунтування вибору типу сховища (public, private).
7. Скріншоти створеного сховища та груп користувачів.
8. При необхідності – пояснення до реалізації.
9. Текст програми.
10. Результати тестів (скріншоти).
11. Матриця відповідальних.
12. Історія коммітов.
13. Висновки.

Лабораторна робота № 2.
Узгодження технічного завдання на розробку. Віхи проекту

Мета лабораторної роботи. Отримати практичні навички у створенні віх проекту. Сформувати технічне завдання на розробку системи контролю версій. Ознайомитися з роллю архітектора проекту (тімліда, бізнес аналітика).

Порядок виконання роботи:

1. Прочитати теоретичний матеріал до лабораторної роботи.
2. Продовжити роботу у команді з 3-5 чоловік.
3. Виділити в команді ролі (тімліда, програміста, тестувальника). Можна змінити ролі.
4. Виділити базові функціональності системи, що розробляється, і оформити у вигляді технічного завдання (ТЗ) (варіанти узгодити з викладачем).
5. Розбити проект на віхи (етапи, milestones) відповідно до ТЗ і узгодити терміни з семестровими планом (у викладача).
6. Розбити проект на групи завдань і автономні завдання.
7. Для кожної ролі члена команди додати проект в Solution. Кожен проект повинен відповідати завданням, виділеним з ТЗ для кожної ролі учасника команди.

8. Для ролі тімліда (основна роль на даному етапі):

a. Створити репозиторій (bitbucket), який буде використаний у 2 – 5 лабораторних роботах.

b. Запросити членів команди в проект, відправивши кожному invite на e-mail.

c. Створити Solution в VisualStudio.

d. Додати в Solution проект Core, який буде містити загальну функціональність і інтерфейси для всього Солюшин.

e. Додати загальні інтерфейси взаємодії компонент.

f. Зафіксувати зміни на сервері Assembla (або будь-якому іншому сервері, що підримує роботу з СКВ).

9. Для ролі програміста (допоміжна роль):

a. Отримати робочу копію сховища на локальний комп'ютер.

b. Створити проект і додати його в репозиторій.

c. Створити прототип програми відповідно до описаної архітектурою.

d. Реалізувати базові функції.

e. Зафіксувати зміни на сервері.

10. Для ролі тестувальника (допоміжна роль):

a. Створити проект (unit-test) і додати його в репозиторій.

b. Зафіксувати зміни на сервері.

c. На основі розробленої специфікації проекту скласти майбутній план тестування.

11. Оформити звіт, перевірити у викладача і захистити роботу.

12. Зберегти файл, який містить звіт з лабораторної роботи, в системі Mentor (mentor.khai.edu).

Презентація.

Презентація проекту учасниками команди. Кожен з учасників команди готує доповідь про виконану роботу, а також про подальше планування своїх дій.

Зміст звіту:

1. Постановка завдання.
2. ТЗ проекту.
3. Віхи проекту.
4. Структура Солюшин з описом кожного проекту.
5. При необхідності – пояснення до реалізації.
6. Текст програми.
7. Скріншоти створеного сховища та історії змін.
8. Базовий план тестування з тестовими випадками.
9. Висновки.

Лабораторна робота № 3.
Тікетування та управління завданнями

Мета лабораторної роботи. Отримати практичні навички щодо додавання завдань до віх проекту. Навчитися призначати завдання членам команди. Ознайомитися з життєвим циклом завдання з прив'язкою до кожної ролі. Навчитися змінювати статуси завдань. Продовжити розробку проекту в ролі програміста.

Порядок виконання роботи:

1. Прочитати теоретичний матеріал до лабораторної роботи.
2. Продовжити роботу у команді з 3-5 чоловік.
3. Виділити в команді ролі (тімліда, програміста, тестувальника). Можна змінити ролі.
4. Розбити проект на 3-4 ітерації (для 2, 3 і 4 лабораторних робіт).
5. Для кожної ролі члена команди виділити компонент системи, за який відповідає дана роль.
6. Обміркувати і спланувати, які завдання готові виконати члени команди в найближчу ітерацію й наступні ітерації.

7. Тімліду (допоміжна роль):

a. Додати кожної ролі учасника проекту завдання (мінімум по 3 завдання на кожну віху проекту – для програмістів розробка архітектури та кодування, для тестувальників – складання unit-тестів і написання документації);

b. Виставити для кожного завдання передбачувану оцінку трудомісткості.

c. Після завершення ітерації відстежити активність виконання завдань кожним членом команди (якщо на проекті більше одного учасника) скласти звіт активності зміни стану завдань.

8. Програмістам (основна роль):

a. Вивчити закріплені завдання і вибрати на тиждень.

b. Відкрити завдання першої ітерації на виконання.

c. Виконати завдання першої ітерації проекту.

d. Зафіксувати зміни в репозиторії з прив'язкою до завдань.

e. Проставити витрачений час роботи для кожного завдання, закрити виконані і відправити на тестування.

f. Повторювати ітераційно, поки не завершиться ітерація або завдання, виділені на ітерацію.

9. Тестувальникам (допоміжна роль):

a. Виконати завдання першої віхи проекту (написати хоча б 1 модульний тест на ті завдання, які реалізують програмісти).

b. Зафіксувати зміни в репозиторії з прив'язкою до завдань.

c. Проставити витрачений час роботи для кожного завдання і закрити виконані.

10. Оформити звіт, перевірити у викладача і захистити роботу.

11. Зберегти файл, який містить звіт з лабораторної роботи, в системі Mentor (mentor.khai.edu).

Презентація.

Презентація проекту учасниками команди. Кожен з учасників команди готує доповідь про виконану роботу, а також про подальше планування своїх дій.

Зміст звіту:

1. Постановка завдання.
2. Опис віх проекту.
3. Опис завдань, розбитих по віхам проекту.
4. Активність роботи над завданнями.
5. Лістинг коду, асоційованого з виконаної завданням.
6. При необхідності – пояснення до реалізації.
7. Лістинг модульних тестів.
8. При необхідності – пояснення до реалізації тестів.
9. Висновки.

Лабораторна робота № 4.
Підготовка коду для тестування. Впровадження залежностей. Робота в окремих гілках

Мета лабораторної роботи. Ознайомитися з шаблонами впровадження залежностей і реалізувати один з них (Dependency injection). Навчитися писати код, зручний для тестування. Створити каркас модульних тестів. Навчиться працювати в різних гілках проекту і одночасно підтримувати кілька версій програмного забезпечення.

Порядок виконання роботи:

1. Прочитати теоретичний матеріал до лабораторної роботи.
2. Продовжити роботу у команді з 3-5 чоловік.
3. Виділити в команді ролі (тімліда, програміста, тестувальника). Можна змінити ролі.
4. Вибрати один з патернів впровадження залежності (вибір обґрунтувати):

- через метод класу (англ. Setter injection);
- через конструктор (англ. Constructor injection);
- через інтерфейс впровадження (англ. Interface injection).

5. Тімліда (основна роль):

a. Вибрати один з існуючих для тестування (даний клас повинен залежати від ще одного класу, що розробляється на даному проекті).

b. Перевірити коректність реалізації паттерна впровадження залежностей.

c. Створити завдання рефакторінга коду для приведення коду відповідно паттерну впровадження залежностей.

d. Створити окрему гілку в репозиторії, в якій буде проводитися рефакторинг.

e. Видати тестувальникам ТЗ на тестування незалежних класів у вигляді тікетів.

f. Після закінчення виконання всіх завдань рефакторінга провести злиття версій в основний стовбур розробки.

7. Програмістам (основна роль):

a. Провести рефакторинг існуючих класів відповідно до уточненими вимогами проекту в окремій гілці проекту.

b. Виділити залежні і незалежні класи.

8. Тестувальникам (основна роль):

a. Виділити тестовані класи і методи.

b. Додати в тестовий проект модульні тести незалежних методів класів.

c. Провести unit-тестування (без використання підміни класів).

9. Оформити звіт, перевірити у викладача і захистити роботу.

10. Зберегти файл, який містить звіт з лабораторної роботи, в системі Mentor (mentor.khai.edu).

Презентація.

Презентація проекту учасниками команди. Кожен з учасників команди готує доповідь про виконану роботу, а також про подальше планування своїх дій.

Зміст звіту:

1. Постановка завдання.
2. Опис обраного паттерна впровадження залежності з обґрунтуванням вибору.
3. Опис завдань рефакторінга, поставлених тімлідом на поточну ітерацію. При необхідності – пояснення до реалізації.
4. Активність роботи над завданнями.
5. Лістинг коду, асоційованого з виконаним завданням. При необхідності додати пояснення до реалізації.
6. Тестовані аспекти, розроблені test-case, звіт про тестування. При необхідності – пояснення до реалізації тестів.
7. Висновки.

Лабораторна робота № 5.
Використання засобів протипування. Mock-фреймворки

Мета лабораторної роботи. Навчитися розробляти макети класів, реалізуючи інтерфейси класів мок-об'єктами. Навчитися використовувати прототипи класів у модульному тестуванні. Зрозуміти роль модульного тестування в процесі безперервної інтеграції проекту.

Порядок виконання роботи:

1. Прочитати теоретичний матеріал до лабораторної роботи.
2. Продовжити роботу у команді з 3-5 чоловік.
3. Виділити в команді ролі (тімліда, програміста, тестувальника). Можна змінити ролі.
4. Для модифікованих класів з попередньої лабораторної роботи написати класи-прототипи з застосуванням Moq бібліотеки, останню версію якої можна скачати з сайту: http://code.google.com/p/moq/downloads/list або використовувати будь-яку іншу звичну бібліотеку.

5. Тімліду (допоміжна роль):

a. Завантажити бібліотеку і додати в проект у вигляді рессурса.

b. Підключити Moq бібліотеку в проект, який відповідає за модульні тести і закоммітіть зміни на сервер.

c. Виділити пов'язані класи, в тестуванні яких необхідно використовувати макетування.

d. Створити завдання в багтрекінговой системі для ролей програмістів і тестувальників.

e. Після завершення розробки провести оцінку покриття коду всіма модульними тестами.

6. Тестувальникам (основна роль):

a. Оновити оформлення зі сховищ до актуальної версії;

b. Вибрати, які методи класів потребують тестування, і визначити набір вхідних параметрів (переробити методи з лабораторної роботи № 4).

c. Створити модульні тести на вибрані методи.

d. Реалізувати в тестових методах класи-обгортки.

e. Дописати змінені тести з урахуванням вхідних параметрів.

f. Запустити тести на виконання.

7. Програмістам (допоміжна роль):

a. Оновити оформлення зі сховищ до актуальної версії.

b. У створених тестувальниками тестах-пустушках додати ініціалізацію макетів замінних класів з використанням Moq.

c. Описати в вигляді коментаря до кожного методу свого тестованого класу, в яких тестових наборах даних потребує даний метод.

d. Зафіксувати зміни на сервері.

8. Оформити звіт, перевірити у викладача і захистити роботу.

9. Зберегти файл, який містить звіт з лабораторної роботи, в системі Mentor (mentor.khai.edu).

Презентація.

Презентація проекту учасниками команди. Кожен з учасників команди готує доповідь про виконану роботу, а також про подальше планування своїх дій.

Зміст звіту:

1. Постановка завдання.

2. Тімліду:

a. Опис обраної бібліотеки для Mock прототипування.

b. Список виданих завдань у вигляді скріншоту призначених тікетів.

c. Оцінка покриття коду всіма модульними тестами.

3. Тестувальникам:

a. Список модульних тестів.

b. Опис тестових випадків для кожного реалізованого тесту.

c. Лістинг коду модульних тестів. При необхідності – пояснення до реалізації тестів.

d. Результат запуску тестів.

e. Звіт про тестування.

4. Програмістам:

a. Лістинг коду, що описує ініціалізацію макетів замінних класів в тестах-пустушках. При необхідності додати пояснення до реалізації.

b. Коментарі до кожного методу свого тестованого класу, яких тестових наборах даних потребує даний метод.

5. Активність роботи над завданнями (опис, скріншоти).

6. Висновки.

Лабораторна робота № 6.
Створення інсталятора. Підпис інсталятора сертифікатом

Мета лабораторної роботи. Створити інсталятор для розробленого програмного продукту. Розібратися в правилах версіонування й унікальних кодах продукту. Підписати інсталятор сертифікатом.

Порядок виконання роботи:

1. Прочитати теоретичний матеріал до лабораторної роботи.
2. Завантажити та встановити WiX v3.6: http://wix.codeplex.com/releases/view/75656
3. Створити проект Windows Installer XML Setup Project.
4. Підключити до проекту WixUIExtension.dll для використання стандартних настановних діалогів.
5. Використовуючи заготовлений шаблон файлу Product.wxs (Додаток А) описати дистрибутив інсталяції свого програмного продукту (замінити всі коди продукту, склад дистрибутива, назву продукту, при необхідності версію, ліцензійну угоду).
6. Зібрати робочий msi інсталятор.
7. Переконатися в працездатності інсталятора.
8. Створити самопідписний сертифікат (self-signed).
9. Підписати інсталятор сертифікатом.
10. Оформити звіт, перевірити у викладача і захистити роботу.
11. Зберегти файл, який містить звіт з лабораторної роботи, в системі Mentor (mentor.khai.edu).

Зміст звіту:

1. Постановка завдання.
2. Лістинг файлу Product.wxs. При необхідності додати пояснення до реалізації.
3. Скріншоти послідовності процесу інсталяції.
4. Скріншот списку встановлених програм з Вашою встановленою програмою.
5. Висновки.

Розрахунково-графічна робота.
Аналіз актуальності використання новітніх інформаційних технологій розроблення програмного забезпечення.
Налаштування серверу безперервної інтеграції.

Проведення демо за підсумками відповідної ітерації:

1 Теоретичні відомості:

- лямбда вирази;
- системи контролю версій (системи управління версіями);
- віхи проекту (Milestone);
- поняття mock-об'єктів.

2 Вимоги до проекту. ТЗ на розроблювальне ПЗ.

3 Календарне планування: реалізувати GUI; реалізувати прототипи компонентів архітектури проекту; реалізувати прототип програми, що моделює реалізовану архітектуру; доопрацювати прототипи компонентів до робочого стану; підготувати прототипи до тестування; провести тестування програмного забезпечення.

4 Архітектурне проектування. Структура рішення (Солюшн).

5 Детальне проектування.

6 Розробка: реалізація інтерфейсу; реалізація класів; реалізувати клас-обгортку.

7 Тестування:

- тестування незалежних класів;
- тестування залежних класів;
- лістинг коду, що описує ініціалізацію макетів замінних класів в тестах-пустушках;
- результати тестування.

8 Висновки.

Мета розрахунково-графічної роботи. Провести аналіз актуальності використання новітніх інформаційних технологій розроблення програмного забезпечення, а також інтеграція проекту до систем безперервної інтеграції.

Порядок виконання роботи:

1. Тему розрахунково-графічної роботи (РГР) можна вибрати можна вибрати із запропонованого списку або самостійно. Основна вимога – продемонструвати сучасні підходи до розробки програмного забезпечення (технології, методи, інструментальні засоби, фреймворки) на реальних існуючих проектах.

2. Кожна тема домашнього завдання – унікальна. Варіанти вибрати та розділити між собою самостійно.

Наприклад, проект «Інформаційна технологія розпізнавання автомобільних номерів» містить наступне:

1) Код написаний в VisualStudio 2017 – backend, VisualCODE – frontend

2) frontend технології:

- Node js
- Redux
- React

3) backend технології:

- .Net Core
- SQL server

4) Проект розпізнавання

- Azure WebJobs
- HaarCascade
- Tesseract
- Open CV

і ще Google Maps і шаблони проектування:

1) трирівнева архітектура,

2) Dependency injection

3) Unit of Work (патерн репозиторій)

System.Drawing

3. Тобто або вибираємо актуальний популярний проект і описуємо використовувані при його розробці парадигми програмування, або, навпаки, для початку визначаємося з мовою розробки програмного забезпечення, а потім показуємо, в яких проектах застосовується і для досягнення яких цілей.

4. Можна також продемонструвати налаштування одного з існуючих серверів інтеграції: TeamCity, Jenkins або будь-якого іншого. Наприклад, OpenShift + Jenkins + Bitbucket – розгортання середовища для складання, тестування і публікації додатків з використанням платформи OpenShift на прикладі PHP проекту.

5. Оформити звіт, перевірити у викладача і захистити роботу.

6. Зберегти файл, який містить звіт з розрахункової роботи, в системі Mentor (mentor.khai.edu).

Зміст звіту:

РГР повинна містити:

- титульний лист;
- зміст;
- постановку завдання;
- теоретичні відомості з обраної тематики;
- приклади використання обраної технології (методу, фреймворка, середовища розробки): в яких існуючих проектах використовувалися до поточного моменту часу, перспективи використання, приклади практичної реалізації (наприклад, доступні на веб-хостингах або opensourse);
- етапи налаштування серверу безперервної інтеграції (детальний опис та скріншоти);
- висновки з розрахункової роботи;
- РГР має містити перелік використаної літератури, посилання на джерела, в тому числі посилання на відео-ресурси, веб-хостинги;
- Обсяг РГР – 20-25 сторінок тексту.

РГР має бути оформлено відповідно до вимог «Нормоконтролер» (методичні рекомендації В. М. Павленко).

МЕТОДИЧНІ ВКАЗІВКИ ДО САМОСТІЙНОЇ РОБОТИ СТУДЕНТІВ

Самостійна робота студента (СРС) – це форма організації навчального процесу, при якій заплановані завдання виконуються студентом під методичним керівництвом викладача, але без його безпосередньої участі.

СРС с основним засобом засвоєння навчального матеріалу під час поза аудиторної навчальної роботи.

Самостійна робота студента спрямована на закріплення теоретичних знань, отриманих студентами за час навчання, їх поглиблений, набуття і удосконалення практичних навичок та умінь відповідно до обраного напряму підготовки.

Відтак, метою самостійної роботи студентів є набуття додаткових знань, перевірка отриманих знань на практиці, вироблення фахових та дослідницьких вмінь та навичок.

Самостійна робота студентів включає:

− підготовку до аудиторних занять (лекцій, практичних);

− виконання завдань з навчальної дисципліни протягом семестру;

− роботу над окремими темами навчальних дисциплін, які згідно з робочою навчальною програмою дисципліни винесені на самостійне опрацювання студентів;

− підготовку до підсумкових контрольних випробувань.

Самостійна робота студента над засвоєнням навчального матеріалу дисципліни може виконуватися у бібліотеці, навчальних кабінетах, а також в домашніх умовах.

Самостійна робота студента забезпечується системою навчально-методичних засобів, передбачених для вивчення конкретної навчальної дисципліни.

Зміст самостійної роботи студента над конкретною дисципліною визначають навчальна програма, завдання та вказівки викладача.

Таким чином, самостійна робота є організованою викладачем активною діяльністю студента, направленою на виконання поставленої дидактичної мети. За своєю суттю самостійна робота є активною розумовою діяльністю студента, пов’язаною з виконанням навчального завдання.

Під час самостійної роботи студенти мають змогу краще використати свої індивідуальні здібності. Вони вивчають, конспектують літературні джерела, за потреби повторно перечитують, звертаються до відповідних довідників і словників.

СРС здійснює і виховний вплив на студентів, сприяючи формуванню і розвитку необхідних моральних якостей.

№ з/п	Назва теми	Кількість годин
1	Налагодження багтрекінгової системи	33
2	Підняття серверу збірки проекту	17
3	Налагодження TeamCity	27
4	Інтеграція проекту до систем безперервної інтеграції	23
Разом		100

ПИТАННЯ, ТЕСТИ ДЛЯ КОНТРОЛЬНИХ ЗАХОДІВ

Змістовий модуль 1 (Системи контролю версій):

1. Дати визначення системи контролю версій (із зазначенням основного призначення).

2. Дати коротку характеристику основних можливостей систем контролю версій.

3. Область (сфери) застосування систем контролю версій. Наведіть приклади (незгірш від трьох).

4. Дати визначення гілки (branch).

5. Дати визначення набору змін (changeset, changelist, activity).

6. Дати визначення фіксації змін (check-in, commit, submit), вилучення (check-out, clone) і злиття (merge, integration).

7. Дати визначення конфлікту.

8. Дати визначення переносу точки розгалуження (rebase).

9. Дати визначення сховища (repository, depot), версії (revision) і робочої копії (working copy) документа (ів).

10. Дати визначення відкладання змін (shelving) і синхронізації (update, sync).

11. Дати визначення мітки (tag, label) і стовбура (trunk, mainline, master).

12. Дати характеристику робочої копії на прикладі Subversion.

13. Базові принципи розробки ПЗ в VCS.

14. Моделі версіонування. Проблема втрати змін. Привести приклад (бажано намалювати схему).

15. Охарактеризуйте процес внесення правок і стан файлової системи після публікації змін.

16. Відстеження сховища робочими копіями. Дві найважливіші властивості, інформацію про яких записує Subversion.

17. Перерахуйте стани, в яких може перебувати робочий файл.

18. Стан файлу «Не змінювався і не застарів».

19. Стан файлу «Змінювався локально і не застарів».

20. Стан файлу «Не змінювався і застарів».

21. Стан файлу «Змінювався локально і застарів».

22. Архітектура системи контролю версій. Реплікація та метадані.

23. Архітектура системи контролю версій. Зберігання даних в формі ациклічного орієнтованого графа (DAG).

24. Архітектура системи контролю версій. Зберігання даних на основі відмінностей файлів.

25. Архітектура системи контролю версій. Способи поширення даних.

26. Злиття. Об'єднання змін.

27. Злиття. Основні принципи.

28. Злиття. Конфлікти. Приклад.

29. Механізми вирішення конфліктів.

30. Відгалуження і мітки. Злиття діапазону ревізій. Навести приклад.

31. Відгалуження і мітки. Злиття двох різних дерев. Навести приклад.

32. Створення відгалуження або мітки.

33. Опишіть механізм створення відгалуження на прикладі tortoise hg (або будь-який інший використовуваної Вами СКВ).

34. Опишіть механізм перемикання між гілками проекту на прикладі tortoise hg (або будь-якої іншої використовуваної Вами СКВ).

35. Опишіть механізм злиття гілок проекту на прикладі tortoise hg (або будь-який інший використовуваної Вами СКВ).

36. Дати коротку характеристику архітектури системи контролю версій на прикладі SVN або GIT, або Mercurial.

37. Модель блокування-зміна-розблокування при роботі з репозиторієм вихідного коду. Привести приклад (бажано намалювати схему).

38. Модель Копіювання-Зміна-Злиття при роботі з репозиторієм вихідного коду. Привести приклад (бажано намалювати схему).

39. «Фатальні» етапи процесу безперервної інтеграції.

40. Що зберігається в репозиторії артефактів?

41. Принципові відмінності відгалуження і мітки в SVN.

42. Віхи проекту (Milestone). Зв'язок віх проекту з нарощуванням функціонала, плануванням версій і розподілом завдань.

43. Які метрики застосовують для оцінки покриття коду модульними тестами?

44. Розподіл завдань (тікетірованіе і прив'язка завдань до коммітов).

45. Робоча копія сховища (що з себе представляє і хто з нею працює).

Змістовий модуль 2 (Безперервна інтеграція, тестування):

1. Яке місце модульне тестування займає в процесі безперервної інтеграції при розробці програм? Дати коротку характеристику.
2. Опишіть, у чому суть процесу безперервної інтеграції. Основні принципи.
3. Вимоги до проекту, необхідні для здійснення безперервної інтеграції.
4. Переваги безперервної інтеграції.
5. Недоліки безперервної інтеграції.
6. Основні принципи CI. «Кожна зміна має інтегруватися».
7. Основні принципи CI. «Швидка збірка».
8. Основні принципи CI. «Інтеграція на виділеному сервері».
9. Основні принципи CI. «Проведення автоматичного тестування».
10. Сервер збірки TFBuild. Приклад.
11. Розгортання сервера збірки.
12. Схема організації сервера інтеграції.

13. Збірка за розкладом. Характеристика.
14. Збірка за розкладом. Переваги.
15. Збірка за розкладом. Недоліки.
16. Автоматизація складання. Характеристика.
17. Просунута автоматизація збірки. Основні можливості.
18. Просунута автоматизація збірки. Переваги.
19. Просунута автоматизація збірки. Типи.
20. Вимоги, що пред'являються до систем збірки.
21. CI. Trigger – запуск збірок.
22. CI. Update – оновлення версії коду.
23. CI. Analyse – попередній аналіз.
24. CI. Build – збірка.
25. CI. Archive – збереження проекту.
26. CI. Report – Етап генерації і публікації звітів. Організація етапу.
27. CI. Report – Етап генерації і публікації звітів. Зміст звітів.
28. Continuous Improvement – Безперервне вдосконалення. Схема. Приклади.
29. Успішні тести на прикладі TeamCity.
30. Провалені тести на прикладі TeamCity.
31. CI. Deploy – розгортання.
32. Визначення та цілі тестування.
33. Види тестування. Класифікація по об'єкту тестування.
34. Види тестування. Класифікація по доступу до системи.
35. Види тестування. Класифікація за ступенем автоматизації.
36. Види тестування. Класифікація за ступенем ізольованості компонентів.
37. Види тестування. Класифікація за часом проведення тестування.
38. Види тестування. Класифікація за ступенем підготовленості до тестування.

39. Алфа-тестування. Характеристика.
40. Бета-тестування. Характеристика.
41. Мутаційне тестування. Навести приклад.
42. Тестування. Покриття коду. Вимірювання покриття.
43. Тестовий випадок. Детальна характеристика.
44. Регресійне тестування. Характеристика.
45. Три основні типи регресивного тестування згідно Сему Канер. Дати характеристику.
46. Розподілені системи контролю версій. Схема архітектури. Приклади.
47. Організація процесу безперервної інтеграції. Етапи процесу безперервної інтеграції. Перерахувати.
48. Централізовані системи управління версіями. Схема архітектури. Приклади.
49. Для чого використовують гілки (brunch) в роботі з системами контролю версій?
50. Цикл використання Subversion.
51. За яким подіям може починатися цикл інтеграції?
52. Які вимоги пред'являються до інтеграційного сервера?

Змістовий модуль 3 (Версіонування програмних продуктів):

1. Опишіть логічну структуру інсталяційного пакета.
2. Для чого призначений Secure Socket Layer сертифікат. Наведіть приклади використання.
3. Опишіть патерн ізолювання Mock Object.
4. Які параметри характеризують встановлений програмний продукт в системі?
5. У чому відмінність між сертифікатами типу Secure Socket Layer і Code Signing.

6. Охарактеризуйте ідею ізоляції в тестуванні. Які об'єкти допомагають здійснювати ізоляцію класів один від одного?

7. Перелічіть і охарактеризуйте етапи процесу установки ПЗ.

8. Для чого призначені Code Signing сертифікати. Наведіть приклади використання.

9. Опишіть патерн проектування «інверсія залежностей» («впровадження зал.»).

10. Опишіть правила складання і подальшої зміни номера версії продукту.

11. Перерахуйте особливості сертифіката розробника.

12. Які існують шаблони реалізації інверсії управління?

13. Як проводиться оцінка покриття коду тестами і які характеристики проекту в результаті оцінки можна отримати?

14. Центри сертифікації.

15. Як відбувається процес видачі сертифікатів?

16. Фіксація довірених сертифікатів в кореневих каталогах Windows.

17. Які способи впровадження залежностей існують?

18. Охарактеризуйте Minor upgrade.

19. За якими правилами змінюються наступні характеристики ПЗ при такому типі оновлень (версія продукту, код продукту, код поновлення).

20. Різниця між самоподпісанного безкоштовним і платними сертифікатами, виданими центром сертифікації.

21. Опишіть шаблон інверсії управління «фабричний метод».

22. Охарактеризуйте Major upgrade.

23. За якими правилами змінюються наступні характеристики ПЗ при такому типі оновлень (версія продукту, код продукту, код поновлення).

24. Для чого використовують самоподпісний сертифікат (self-signed). На яких етапах розробки ПЗ.

25. Опишіть патерн впровадження залежностей через інтерфейс (Interface Injection)

26. Охарактеризуйте Small update.

27. За якими правилами змінюються наступні характеристики ПЗ при такому типі оновлень (версія продукту, код продукту, код поновлення).

28. Намалюйте послідовність дій і опишіть процес підпису коду серітіфікатом розробника.

29. Які метрики застосовують для оцінки покриття коду модульними тестами?

30. Для чого використовується WIX?

31. Опишіть основні принципи роботи з WIX.

32. Намалюйте послідовність дій і опишіть процес перевірки підписаного коду серітіфікатом розробника.

33. Для чого і як при проведенні модульного тестування використовують класи Assert?

34. Для чого використовують Timestamp або тимчасову мітку при підпису коду сертифікатом розробника?

35. Вкажіть параметри, які будуть змінені в результаті перерахованих видів оновлень:

	Package Code	Product Version	Product Code	Upgrade Code
Small Update				
Minor Upgrade				
Major Upgrade				

Прокоментуйте свій вибір. Що характеризує кожен з параметрів?

ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Класифікація інформаційних технологій;
2. Використання інформаційних технологій;
3. IT архітектура підприємства;
4. Розробка інформаційних технологій (ПЗ);
5. Супровід інформаційних технологій протягом їх життєвого циклу;
6. Інформаційні технології в науковій діяльності;
7. Інформаційні технології в інноваційній діяльності;
8. У чому полягає економія часу при використанні системи контролю версій?
9. У чому переваги використання системи контролю версій?
10. Що таке Git (Mercurial)?
11. Як почати використовувати git (Mercurial)?
12. Як почати використовувати GitHub (Bitbucket)?
13. Основні (найбільш часто використовувані) команди Git (Mercurial).
14. Які сервіси існують для Git (Mercurial)?
15. Як працювати з локальним репозиторієм?
16. Як працювати з розподіленим репозиторієм?
17. Який головний мотив використання системи контролю версій?
18. Які фактори впливають на вибір системи контролю версій?
19. Як впровадити використання системи контролю версій в команді?
20. Що не влаштовує розробників в Git (Mercurial)?
21. Чому деякі вибирають інші менш популярні рішення?
22. Наскільки поширене використання систем контролю версій для управління іншими файлами, а не тільки кодом?
23. Перелічіть SOLID принципи.
24. SRP: The Single Responsibility Principle.
25. OCP: The Open Closed Principle.
26. LSP: The Liskov Substitution Principle.

27. ISP: The Interface Segregation Principle.
28. DIP: The Dependency Inversion Principle.
29. Класичні системи контролю версій (СКВ).
30. Базові принципи роботи з системами контролю версій.
31. Історія появи, файлові СКВ, CVS.
32. Основні принципи роботи (команди checkout, commit, update).
33. Колективна розробка з використанням СКВ.
34. Проблеми колективної розробки з використанням СКВ.
35. Вирішення конфліктів (команда merge).
36. Гілки і теги (команди branch і tag).
37. Нюанси розробки з використанням гілок.
38. Розподілені системи контролю версій (РСКВ).
39. Передумови до появи РСКВ.
40. Переваги РСКВ.
41. Устрій РСКВ на прикладі Git і Mercurial.
42. Сценарії використання РСКВ.
43. Просунуте використання СКВ.
44. Інтеграція СКВ в проектну інфраструктуру.
45. Розгортання СКВ.
46. Системи іменування гілок і версій.
47. Необхідність багтрекеру.
48. Історія появи багтрекеру, JIRA.
49. Життєвий цикл бага.
50. Сучасні багтрекери.
51. Варіації життєвого циклу завдання.
52. Інструменти статичного аналізу коду.
53. Можливості та обмеження статичних аналізаторів коду.
54. Статичний аналіз коду на C на прикладі linq.
55. Статичний аналіз коду на Java на прикладі PMD.

56. Метрики програмного забезпечення
57. Принципи модульного тестування.
58. Бібліотеки для модульного тестування.
59. Вимоги до модульних тестів.
60. Аналіз тестового покриття (code coverage).
61. Поширені підходи до модульного тестування. Test Drived Development.
62. Подальший розвиток ідеї модульних тестів. Behavior Driven Development.
63. Необхідність автоматизації збирання
64. Системи збирання першого покоління. Make, rake, cake, SBT.
65. Перші декларативні системи збирання. Apache Ant.
66. Подальший розвиток декларативних систем збірки. Apache Maven.
67. Системи безперервної інтеграції на прикладі Team City.
68. Архітектура інструментів для безперервної інтеграції.
69. Безперервна інтеграція великого проекту.
70. Складності безперервної інтеграції.
71. Інші інструменти безперервної інтеграції.
72. Введення до Configuration management.
73. Проблеми колективної розробки.
74. Головне питання конфігураційного управління.
75. Роль CM в управлінні проектом.
76. Інструменти CM.
77. Ручний вимір продуктивності.
78. Аналіз лог-файлів.
79. Спеціалізовані профайлери на прикладі cProfile.
80. Обмеження профайлером.
81. Автоматизація технічної підтримки (Trouble ticket systems).

82. Організація взаємодії з користувачами.

83. Інструменти автоматизації технічної підтримки

84. Способи організації знань (списки питань, що часто ставляться (FAQ), HOWTO).

85. Системи генерації документації.

86. Необхідність автоматичної генерації документації.

87. Генератори документації на прикладі doxygen і javadoc.

88. Обмеження генераторів документації.

89. Технічна інфраструктура open source проектів.

90. Особливості інфраструктури open source проектів.

91. Взаємодія між учасниками проекту (IRC, листи розсилки).

92. Зберігання коду (sourceforgfe, github, google code).

93. Нумерація версій.

94. Процес розробки програмного забезпечення з використанням Team Foundation Server.

95. Процес розробки програмного забезпечення з використанням TeamCity Server.

96. Процес розробки програмного забезпечення з використанням Jenkins Server.

97. Архітектура Team Foundation Server.

98. Архітектура TeamCity Server.

99. Архітектура Jenkins Server.

100. Безперервна інтеграція баз даних.

101. Безперервна перевірка.

102. Безперервна інспекція.

103. Безперервне розгортання.

104. Безперервна зворотний зв'язок.

105. Що таке профілювання?

106. Які види профілювальник існують?

107. Як визначити вузькі місця в додатку за допомогою Профілювальника?

108. На що важливо звертати увагу при профілювання візуального інтерфейсу WPF додатків?

109. Навіщо застосовуються інструменти статичного аналізу?

110. Які види статичного аналізу відомі?

111. У чому полягає головна проблема інструментів статичного аналізу?

112. Які перспективи використання інструментів статичного аналізу?

113. Які категорії вбудованого аналізатора Visual Studio Вам відомі?

114. Навіщо застосовується документування вихідного коду?

115. Які види документації Вам відомі?

116. До якого виду документації можна віднести UML-діаграми?

117. Які синтаксичні конструкції Вам відомі для документування C # коду?

118. Яким чином працюють автоматичні генератори документації?

119. Що таке репозиторій?

120. Що таке робоча копія?

121. Опишіть призначення команди «Checkout».

122. Опишіть призначення команди «Update».

123. Опишіть призначення команди «Commit».

124. Опишіть призначення команди «Show log».

125. Опишіть механізм контролю версій.

СЛОВНИК ТЕРМІНІВ

Конфлікт (conflict) – ситуація, коли кілька користувачів змінили одну і ту ж ділянку документа. Конфлікт виявляється, коли один користувач зафіксував свої зміни, а другий намагається зафіксувати, і система сама не може коректно злити конфліктуючі зміни. Оскільки програма може бути недостатньо розумною для того, щоб визначити, яка зміна є «коректною», другому користувачеві потрібно самому вирішити конфлікт (resolve).

Основна версія (head) – найсвіжіша версія для гілки / стовбура, що знаходиться в сховищі. Скільки гілок, стільки основних версій.

Злиття (merge, integration) – об'єднання незалежних змін в єдину версію документа. Здійснюється, коли двоє людей змінили один і той же файл або при перенесенні змін з однієї гілки на іншу.

Перенесення точки розгалуження (rebase) (версії, від якої починається гілка) на більш пізню версію основної гілки. Наприклад, після випуску версії 1.0 проекту в стовбурі триває доопрацювання (виправлення помилок, дороблення наявного набору функцій), одночасно починається робота над новою функціональністю в новій галузі. Через деякий час в основній гілці відбувається випуск версії 1.1 (з виправленнями); тепер бажано, щоб гілка розроблення нової функціональності включала зміни, що відбулися в стовбурі. Взагалі, це можна зробити базовими засобами, за допомогою злиття (merge), виділивши набір змін між версіями 1.0 і 1.1 і злив його в гілку. Але при наявності в системі підтримки перебазування гілки ця операція робиться простіше, однією командою: за командою rebase (з параметрами: гілкою і новою базовою версією) система самостійно визначає потрібні набори змін і виробляє їх злиття, після чого для гілки базовою стає версія 1.1; при подальшому злитті гілки зі стовбуром система не розглядає повторно зміни, внесені між версіями 1.0 і 1.1, оскільки гілка логічно вважається виділеною після версії 1.1.

Сховище документів (Repository, Depot) – місце, де система управління версіями зберігає всі документи разом з історією їх змінення та іншою службовою інформацією.

Версія документа (Revision). Системи управління версіями розрізняють версії за номерами, які надаються автоматично.

Відкладання змін (Shelving). Надана деякими системами можливість створити набір змін (changeset) і зберегти його на сервері без фіксації (commit'a). Відкладений набір змін доступний для читання іншим учасникам проекту, але до спеціальної команди не входить в основну гілку. Підтримка відкладання змін дає можливість користувачам зберігати незавершені роботи на сервері, не створюючи для цього окремих гілок.

Мітка (Tag, Label) – це символічне ім'я для групи документів, яке можна надати певній версії документа. Мітка описує не тільки набір імен файлів, але і версію кожного файла. Версії занесених до мітки документів можуть відноситися до різних моментів часу.

Стовбур (Trunk, Mainline, Master) – головна галузь розроблення проекту. Політика роботи зі стовбуром може відрізнятися від проекту до проекту, але в цілому вона така: більшість змін вноситься у стовбур; якщо потрібна серйозна зміна, здатна призвести до нестабільності; створюється гілка, яка зливається зі стовбуром, коли нововведення буде достатньою мірою випробувано; перед випуском чергової версії створюється «релізна» гілка, в яку вносяться тільки виправлення.

Синхронізація (Update, Sync.) робочої копії до деякого заданого стану сховища. Найчастіше ця дія означає оновлення робочої копії до актуального стану сховища. Однак, за необхідності, можна синхронізувати робочу копію і до попереднього його стану, ніж поточний.

Робоча (локальна) копія документів **(Working copy).**

У документації Visual Studio використовується ряд термінів для опису функцій і понять системи управління версіями [1]. У таблиці А.1 подано деякі загальні терміни.

Таблиця А.1 – Словник термінів

Термін	Значення
Основна версія	Серверна версія файла, з якої виймається локальна версія
Прив'язка	Інформація, яка зіставляє робочу папку рішення або проекту на диску у відповідній папці в базі даних
Розгалуження	Процес створення нової версії або гілки для спільно використовуваного файла або проекту, що знаходиться під управлінням системи управління версіями. Коли гілка створена, дві версії, що знаходяться під управлінням системи управління версіями, будуть мати загальний журнал до певної точки, який відрізняється від журналів після цієї точки
Конфлікт	Наявність двох або декількох відмінних змін на тому самому рядку коду, коли два або більше розробників витягли і змінили один і той же файл
Підключення	Поточна лінія передачі даних між системою управління версіями (наприклад, Visual Studio) і сервером бази даних системи управління версіями
База даних	Сховище, де розміщуються основні копії, журнал, структури проектів і призначена для користувача інформація. Проект завжди міститься в одній базі даних. Кілька проектів можуть зберігатися в одній базі даних і можуть використовувати кілька баз даних. Інші терміни, які використовують для бази даних, – це репозитарій і сховище

Закінчення таблиці 1.1

Термін	Значення
Журнал	Використовується для запису змін у файлі відразу ж після його додавання в систему управління версіями. За допомогою управління версіями можна повернутися в будь-яку точку в журналі файла і відновити його в тому вигляді, в якому він був у цій точці
Мітка	Ім'я, визначене користувачем, яке приєднане до конкретної версії елемента з керуванням версіями
Локальна копія	Файл у робочій папці користувача, в якому зберігаються зміни, до тих пір, поки не проводиться повернення. Локальна копія іноді визначається як робоча копія
Головна копія	Найчастіше – це повернута версія файла з контролем версій, на противагу до локальної копії файла в робочій папці користувача. Іншими термінами для головної копії є серверна версія і версія бази даних
Злиття	Процес об'єднання в новій версії файла відмінностей у двох або більше змінених версіях файла. Злиття можна застосовувати до різних версій одного і того ж файла або до змін, зроблених у тому ж самому файлі
Загальний файл	Файл, який має версії, що знаходяться в декількох розташуваннях системи управління версіями. Інші терміни для загального файла – це копія і ярлик
Корінь рішення	Порожня папка в базі даних, яка містить всі елементи рішення з керуванням версіями. За замовчуванням це папка <ім'я_решенія> .root
Супер-уніфікований корінь	Віртуальний контейнер, в якому розміщуються всі проекти і файли в рішенні з керуванням версіями. Наприклад, [SUR]: \ – це суперуніфікований корінь рішення з керуванням версіями, який містить проекти, розташовані в [SUR]: \ C: \ Solution \ ProjOne і [SUR]: \ D: \ ProjTwo
Уніфікований корінь	Шлях до батьківського каталогу для всіх робочих папок і файлів у рішенні або проект з керуванням версіями. Наприклад, C: \ Solution – це уніфікований корінь рішення з керуванням версіями, який містить файли, розміщені в C: \ Solution, C: \ Solution \ ProjOne і C: \ Solution \ ProjTwo
Робоча папка	Сховище, де зберігаються локальні копії елементів з керуванням версіями (зазвичай на комп'ютері користувача). Інший термін для робочої папки – робоча область

ПРИКЛАДИ ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ

Лабораторна робота № 1.

Хід роботи

1 Калькулятор

1.1 Скріншоти створеного сховища та груп користувачів

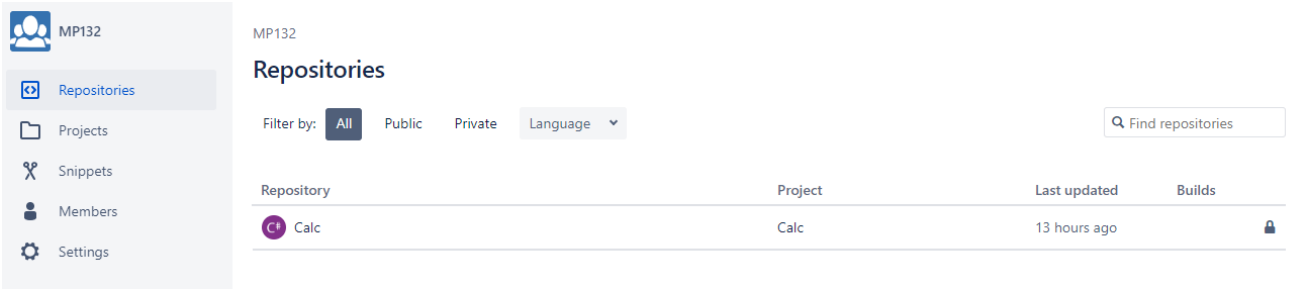

Рисунок 1.1 – Створений репозиторій

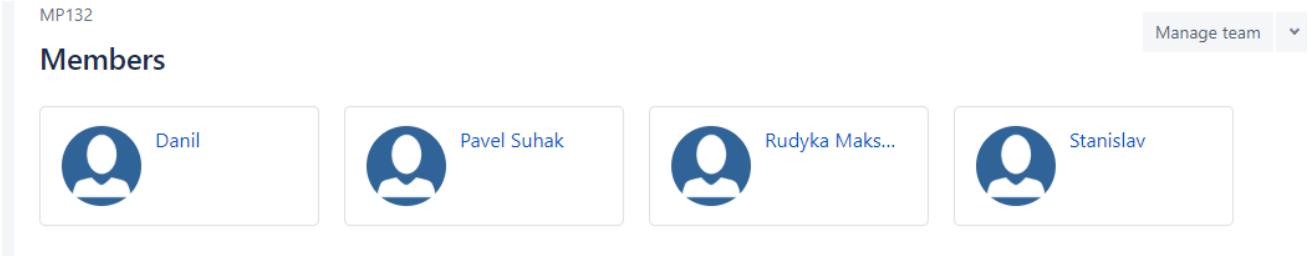

Рисунок 1.2 – Група користувачів

1.2 Матриця відповідальних

Види робіт	Сугак	Трощій	Рудика	Скляр
розподіл завдань	R			
Розробка класів		R		
Розробка форми		R		
тестування класів			R	
тестування форми				R

* R – відповідальний (responsibility)

1.3 Історія коммітів

Author	Commit	Message	Date	Builds
Pavel Suhak	fa78121 D	Слияние	13 hours ago	
Pavel Suhak	9130393 D	Closed branch Visual_test Visual_test	2 days ago	
Stanislav	741b3f4 D	Протестировано и исправлено некорректное введение данных... Visual_test	2 days ago	
Pavel Suhak	3e408b0 D	Слияние	2 days ago	
Pavel Suhak	7a606a2 D	Closed branch UnitTest2 UnitTest2	2 days ago	
Rudyka Maksim	2e55d4e D	Были испрвлены ошибки которые не были исправлены в первом... UnitTest2	2 days ago	
Pavel Suhak	338840b D	Closed branch UnitTest UnitTest	2 days ago	
Pavel Suhak	fb78c9c D	Слияние	2 days ago	
Rudyka Maksim	01de1f5 D	Было созданно 5-ть UnitTest-ов, которые проверяли работу метод... UnitTest	2 days ago	
Pavel Suhak	a9d13ae D	Closed branch TypicalCalculator TypicalCalculator	2 days ago	
Pavel Suhak	0817ef9 D	Слияние	2 days ago	
Danil	7f17469 D	Обновлена верхняя панель. TypicalCalculator	2 days ago	
Danil	bfb5102 D	Удалены лишние файлы.	2 days ago	
Danil	c080eed D	Проект с классами заменен на библиотеку классов. Проект "Типичный кальку...	2 days ago	
Danil	dbef81a D	Созданы классы, реализующие интерфейсы. Классы и интерфейсы сгруппиров...	2 days ago	
Pavel Suhak	c45bf27 D	Солюшн и интерфейсы	2 days ago	

Рисунок 1.3 – Історія коммітів

1.4 Помилки

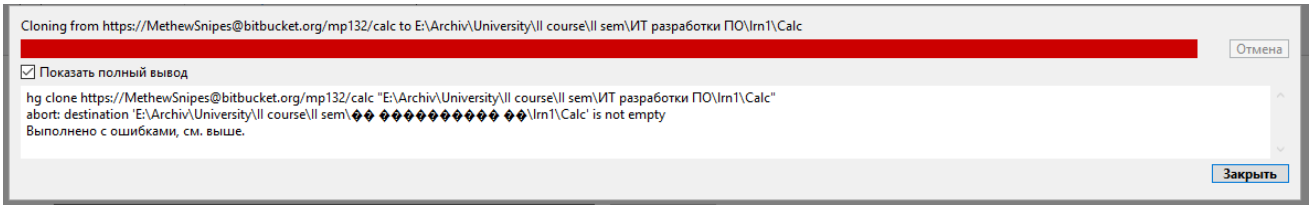

Рисунок 1.4 – Помилка під час злиття

1.5 Результат роботи програми

Результат роботи програми показаний на рисунку 1.5.

Лістинг програмного коду знаходиться в додатку до прикладу виконання лабораторної роботи.

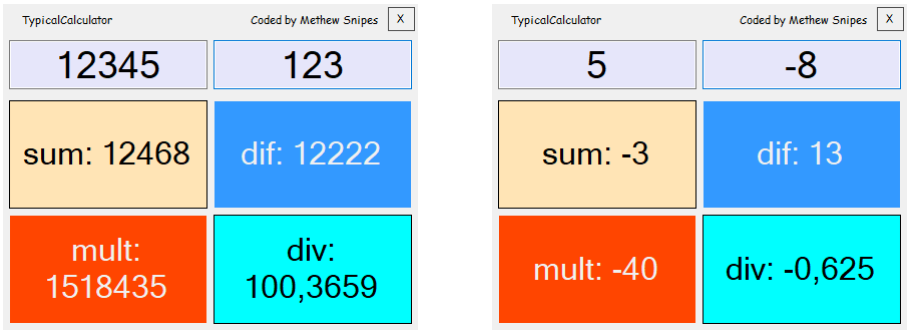

Рисунок 1.5 – Результат роботи програми

2 Генератор запитів

2.1 Опис проекту

Даний проект повинен мати наступний функціонал:
- генерація запитів на додавання в таблицю;
- додавання в базу нових атрибутів;
- додавання нових значень, які приймають існуючі атрибути.

2.2 Обгрунтування вибору проекту

Даний проект буде корисний для тестування коректного функціонування баз даних і запитів до баз даних.

Дуже часто для адекватного тестування баз даних потрібно заповнювати таблиці адекватною кількістю записів.

Аудиторія поточного проекту - розробники і тестувальники БД.

2.3 Обгрунтування вибору системи контролю версій

2.3.1 Переваги Mercurial:

- у вас є повна копія сховища на вашому комп'ютері, так що вам не доведеться покладатися на резервні копії сервера в разі, якщо сервер «впаде»;
- якщо у вас немає підключення до інтернету, ви все ще можете працювати і фіксувати зміни в локальному репозиторії. Коли з'єднання знову відновиться, можна відправити зміни на сервер.
- mercurial організовує ревізії як набори змін, які дозволяють дуже легко гілок / об'єднати кодову базу.

Багато програмістів вважають за краще використовувати графічний інтерфейс до своєї системи управління версіями замість того, щоб працювати через командний рядок. Для полегшення роботи з Mercurial було створено кілька сторонніх програм з графічним інтерфейсом.

Одним з інструментів є Sourcetree, безкоштовний Git і Mercurial клієнт, пропонований Atlassian. Sourcetree - це просте у використанні додаток, яке підключається до багатьох популярних інструментів хостингу вихідного коду, таким як Bitbucket, Kiln і (для користувачів Git) GitHub.

2.3.2 Погляд техліда / PM – чому Mercurial зручніше

Пропоную розглянути наступні відмінності git vs. mercurial в контексті їх зручності для PM і комерційного проекту в цілому:
- неможливість видалення гілок, коммітов, тобто зміни історії сховища;
- рівень складності навчання;
- «Важкі» гілки, в коммітов яких гілка прив'язана безпосередньо.

Всі вищевказані плюси в якійсь мірі можуть стати і мінусами. Наприклад, сама проблемна ситуація - не варто додавати величезні файли в репозиторій. Один наш замовник вирішив, що таким чином можна додати media файли, включаючи рекламні ролики, і розмір сховища різко виріс на розмір відео, помножений на два. Видалити їх простим способом було неможливо, довелося використовувати ConvertExtension.

Другий мінус - до хорошого звикаєш швидко. Попрацювавши трохи з mercurial, повертатися на git досить складно, так як доводиться згадувати зубодробильний синтаксис для простих речей.

Третій мінус - ви, як PM, швидше за все, не зможете переконати своїх програмістів в необхідності переходу на Mercurial. Причина проста - вони вже вивчили один синтаксис, переучуватися не хочуть. Тим більше, на свідомо більш слабку по фічам систему. Є шанс, якщо стартувати новий проект.

2.3.3 Обгрунтування вибору типу сховища

Для сховища був обраний тип - private. Цей тип був обраний з-за того, що команда, яка буде займатися проектом, вже чітко сформована і втручання сторонніх осіб лише ускладнить розробку проекту, а також може з'явиться ситуація, яка посприяє зриву плану розробки проекту.

Також private дозволить приховати проект від сторонніх осіб, що дозволить виключити ситуації плагіату, а також передчасне інформування про проект в маси.

Лістинг програмного коду

MainForm.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace TypicalCalc
{
 public partial class MainForm: Form
 {
 public MainForm ()
 {
 InitializeComponent ();
 }

 private void leftFieldTB_TextChanged (object sender, EventArgs e)
 {
 double num;
 if (double.TryParse (leftFieldTB.Text, out num) && double.TryParse (rightFieldTB.Text, out num))
 {
 try
 {
 addBtn.Text = "sum:" + Math.Round (new Calc.Classes.Plus (double.Parse (leftFieldTB.Text), double.Parse (rightFieldTB.Text)). Res, 4) .ToString ();
 difBtn.Text = "dif:" + Math.Round (new Calc.Classes.Minus (double.Parse (leftFieldTB.Text), double.Parse (rightFieldTB.Text)). Res, 4) .ToString ();
 multBtn.Text = "mult:" + Math.Round (new Calc.Classes.Multiply (double.Parse (leftFieldTB.Text), double.Parse (rightFieldTB.Text)). Res, 4) .ToString ();
 divBtn.Text = "div:" + Math.Round (new Calc.Classes.Division (double.Parse (leftFieldTB.Text), double.Parse (rightFieldTB.Text)). Res, 4) .ToString ();
 }
catch {MessageBox.Show ( "Помилка"); }
 }
 else
 {
 addBtn.Text = "sum: -";
 difBtn.Text = "dif: -";
 multBtn.Text = "mult: -";
 divBtn.Text = "div: -";
 }
 }

 private void leftFieldTB_KeyPress (object sender, KeyPressEventArgs e)
```

```
 {
 if (e.KeyChar> = '0' && e.KeyChar <= '9' ||
e.KeyChar == 8 || e.KeyChar == '-' || e.KeyChar ==
',')
 {
 if ((sender as TextBox) .Name == "leftFieldTB")
 {
 if ((e.KeyChar == '-' && leftFieldTB.Text.Length ==
0) || (e.KeyChar == ',' && leftFieldTB.Text.Length>
0)
|| e.KeyChar! = '-' || e.KeyChar! = ',')
 e.Handled = false;
 else
 e.Handled = true;
 }
 else
 {
 if ((e.KeyChar == '-' && rightFieldTB.Text.Length
==  0)  ||  (e.KeyChar  ==  ','  &&
rightFieldTB.Text.Length> 0)
|| e.KeyChar! = '-' || e.KeyChar! = ',')
 e.Handled = false;
 else
 e.Handled = true;
 }
 }
 else
 e.Handled = true;
 }

 private  void  leftFieldTB_MouseClick  (object
sender, MouseEventArgs e)
 {
 if (e.Button == MouseButtons.Right)
 {
 leftFieldTB.ContextMenu = new ContextMenu ();
 rightFieldTB.ContextMenu = new ContextMenu ();
 }
 }

 private  void  headerPanel_MouseDown  (object
sender, MouseEventArgs e)
 {
 headerPanel.Capture = false;
 titleLab.Capture = false;
 credits.Capture = false;
 Message  m  =  Message.Create  (base.Handle,
0xA1, new IntPtr (2), IntPtr.Zero);
 base.WndProc (ref m);
 }

 private  void  exitBtn_Click  (object  sender,
EventArgs e)
 {
 Application.Exit ();
 }

 private  void  leftFieldTB_Leave  (object  sender,
EventArgs e)
 {
 double resul;
 if (! double.TryParse ((sender as TextBox) .Text,
out resul))
 (Sender as TextBox) .Text = "";
 }
 }
}
```

Division.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Calc.Classes
{
 public class Division: IDivision
 {
 private double left;
 private double right;
public double Res {get; }
 public Division (double left, double right)
 {
 this.left = left;
 this.right = right;
 Res = ToDivide (left, right);
 }
 public double ToDivide (double left, double right)
 {
 return left / right;
 }
 }
}
```

Minus.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Calc.Classes
{
 public class Minus: IMinus
 {
 private double left;
 private double right;
public double Res {get; }
 public Minus (double left, double right)
 {
 this.left = left;
 this.right = right;
 Res = ToSubtract (left, right);
 }
 public double ToSubtract (double left, double right)
 {
 return left - right;
 }
 }
}
```

Multiply.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Calc.Classes
{
 public class Multiply: IMultiply
 {
 private double left;
 private double right;
public double Res {get; }
```

```
 public Multiply (double left, double right)
 {
 this.left = left;
 this.right = right;
 Res = ToMultiply (left, right);
 }
 public double ToMultiply (double left, double right)
 {
 return left * right;
 }
 }
}
```

Plus.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Calc.Classes
{
 public class Plus: IPlus
 {
 private double left;
 private double right;
public double Res {get; }
 public Plus (double left, double right)
 {
 this.left = left;
 this.right = right;
 Res = ToPlus (left, right);
 }
 public double ToPlus (double left, double right)
 {
 return left + right;
 }
 }
}
```

Лабораторна робота № 2.

Хід роботи

1 Milestones

На рисунку 2.1 показані Milestones проекту.

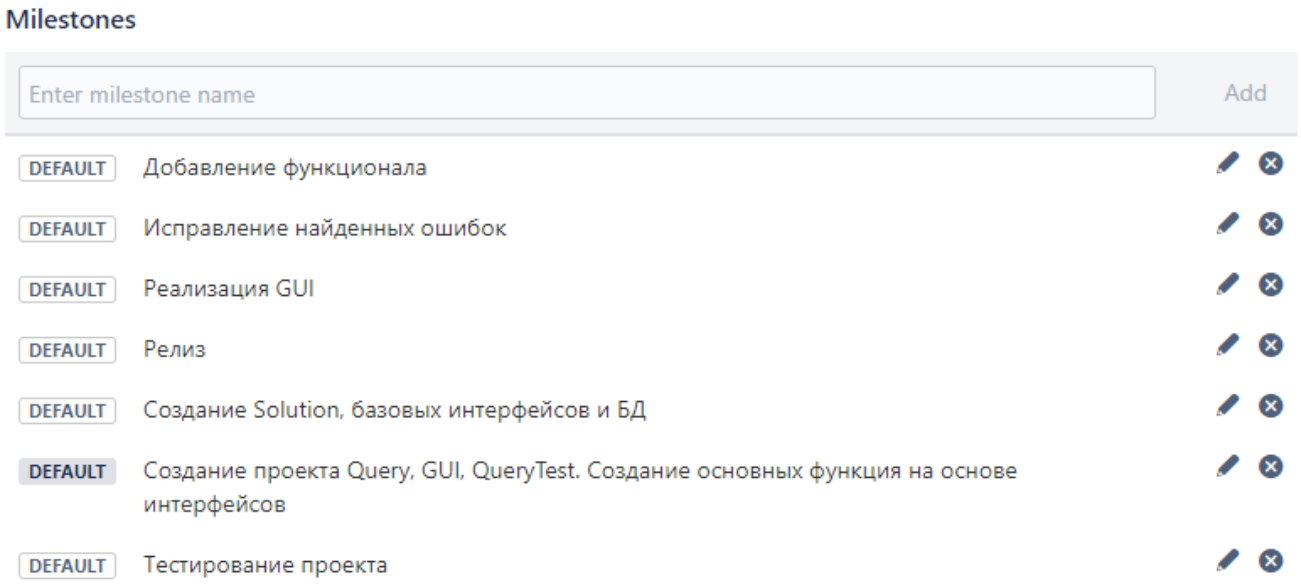

Рисунок 2.1 – Milestones (етапи проекту)

2 Issues

На рисунку 2.2 показані Issues проекту.

Issues (1–19 of 19)

Title	T	P	Status	Votes	Assignee	Milestone	Created	Updated
#19: Реализация логики GUI	☑	⌄⌄	NEW		Danil	Реализ...	32 seconds ago	32 seconds ago
#12: Тестирование функции AddNewValueToAtribute	☑	⌄⌄	RESOLVED		Максим Рудика	Тестир...	2019-04-01	a minute ago
#11: Тестирование функции AddNewAtribute	☑	⌄⌄	RESOLVED		Максим Рудика	Тестир...	2019-04-01	2 minutes ago
#10: Тестирование функции GenerateQuery	☑	⌄⌄	RESOLVED		Максим Рудика	Тестир...	2019-04-01	2 minutes ago
#5: Создание проекта для реализации интерфейсов	☑	⌃⌃	CLOSED		Danil	Создан...	2019-04-01	9 minutes ago
#6: Реализация функции GenerateQuery интерфейса IQuery	☑	⌃⌃	CLOSED		Danil	Создан...	2019-04-01	9 minutes ago
#7: Реализация функции AddNewAtribute интерфейса IQuery	☑	⌃⌃	CLOSED		Danil	Создан...	2019-04-01	9 minutes ago
#8: Реализация функции AddNewValueToAtribute интерфейса IQuery	☑	⌃⌃	CLOSED		Danil	Создан...	2019-04-01	9 minutes ago
#9: Реализация интерфейса IInterval	☑	⌄⌄	CLOSED		Danil	Создан...	2019-04-01	9 minutes ago
#18: Реализация выдачи диагностических сообщений	☑	↓	NEW		Danil	Добавл...	18 hours ago	18 hours ago
#17: Создание UI	☑	⌃⌃	NEW		Стас Скляр	Реализ...	18 hours ago	18 hours ago
#13: Создание проекта для реализации графического интерфейса пользователя	☑	⌄⌄	NEW		Стас Скляр	Создан...	2019-04-01	18 hours ago
#16: Создание проекта QueryTest для выполнения тестирования проекта	☑	⌃⌃	CLOSED		Максим Рудика	Создан...	2019-04-02	2 days ago
#14: Исправить ошибку интерфейса IQuery функции GenerateQuery	■	⊘	CLOSED		Pavel Suhak	Исправ...	2019-04-01	2019-04-02
#15: Заполнить БД ПО QueryHelper данными	☑	↑	CLOSED		Стас Скляр	Создан...	2019-04-01	2019-04-02
#4: Создать БД ПО QueryHelper	☑	⊘	CLOSED		Стас Скляр	Создан...	2019-04-01	2019-04-02
#3: Добавить в проект Core интерфейсы для всего Solution	☑	⌃⌃	CLOSED		Pavel Suhak	Создан...	2019-04-01	2019-04-02
#2: Добавить в Solution проект Core	☑	⌃⌃	CLOSED		Pavel Suhak	Создан...	2019-04-01	2019-04-02
#1: Создание пустого Solution	☑	⌃⌃	CLOSED		Pavel Suhak	Создан...	2019-04-01	2019-04-02

Рисунок 2.2 – Issues (задачі, тікети проекту)

3 Терміни

Даний проект буде «розбитий» на три основних спринти, кожен з яких – тривалістю в два тижні.

До першого спринту відносяться наступні завдання: 1-12, 14-16.

До другого спринту відносяться наступні завдання: 13, 17-19.

У третьому спринті буде виконана оптимізація коду, виправлення помилок. Список завдань буде уточнено пізніше.

Список всіх завдань для перших двох спринтів відображений в пункті 2, на рисунку 2.2.

4 Структура рішення

Рішення складається з чотирьох проектів:

– GUI - містить форму, яка представляє собою графічний інтерфейс;

– Core - бібліотека класів, яка містить необхідні інтерфейси, а саме інтерфейс IInterval - задає шаблон структури зберігання інтервалу даних і інтерфейс IQuery - задає основний функціонал ПО;

– Query - бібліотека класів, яка містить необхідні класи, а саме структура Interval, який реалізує інтерфейс IInterface і QueryGenerator, який реалізує інтерфейс IQuery, що містить 3 основні методи: генерація запиту, додавання атрибута, додавання значень атрибута;

– QueryTest - проект Unit-тестування.

5 Скріншот створеного сховища

На рисунку 2.3 показаний створений репозиторій.

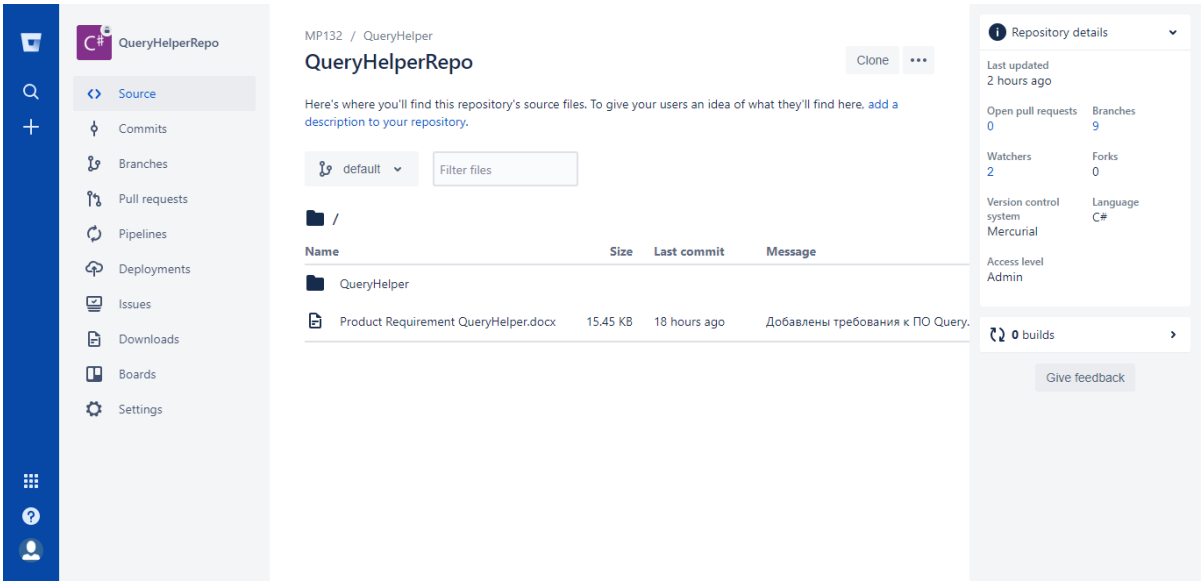

Рисунок 2.3 – Створений репозиторій

6 Граф розробки проекту

На рисунку 2.4 показаний граф розробки проекту.

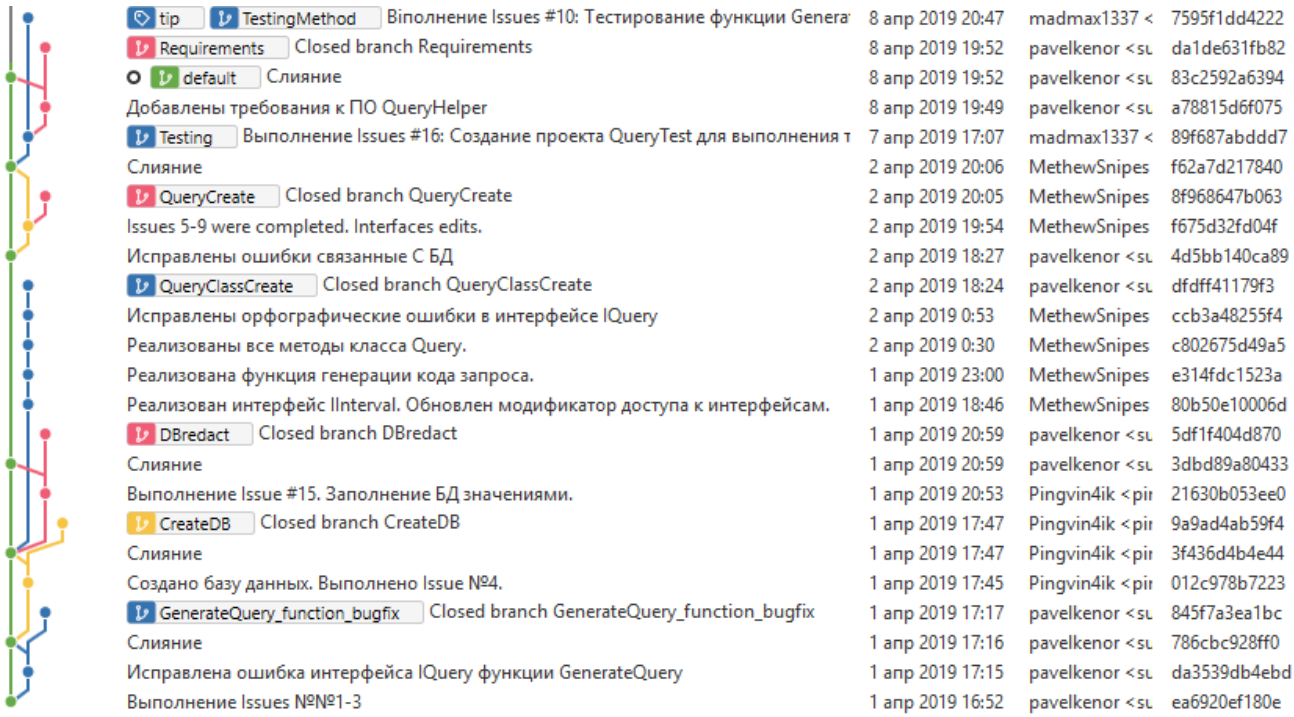

Рисунок 2.4 – Граф розробки проекту

Лабораторна робота № 3.

Хід роботи

1 Опис віх проекту

Гілка QueryCreate – основна гілка розробки логіки програмного забезпечення. У даній гілці було створено два класи: Interval і QueryGenerator.

Гілка GUILogic – в даній гілці необхідно розробити логічну частину графічного інтерфейсу.

2 Опис завдань, розбитих по віх проектів

Гілка QueryCreate має на увазі виконання завдань 5-9, а саме:

- створення проекту;
- розробка методу GenerateQuery;
- розробка методу AddAttribute;
- розробка методу AddNewValueToAtribute;
- реалізація інтерфейсу IInterval.

Гілка GUILogic має на увазі виконання завдання 19 – реалізація логіки GUI.

Всі завдання представлені в п. 3, на рисунку 3.1.

3 Issues

На рисунку 3.1 показані Issues проекту.

Issues (1–19 of 19)

Title	T	P	Status	Votes	Assignee	Milestone	Created	Updated
#19: Реализация логики GUI			NEW		Danil	Реализ...	32 seconds ago	32 seconds ago
#12: Тестирование функции AddNewValueToAtribute			RESOLVED		Максим Рудика	Тестир...	2019-04-01	a minute ago
#11: Тестирование функции AddNewAtribute			RESOLVED		Максим Рудика	Тестир...	2019-04-01	2 minutes ago
#10: Тестирование функции GenerateQuery			RESOLVED		Максим Рудика	Тестир...	2019-04-01	2 minutes ago
#5: Создание проекта для реализации интерфейсов			CLOSED		Danil	Создан...	2019-04-01	9 minutes ago
#6: Реализация функции GenerateQuery интерфейса IQuery			CLOSED		Danil	Создан...	2019-04-01	9 minutes ago
#7: Реализация функции AddNewAtribute интерфейса IQuery			CLOSED		Danil	Создан...	2019-04-01	9 minutes ago
#8: Реализация функции AddNewValueToAtribute интерфейса IQuery			CLOSED		Danil	Создан...	2019-04-01	9 minutes ago
#9: Реализация интерфейса IInterval			CLOSED		Danil	Создан...	2019-04-01	9 minutes ago
#18: Реализация выдачи диагностических сообщений			NEW		Danil	Добавл...	18 hours ago	18 hours ago
#17: Создание UI			NEW		Стас Скляр	Реализ...	18 hours ago	18 hours ago
#13: Создание проекта для реализации графического интерфейса пользователя			NEW		Стас Скляр	Создан...	2019-04-01	18 hours ago
#16: Создание проекта QueryTest для выполнения тестирования проекта			CLOSED		Максим Рудика	Создан...	2019-04-02	2 days ago
#14: Исправить ошибку интерфейса IQuery функции GenerateQuery			CLOSED		Pavel Suhak	Исправ...	2019-04-01	2019-04-02
#15: Заполнить БД ПО QueryHelper данными			CLOSED		Стас Скляр	Создан...	2019-04-01	2019-04-02
#4: Создать БД ПО QueryHelper			CLOSED		Стас Скляр	Создан...	2019-04-01	2019-04-02
#3: Добавить в проект Core интерфейсы для всего Solution			CLOSED		Pavel Suhak	Создан...	2019-04-01	2019-04-02
#2: Добавить в Solution проект Core			CLOSED		Pavel Suhak	Создан...	2019-04-01	2019-04-02
#1: Создание пустого Solution			CLOSED		Pavel Suhak	Создан...	2019-04-01	2019-04-02

Рисунок 3.1 – Issues проекту

4 Активність роботи над завданнями

На рисунку 3.2 показана активності роботи над завданнями.

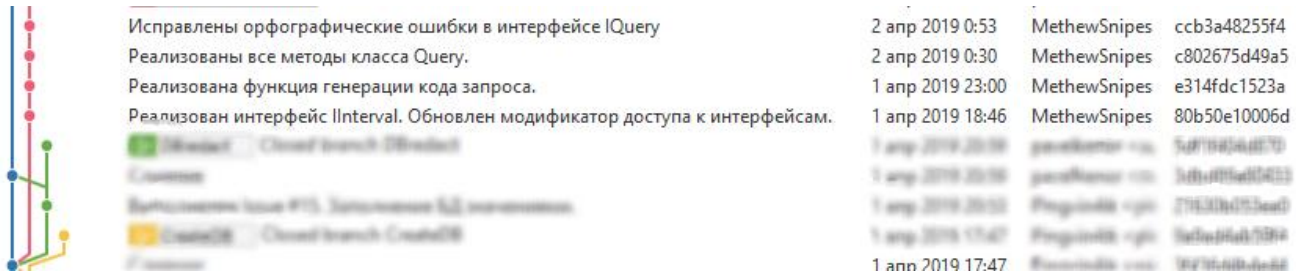

Рисунок 3.2 – Активність роботи над завданнями

Лабораторна робота № 4.

Хід роботи

1 Вибір паттерна впровадження залежності

Для впровадження залежності був обраний патерн Constructor Injection, тобто впровадження залежності через конструктор.

Суть паттерна зводиться до того, що всі залежності, необхідні деякого класу, передаються йому в якості параметрів конструктора, представлених у вигляді інтерфейсів або абстрактних класів.

Constructor Injection є базовим патерном впровадження залежностей, і він інтенсивно застосовується більшістю програмістів, навіть якщо вони про це не замислюються. Однією з головних цілей більшості «стандартних» патернів проектування є отримання слабозв'язаного дизайну, тому не дивно, що більшість з них в тому або іншому вигляді використовують впровадження залежностей.

Головною перевагою впровадження залежностей через конструктор є чіткий поділ відповідальностей і явність інтерфейсу. Основна проблема використання глобальних об'єктів і Сінглтон полягає в тому, що, читаючи «заголовок класу» (його публічний інтерфейс) неможливо визначити складність його поведінки, і для цього потрібно проаналізувати реалізацію цього класу з усіма його методами. У разі, коли всі зовнішні залежності передаються через конструктор, складність класу стає більш очевидною.

Впровадження залежностей за допомогою конструктора для даного проекту показано нижче:

```
 public interface IDatabase
 {
OleDbConnection Dbc {get; set; }
 void Open ();
 void Close ();
 }
 public class Database: IDatabase
 {
 private OleDbConnection dbc;
public OleDbConnection Dbc {get => dbc; set => dbc = value; }
```

```
public Database (string DBpath)
{
Dbc = new OleDbConnection ( "data source =" + DBpath + "; provider = microsoft.jet.oledb.4.0;");
}
        ...
}
public class QueryGenerator: Core.IQuery
{
private IDatabase db;
public QueryGenerator (IDatabase db)
{
      ...
}
        ...
}
```

2 Опис завдань рефакторінга, поставлених тімліда на поточну ітерацію

завдання:

- виділити клас Database з поточного класу QueryGenerator;
- створити залежність між класами QueryGenerator і Database;
- використовувати патерн Dependency Injection.

3 Активність роботи над завданнями

На рисунку 4.1 показані активності роботи над завданнями.

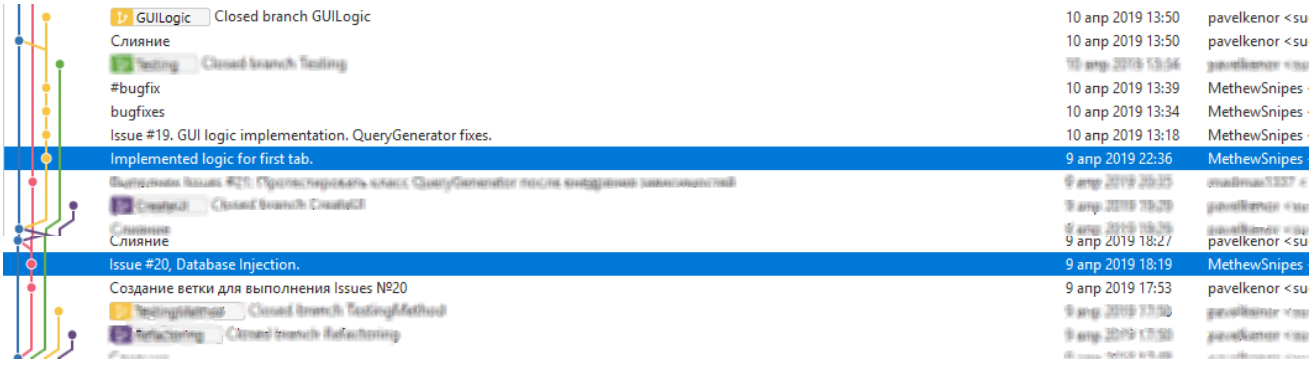

Рисунок 4.1 – Активність роботи над завданнями

Лабораторна робота № 5.

Хід роботи

1 Лістинг коду, що описує ініціалізацію макетів замінних класів в тестах-пустушках

```
string tableName = "Table";
int amQueries = 1;
List <string> attributeList = new List <string> ();

attributeList.Add ( "Прізвище");
attributeList.Add ( "Ім'я");
attributeList.Add ( "По батькові");

List <IInterval> intervalList = new List <IInterval> ();

intervalList.Add (new Interval (1, 5));
intervalList.Add (new Interval (1, 4));
intervalList.Add (new Interval (1, 6));

var mock = new Mock <IQuery> ();

 mock.Setup (repo => repo.GenerateQuery (tableName, attributeList, intervalList, amQueries)).

Returns ( "INSERT INTO Table (Прізвище, Ім'я, По батькові) \ n VALUES ( 'Рудик', 'Інна', 'Яновна') \ n ");

 QueryGeneratorDecorator decorator = new QueryGeneratorDecorator (mock.Object);
```

2 Коментарі до кожного методу свого тестованого класу, в яких тестових наборах даних потребує даний метод

```
decorator.GenerateQuery (/ *
 // 1) tableName
 // 2) attributeList
 // 3) intervalList
 // 4) amQueries * /);
```

3 Активність роботи над завданнями

На рисунку 5.1 показані активності роботи над завданнями.

Рисунок 5.1 – Активність роботи команди над завданнями проекту

Лабораторна робота № 6.

Хід роботи

1. Скріншоти послідовності процесу інсталяції

На рисунках 6.1 – 6.6 показаний процес роботи установника.

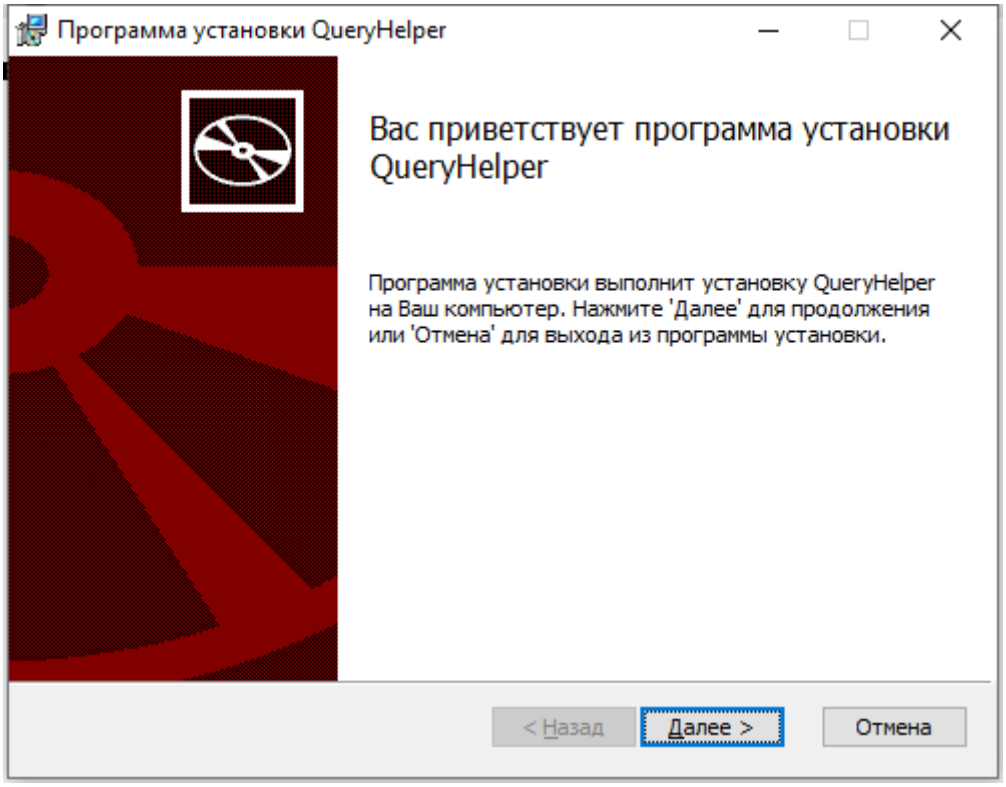

Рисунок 6.1 – Установка. Етап 1

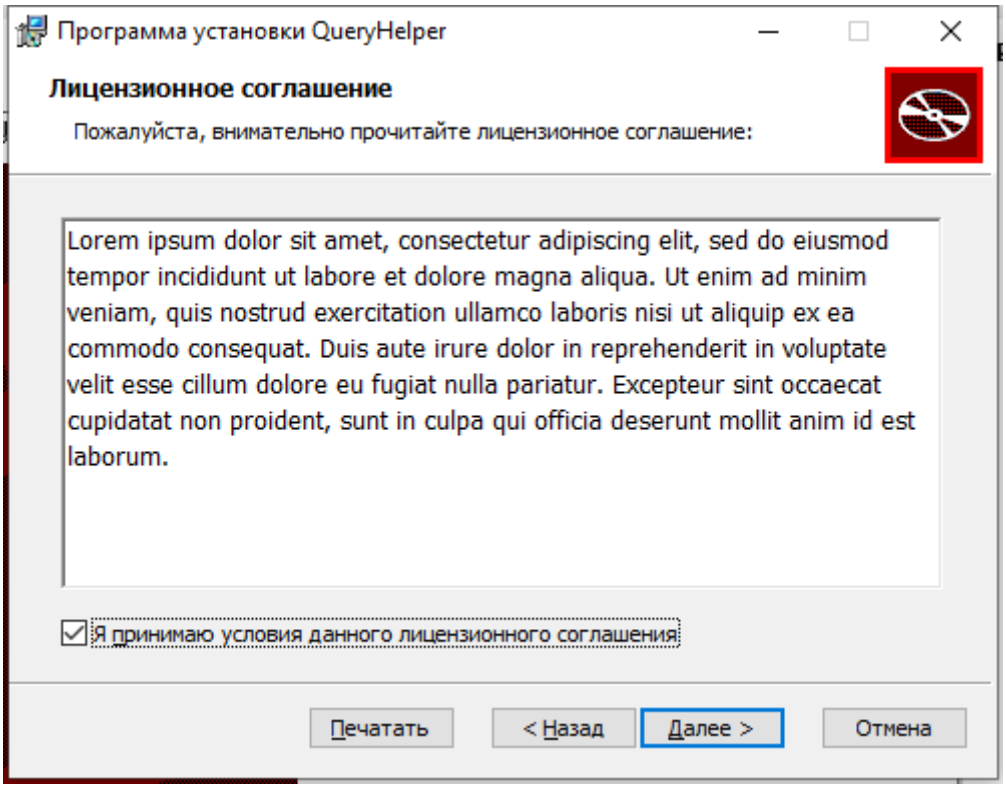

Рисунок 6.2 – Установка. Етап 2

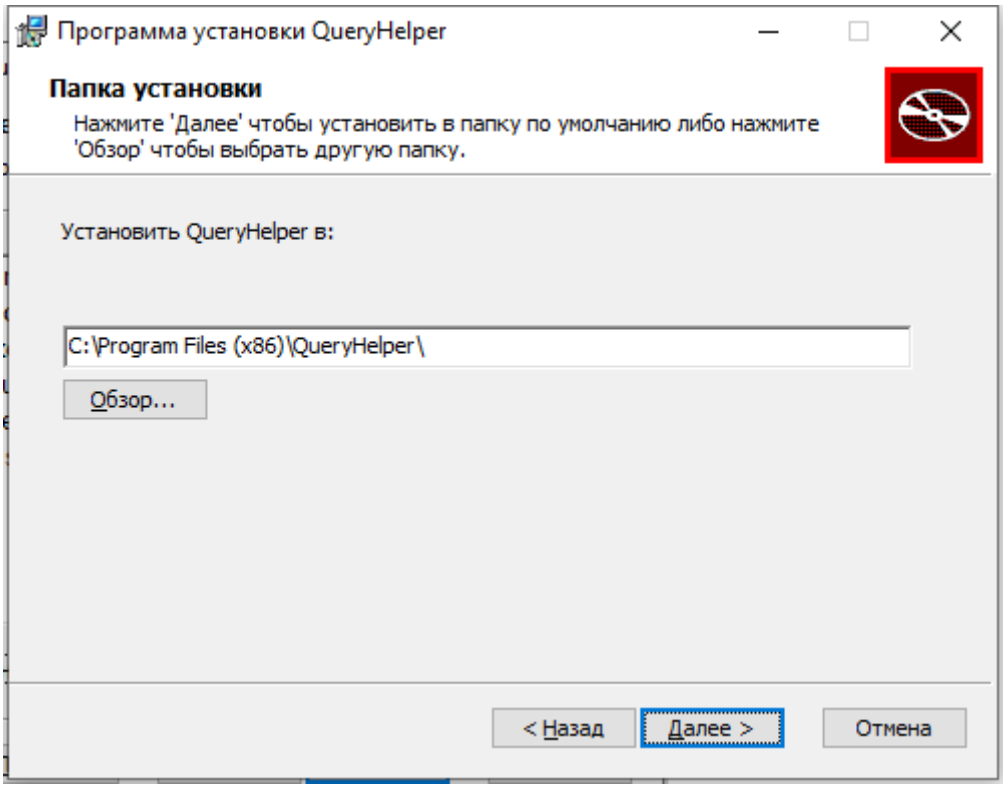

Рисунок 6.3 – Установка. Етап 3

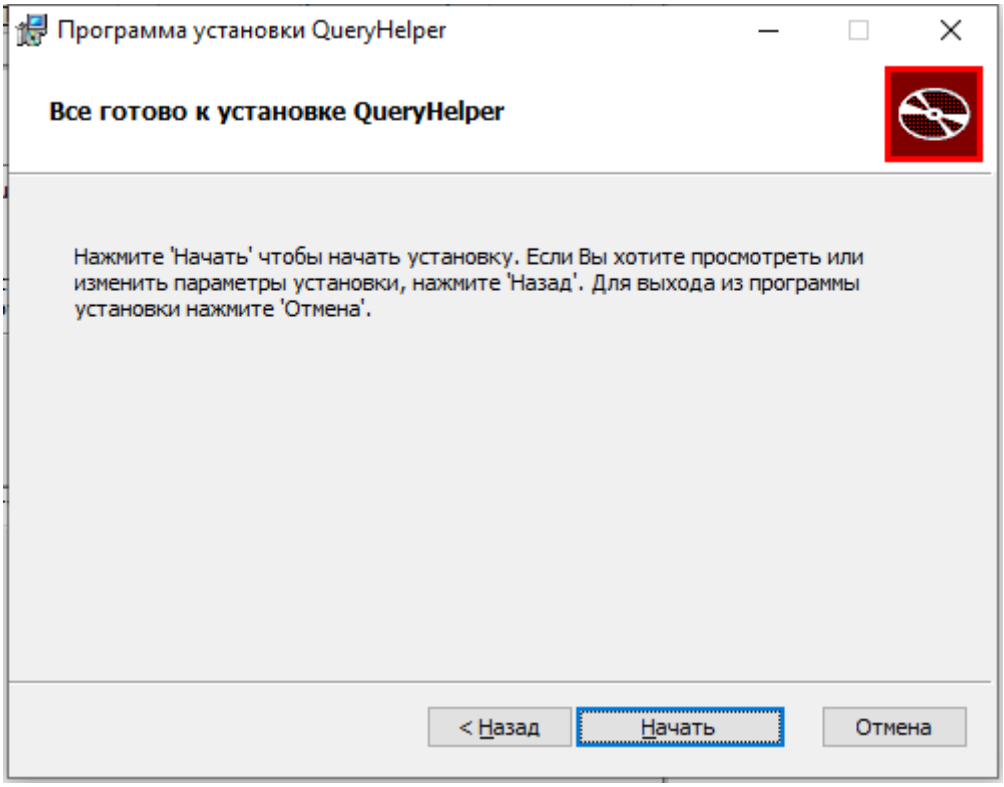

Рисунок 6.4 – Установка. Етап 4

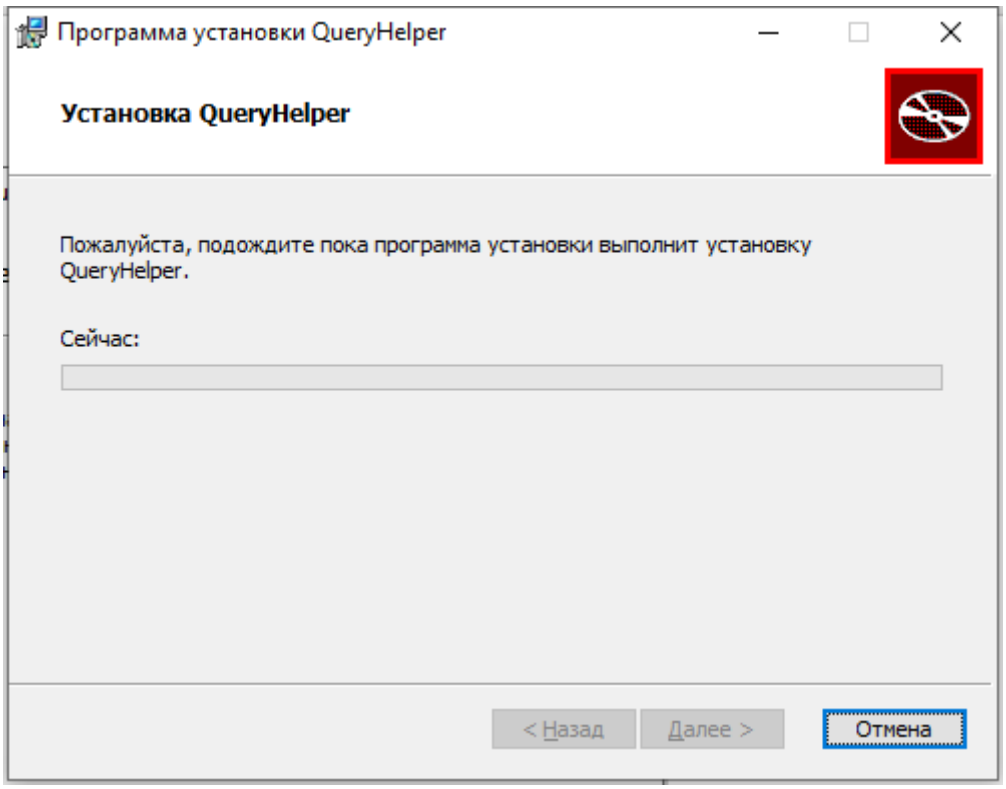

Рисунок 6.5 – Установка. Етап 5

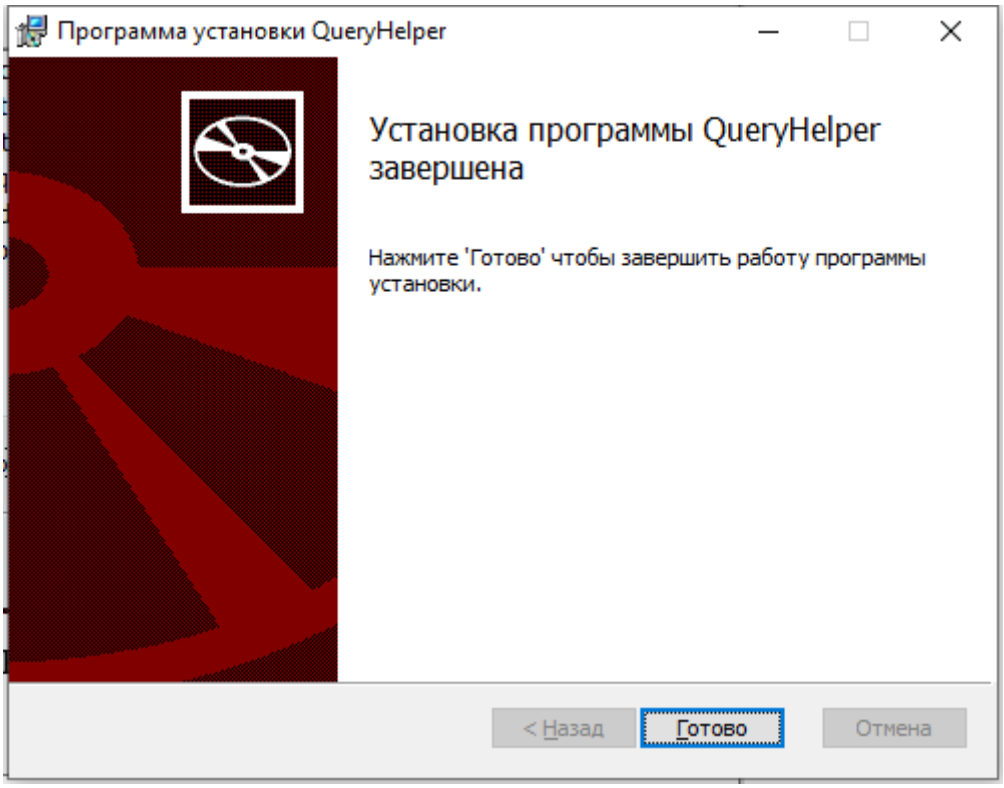

Рисунок 6.6 – Установка. Етап 6

Скріншот списку встановлених програм з нашою встановленою програмою наведено на рисунку 6.7.

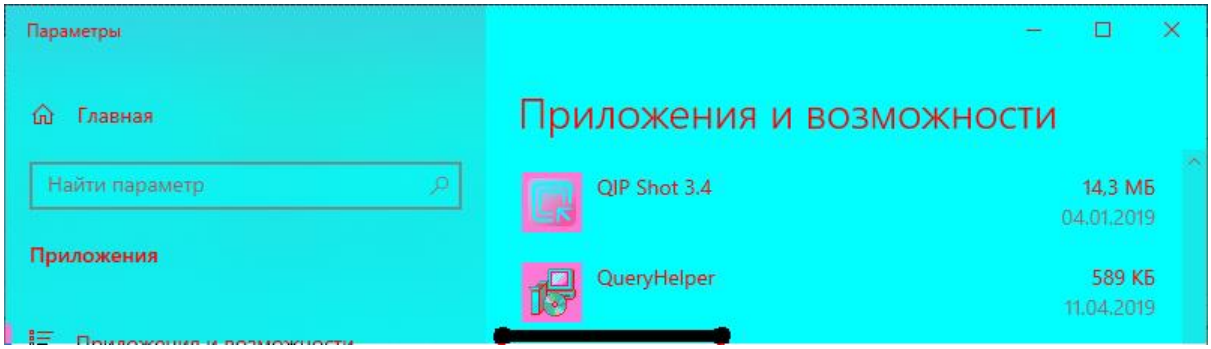

Рисунок 6.7 – Результат установки

Лістинг програмного коду

Product.wxs

```
<? Xml version = "1.0" encoding = "UTF-8"?>

<Wix xmlns = "http://schemas.microsoft.com/wix/2006/wi">
 <? Define ProductName = "QueryHelper"?>
 <? Define ProductVersion = "1.0.0.0"?>
 <? Define ProductCode = "b7bc7c6f-9a4e-4973-be84-eca8e3427c97"?>
 <? Define UpgradeCode = "06a81104-1e30-463d-87e1-e8a79b4c682a"?>
 <? Define Manufacturer = "MP132"?>

 <Product Id = "$ (var.ProductCode)" Name = "$ (var.ProductName)" Language = "1049" Version = "$ (var.ProductVersion)" Manufacturer = "$ (var.Manufacturer)" UpgradeCode = " $ (var.UpgradeCode) ">
 <Package InstallerVersion = "200" Compressed = "yes" />

 <Media Id = "1" Cabinet = "media1.cab" EmbedCab = "yes" />

 <Directory Id = "TARGETDIR" Name = "SourceDir">
 <Directory Id = "ProgramFilesFolder">
 <Directory Id = "INSTALLLOCATION" Name = "$ (var.ProductName)">
 <Component Id = "ProductComponent" Guid = "b11556a2-e066-4393-af5c-9c9210187eb2">
 <File Id = 'QueryHelper' DiskId = '1' Source = 'C: \ Users \ Danil \ Documents \ QueryHelper \ QueryHelper \ GUI \ bin \ Debug \ QueryHelper.exe' />
 </ Component>
 <Component Id = "ProductComponent1" Guid = "b11556a2-e066-4393-af5c-9c9210187eb3">
 <File Id = 'QueryHelper1' DiskId = '1' Source = 'C: \ Users \ Danil \ Documents \ QueryHelper \ QueryHelper \ GUI \ bin \ Debug \ QueryHelper.mdb' />
 </ Component>
 <Component Id = "ProductComponent2" Guid = "b11556a2-e066-4393-af5c-9c9210187eb4">
 <File Id = 'QueryHelper2' DiskId = '1' Source = 'C: \ Users \ Danil \ Documents \ QueryHelper \ QueryHelper \ GUI \ bin \ Debug \ IQueryHelper.dll' />
 </ Component>
 <Component Id = "ProductComponent3" Guid = "b11556a2-e066-4393-af5c-9c9210187eb5">
 <File Id = 'QueryHelper3' DiskId = '1' Source = 'C: \ Users \ Danil \ Documents \ QueryHelper \ QueryHelper \ GUI \ bin \ Debug \ Query.dll' />
 </ Component>
```

```
 </ Directory>
 </ Directory>
 <Directory Id = "ProgramMenuFolder">
 <Directory Id = "ApplicationProgramsFolder" Name = "$ (var.ProductName)">
 <Component Id = "ApplicationShortcutQueryHelper" Guid = "4CEBD68F-E933-47f9-B02C-A4FC69FDB551">
 <Shortcut Id = "ShortcutQueryHelper"
 Name = "QueryHelper"
 Description = "$ (var.ProductName)"
 Target = "[INSTALLLOCATION] QueryHelper.exe"
 WorkingDirectory = "INSTALLLOCATION" />
 <RemoveFolder Id = "ApplicationProgramsFolder" On = "uninstall" />
 <RegistryValue Root = "HKCU" Key = "Software \ $ (var.Manufacturer) \ $ (var.ProductName)" Name = "installed" Type = "integer" Value = "1" KeyPath = "yes" />
 </ Component>
 </ Directory>
 </ Directory>
 </ Directory>

 <Feature Id = "ProductFeature" Title = "QueryHelper" Level = "1">
 <ComponentRef Id = "ProductComponent" />
 <ComponentRef Id = "ProductComponent1" />
 <ComponentRef Id = "ProductComponent2" />
 <ComponentRef Id = "ProductComponent3" />
 <ComponentRef Id = "ApplicationShortcutQueryHelper" />
 </ Feature>

 <Property Id = "WIXUI_INSTALLDIR" Value = "INSTALLLOCATION"> </ Property>
 <UIRef Id = "WixUI_InstallDir" />

 </ Product>
</ Wix>
<! -
<? Xml version = "1.0" encoding = "UTF-8"?>
<Wix xmlns = "http://schemas.microsoft.com/wix/2006/wi">
 <? Define ProductName = "QueryHelper"?>
 <? Define ProductVersion = "0.0.0.1"?>
 <? Define ProductCode = "b7bc7c6f-9a4e-4973-be84-eca8e3427c97"?>
 <? Define UpgradeCode = "06a81104-1e30-463d-87e1-e8a79b4c682a"?>
 <? Define Manufacturer = "MP132"?>
 <Product Id = "$ (var.ProductCode)" Name = "$ (var.ProductName)" Language = "1049" Version = "$ (var.ProductVersion)" Manufacturer = "$ (var.Manufacturer)" UpgradeCode = " $ (var.UpgradeCode) ">
		<Package InstallerVersion = "200" Compressed = "yes" InstallScope = "perMachine" />

		<MajorUpgrade DowngradeErrorMessage = "A newer version of [ProductName] is already installed." />
		<MediaTemplate />

 <Feature Id = "ProductFeature" Title = "$ (var.ProductName)" Level = "1">
 <ComponentRef Id = "ProductComponent" />
 </ Feature>

 <Property Id = "WIXUI_INSTALLDIR" Value = "INSTALLLOCATION"> </ Property>
 <WixVariable Id = "WixUILicenseRtf" Overridable = "yes" Value = "License.rtf" />
 <UIRef Id = "WixUI_InstallDir" />
```

```
</ Product>

        <Fragment>
            <Directory Id = "TARGETDIR" Name = "SourceDir">
                <Directory Id = "ProgramFilesFolder">
                    <Directory Id = "INSTALLFOLDER" Name = "SetupProject1" />
                </ Directory>
<Directory Id = "ProgramMenuFolder">
<Directory Id = "ApplicationProgramsFolder" Name = "$ (var.ProductName)">
<Component Id = "ApplicationShortcutQueryHelper" Guid = "4CEBD68F-E933-47f9-B02C-A4FC69FDB551">
<Shortcut Id = "ShortcutQueryHelper"
Name = "QueryHelper"
Description = "$ (var.ProductName)"
Target = "[INSTALLLOCATION] QueryHelper.exe"
WorkingDirectory = "INSTALLLOCATION" />
<RemoveFolder Id = "ApplicationProgramsFolder" On = "uninstall" />
<RegistryValue Root = "HKCU" Key = "Software \ $ (var.Manufacturer) \ $ (var.ProductName)" Name = "installed" Type = "integer" Value = "1" KeyPath = "yes" />
</ Component>
</ Directory>
</ Directory>
</ Directory>
        </ Fragment>

        <Fragment>
            <ComponentGroup Id = "ProductComponents" Directory = "INSTALLFOLDER">
<Component Id = "ProductComponent" Guid = "b11556a2-e066-4393-af5c-9c9210187eb3">
<File Id = 'QueryHelper' DiskId = '1' Source = 'C: \ Users \ Danil \ Documents \ QueryHelper \ QueryHelper \ GUI \ bin \ Debug \ QueryHelper.exe' />
</ Component>
<Component Id = "ProductComponentDB" Guid = "b11556a2-e066-4393-af5c-9c9210187eb2">
<File Id = 'QueryHelpreDB' DiskId = '1' Source = 'C: \ Users \ Danil \ Documents \ QueryHelper \ QueryHelper \ GUI \ bin \ Debug \ QueryHelperDB.mdb' />
</ Component>
            </ ComponentGroup>
        </ Fragment>
</ Wix> ->
```

ДОДАТОК А.
Лістинг шаблону установочного дистрибутива

```
<?xml version="1.0" encoding="UTF-8"?>
<Wix xmlns="http://schemas.microsoft.com/wix/2006/wi">
<?define ProductName="Laba6" ?>
<?define ProductVersion="1.0.0.0" ?>
<?define ProductCode="b7bc7c6f-9a4e-4973-be84-eca8e3427c97"?>
<?define UpgradeCode="06a81104-1e30-463d-87e1-e8a79b4c682a"?>
<?define Manufacturer="Student KhAI"?>
<Product Id="$(var.ProductCode)" Name="$(var.ProductName)"
Language="1049" Version="$(var.ProductVersion)"
Manufacturer="$(var.Manufacturer)" UpgradeCode="$(var.UpgradeCode)">
<Package InstallerVersion="200" Compressed="yes" />
<Media Id="1" Cabinet="media1.cab" EmbedCab="yes" />
<Directory Id="TARGETDIR" Name="SourceDir">
<Directory Id="ProgramFilesFolder">
<Directory Id="INSTALLLOCATION" Name="$(var.ProductName)">
<Component Id="ProductComponent" Guid="b11556a2-e066-4393-af5c-
9c9210187eb2">
<File Id='Calc' DiskId='1' Source='C:\WINDOWS\system32\calc.exe'/>
</Component>
</Directory>
</Directory>
<Directory Id="ProgramMenuFolder">
<Directory Id="ApplicationProgramsFolder" Name="$(var.ProductName)">
<Component Id="ApplicationShortcutCalc" Guid="4CEBD68F-E933-47f9-B02C-
A4FC69FDB551">
<Shortcut Id="ShortcutCalc"
Name="Calc"
Description="$(var.ProductName)"
Target="[INSTALLLOCATION]Calc.exe"
WorkingDirectory="INSTALLLOCATION"/>
<Shortcut Id="ShortcutUninstall"
```

```
    Name="Uninstall"
    Description="Uninstall"
    Target="[SystemFolder]\msiexec"
    Arguments="/x [ProductCode]"
    WorkingDirectory="INSTALLDIR" />
    <RemoveFolder Id="ApplicationProgramsFolder" On="uninstall"/>
    <RegistryValue Root="HKCU"
Key="Software\$(var.Manufacturer)\$(var.ProductName)" Name="installed"
Type="integer" Value="1" KeyPath="yes"/>
    </Component>
    </Directory>
    </Directory>
    </Directory>
    <Feature Id="ProductFeature" Title="SetupProject1" Level="1">
    <ComponentRef Id="ProductComponent" />
    <ComponentRef Id="ApplicationShortcutCalc" />
    </Feature>
    <Property Id="WIXUI_INSTALLDIR" Value="INSTALLLOCATION" ></Property>
    <WixVariable Id="WixUILicenseRtf" Overridable="yes"
Value="License.rtf"/>
    <UIRef Id="WixUI_InstallDir"/>
    </Product>
    </Wix>
```

ДОДАТОК Б.

Теоретичний матеріал до виконання розрахунково-графічної роботи

Безперервна інтеграція (CI, англ. Continuous Integration) – практика розробки програмного забезпечення, яка полягає у виконанні частих автоматизованих збірок проекту для якнайшвидшого виявлення потенційних дефектів і рішення інтеграційних проблем. У звичайному проекті, де над різними частинами системи розробники трудяться незалежно, стадія інтеграції є заключною. Вона може непередбачувано затримати закінчення робіт. Перехід до безперервної інтеграції дозволяє знизити трудомісткість інтеграції і зробити її більш передбачуваною за рахунок найбільш раннього виявлення та усунення помилок і протиріч, але основною перевагою є скорочення вартості виправлення дефекту, за рахунок раннього його виявлення.

До переваг безперервної інтеграції належать такі моменти:

– проблеми інтеграції виявляються і виправляються швидко, що виявляється дешевше;

– негайний прогін модульних тестів для останніх редагувань;

– постійна наявність поточної стабільної версії разом з продуктами збірок – для тестування, демонстрації, і т. п.

– негайний ефект від неповного або непрацюючого коду привчає розробників до роботи в итеративном режимі з більш коротким циклом.

При цьому, практика не позбавлена недоліків, зокрема:

– значні витрати на підтримку роботи безперервної інтеграції;

– необхідність в додаткових обчислювальних ресурсах під потреби безперервної інтеграції.

Приклад № 1. Використовуючи інструментальні засоби, зазначені викладачем (наприклад, сервер збірки TeamCity), продемонструвати його настройку. Система контролю версій – Mercurial, веб-хостинг – BitBucket.

Розгорнути середу для збірки, тестування додатків з використанням сервера безперервної інтеграції.

Встановлення TeamCity.

Для виконання роботи будемо використовувати TeamCity. Розміщуватися CI буде на окремій машині з операційною системою Windows 10.

При установці будемо використовувати графічний інтерфейс.

На рисунках Б.1 – Б.8 наведено послідовність встановлення сервера збірки.

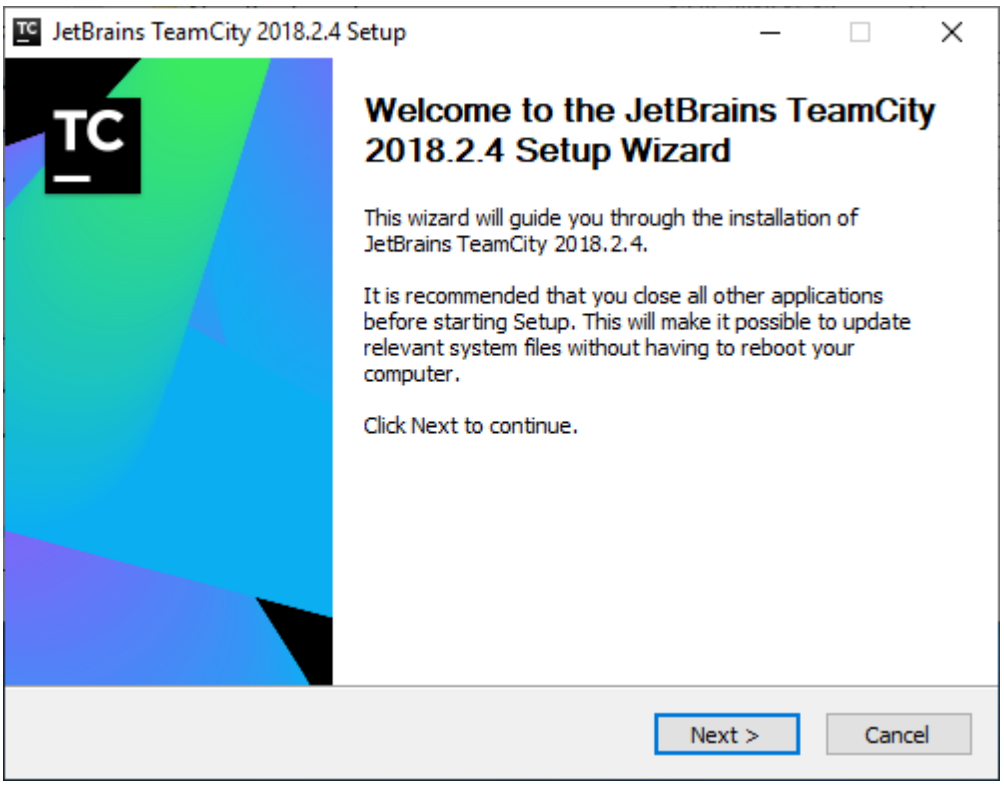

Рисунок Б.1 – Початкове вікно установки

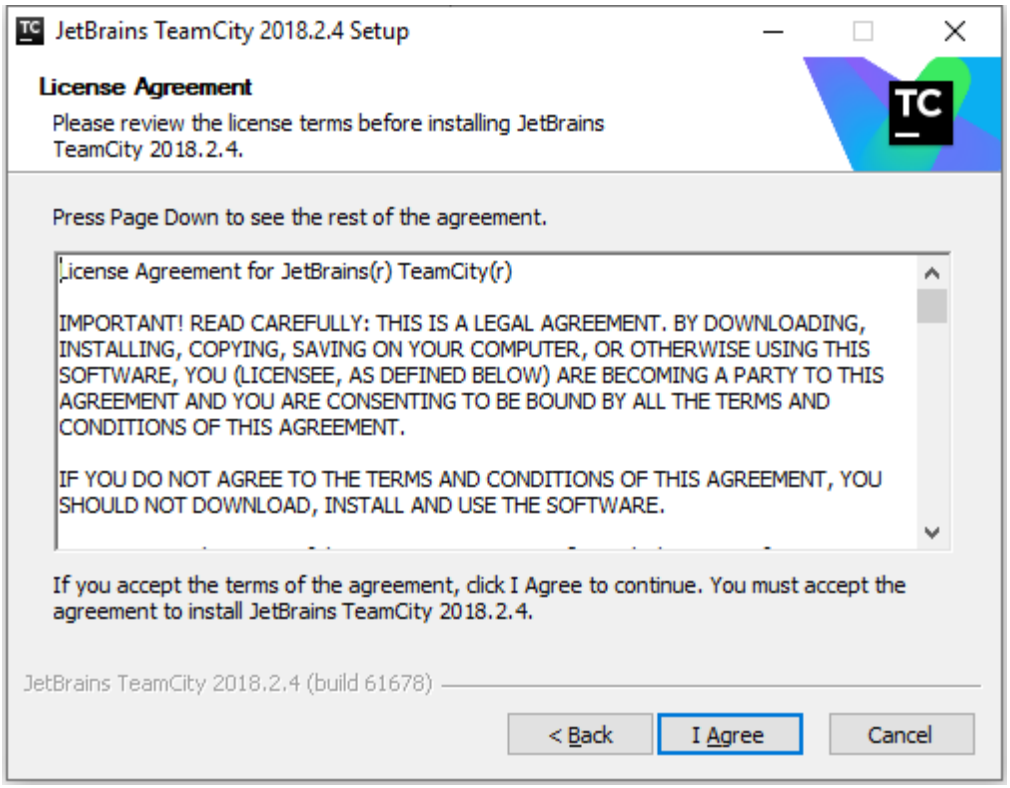

Рисунок Б.2 – Прийняття ліцензійної угоди

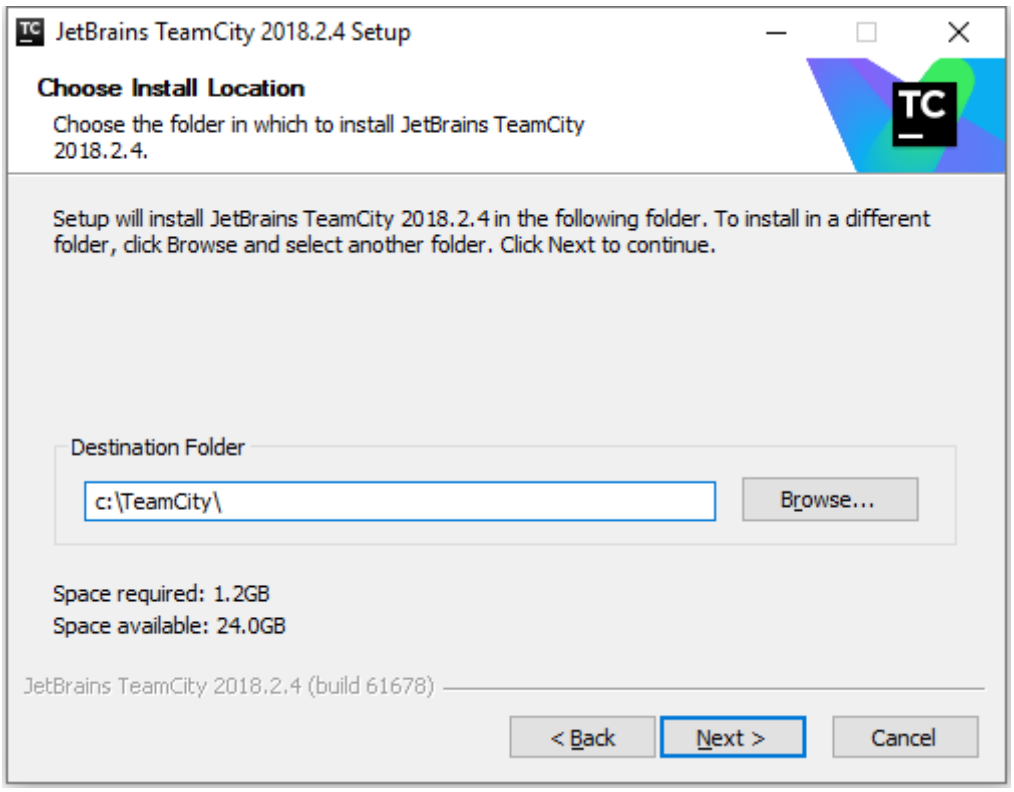

Рисунок Б.3 – Вибір шляху для установки TeamCity

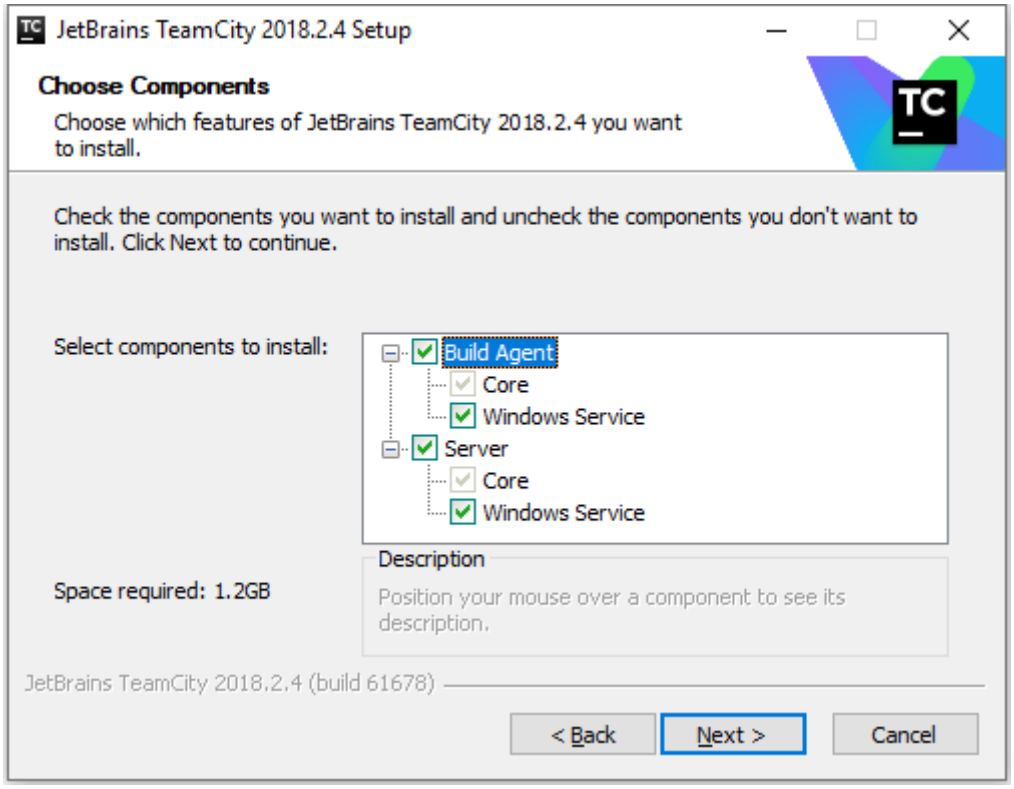

Рисунок Б.4 – Вибір компонентів для установки TeamCity

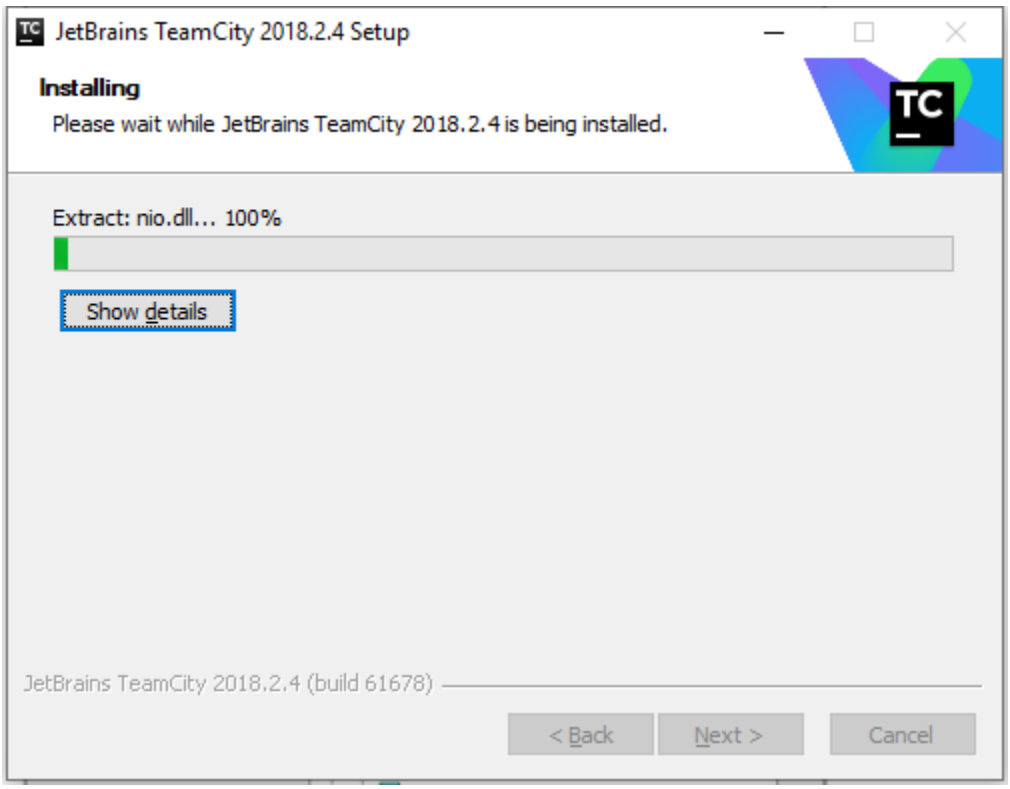

Рисунок Б.5 – Процес установки TeamCity

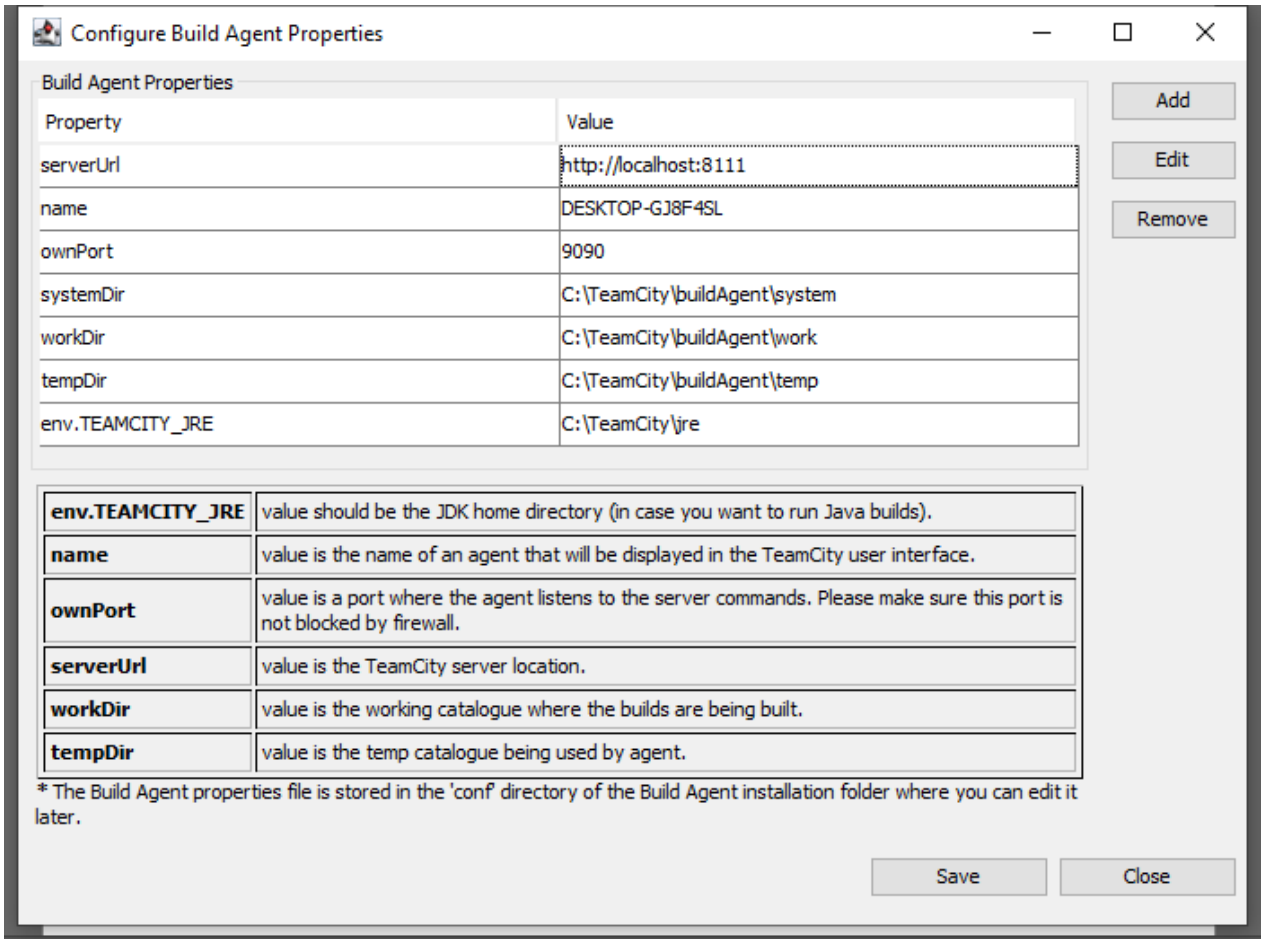

Рисунок Б.6 – Установча настройка Build Agent

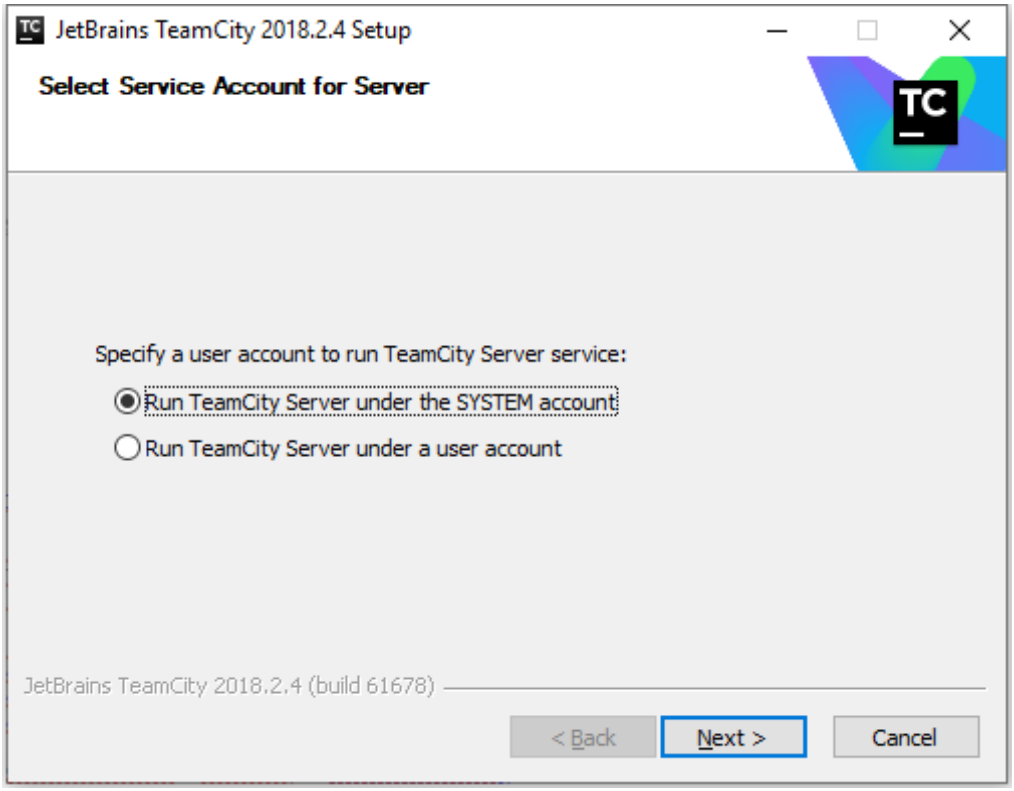

Рисунок Б.7 – Вибір рівня доступу

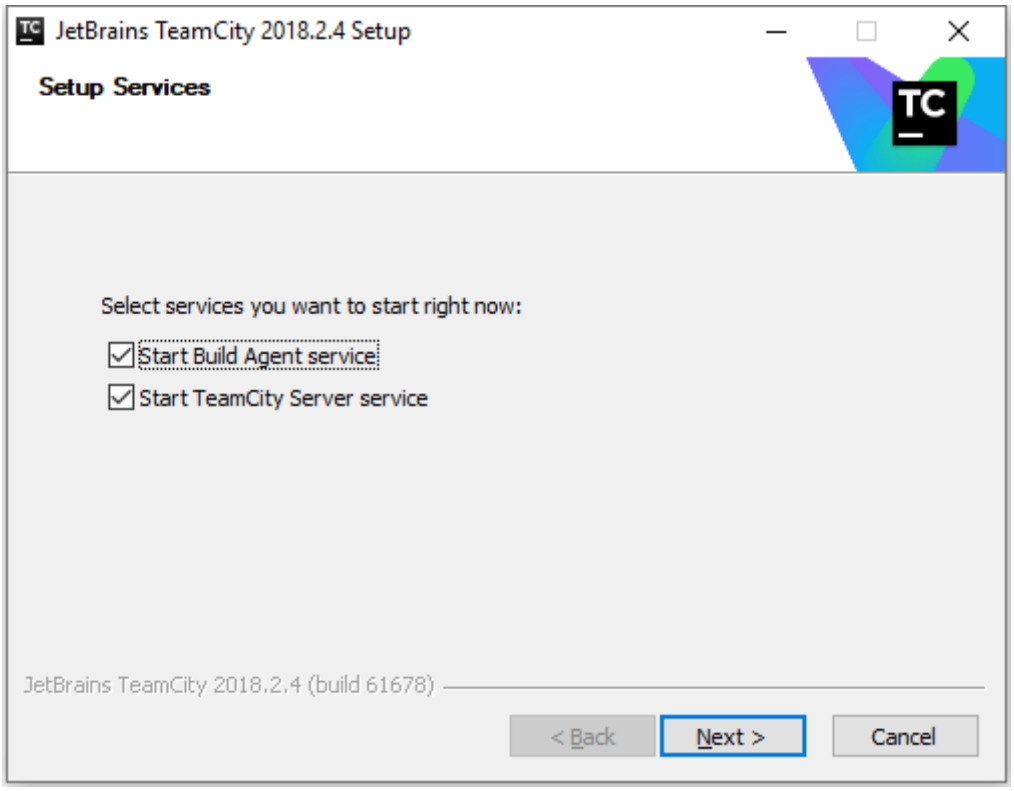

Рисунок Б.8 – Завершення установки TeamCity

Запуск TeamCity. Після установки запускаємо сервер TeamCity і раніше налаштований Build Agent (див. рисунки Б.9 – Б.12).

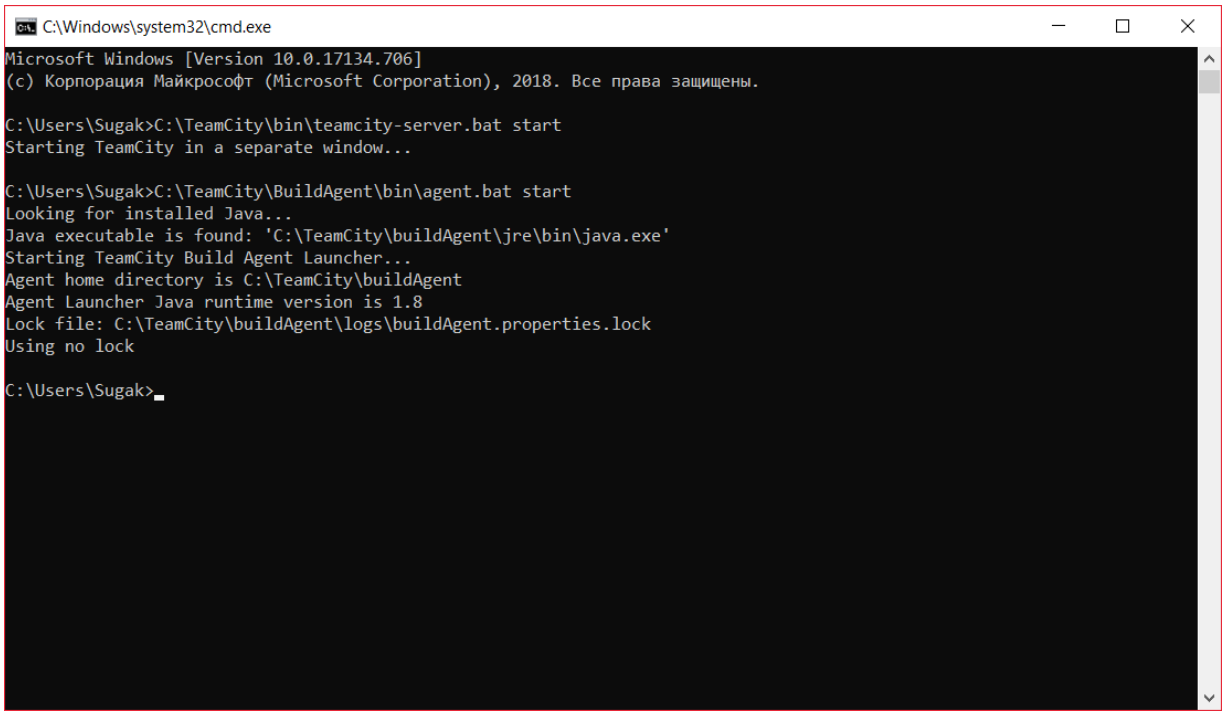

Рисунок Б.9 – Запуск сервера і агента за допомогою командного рядка

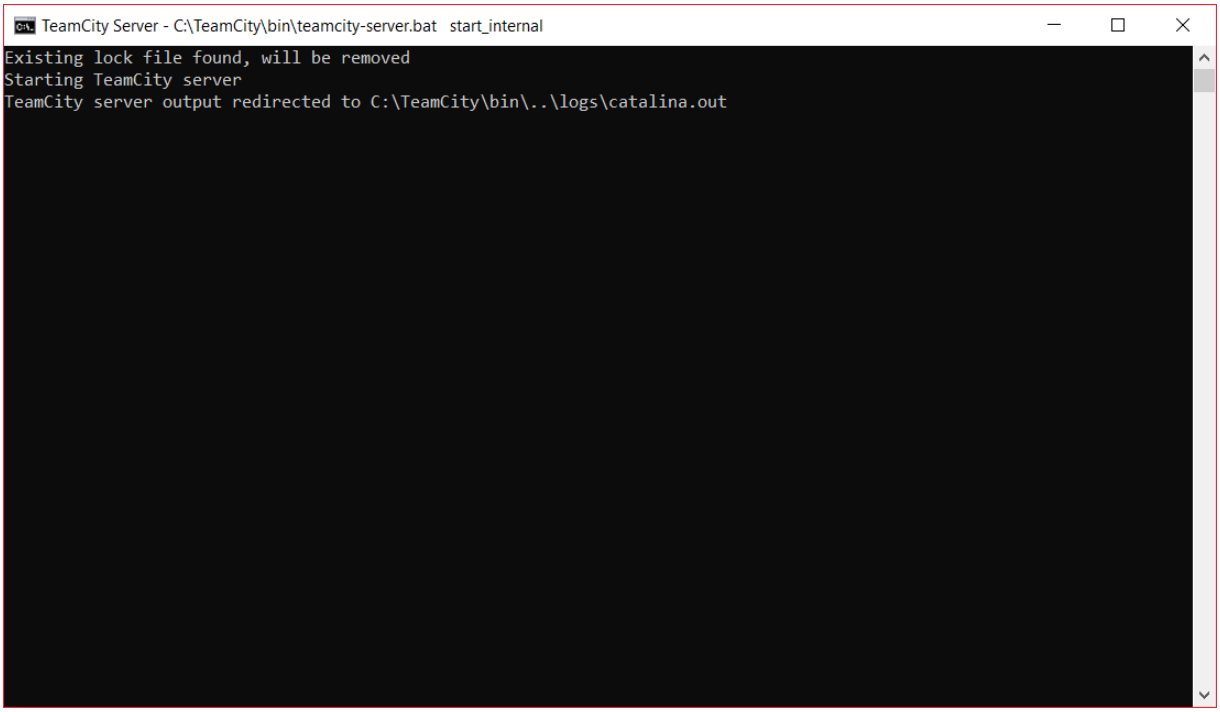

Рисунок Б.10 – Запущений сервер

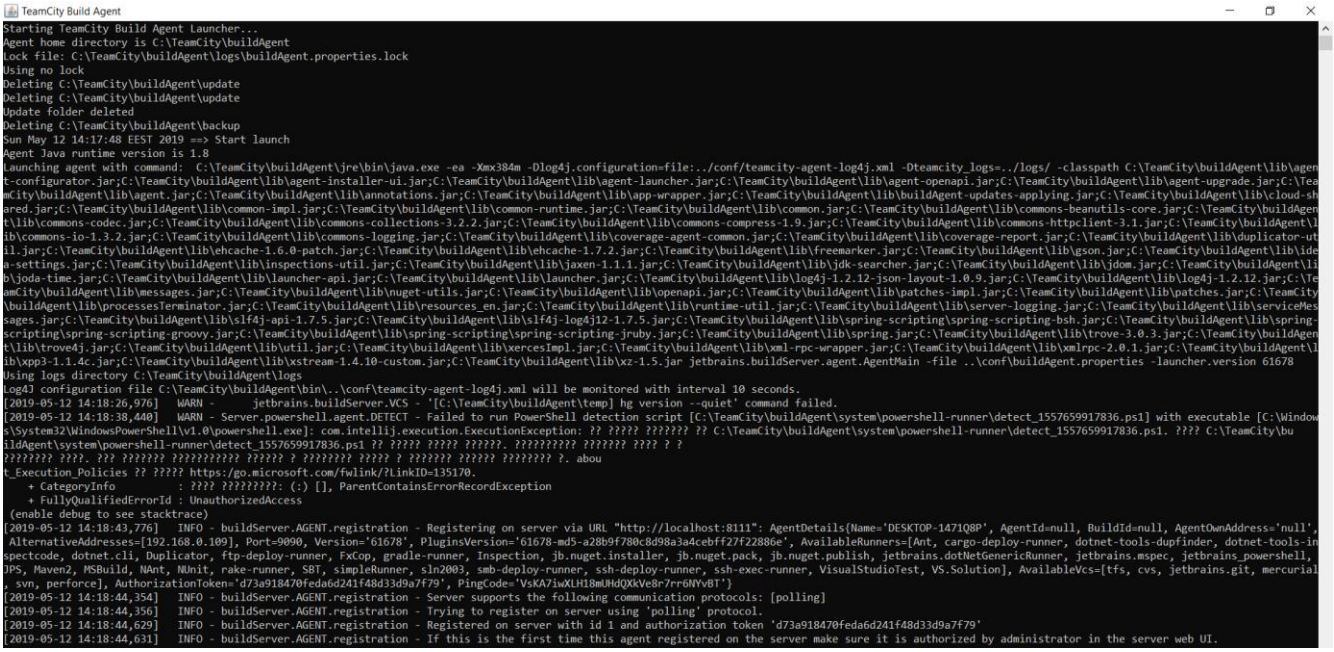

Рисунок Б.11 – Запущений агент

Налаштування TeamCity. Після запуску сервера TeamCity і агента виконуємо налаштування. (див. рисунки Б.12 – Б.24).

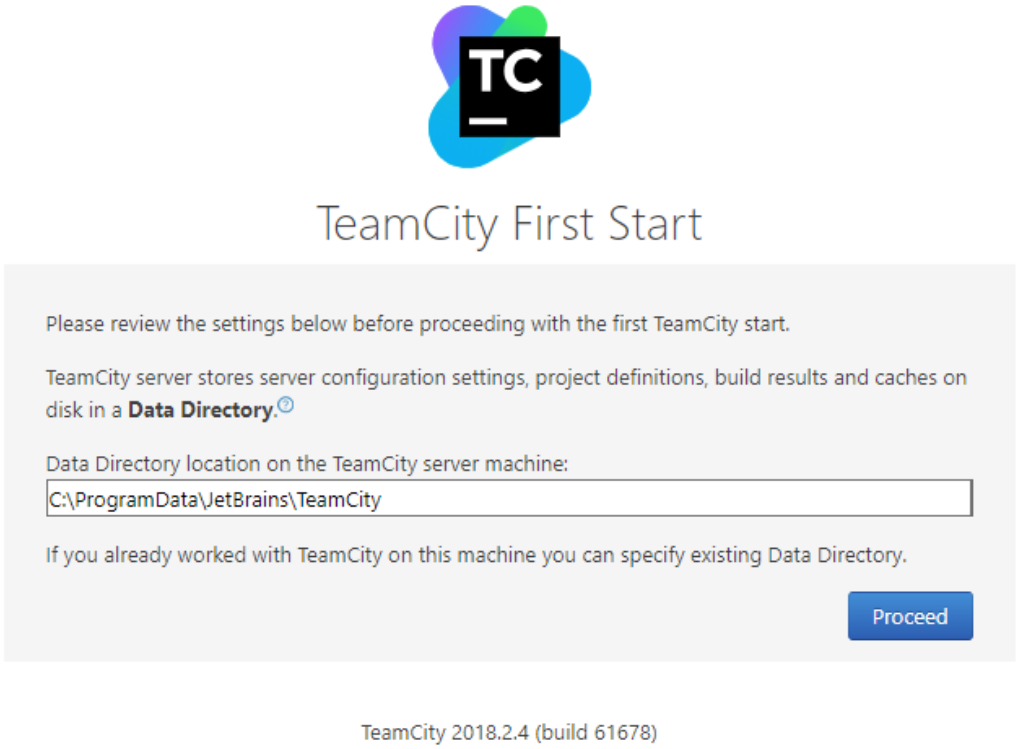

Рисунок Б.12 – Вибір директорії
для зберігання конфігурації сервера TeamCity

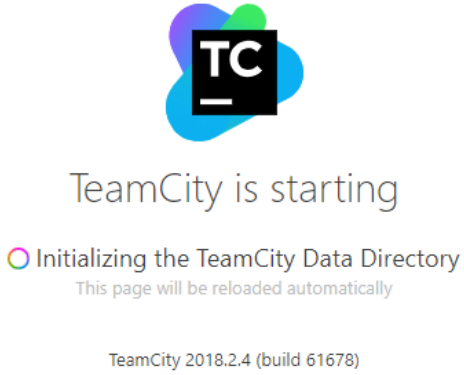

Рисунок Б.13 – Процес запуску сервера

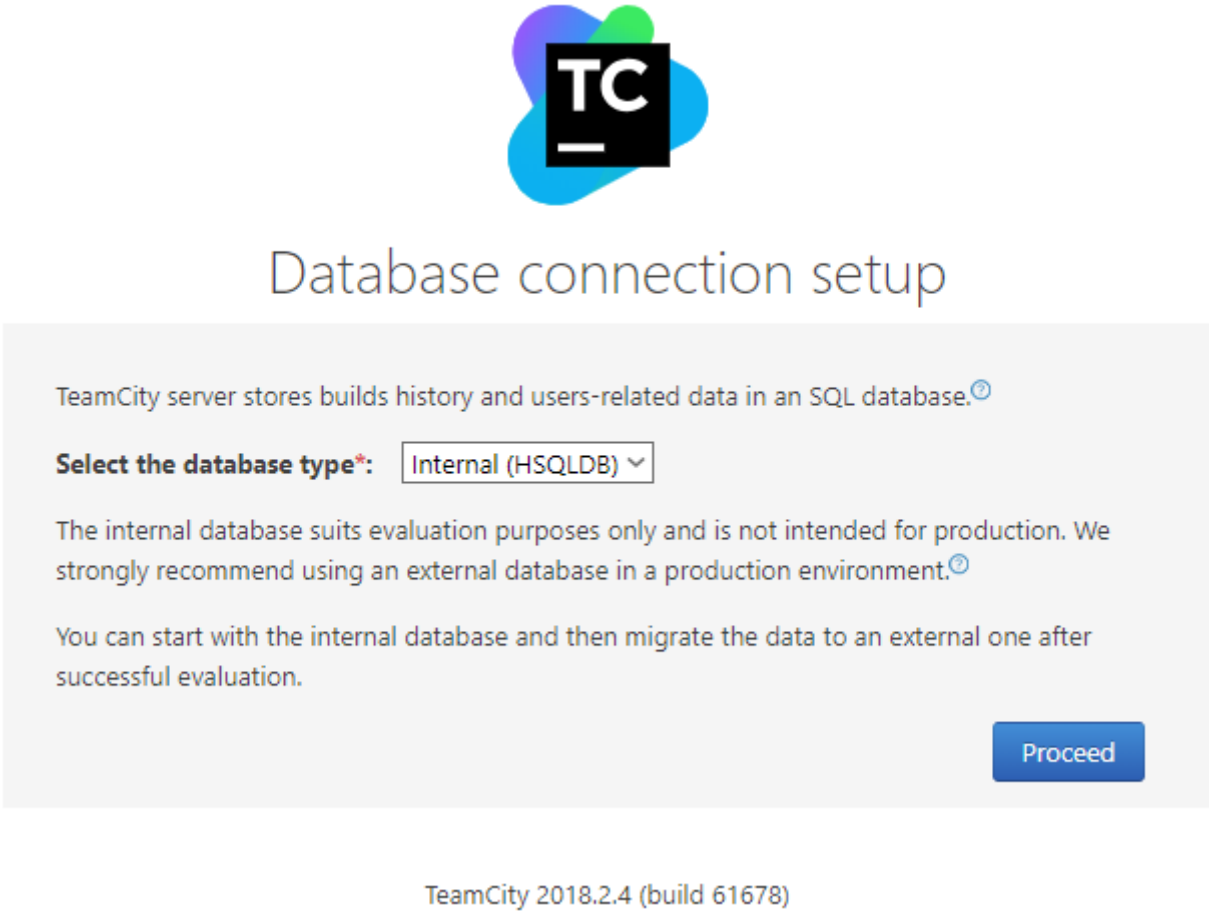

Рисунок Б.14 – Вибір типу бази даних

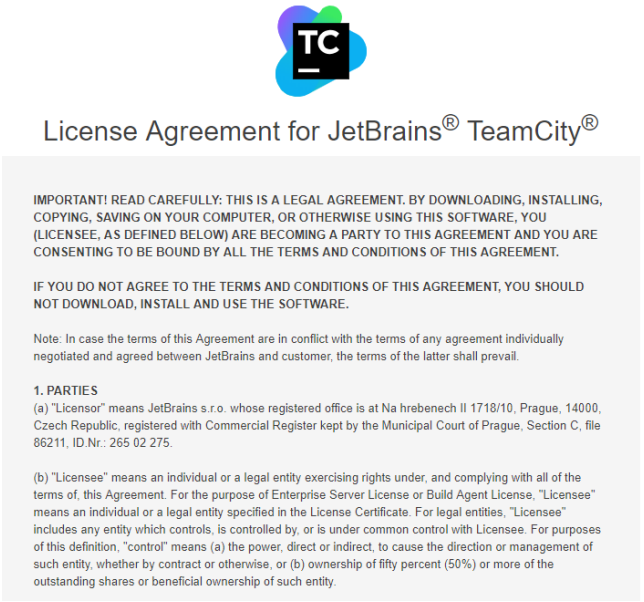

Рисунок Б.15 – Прийняття ліцензійної угоди

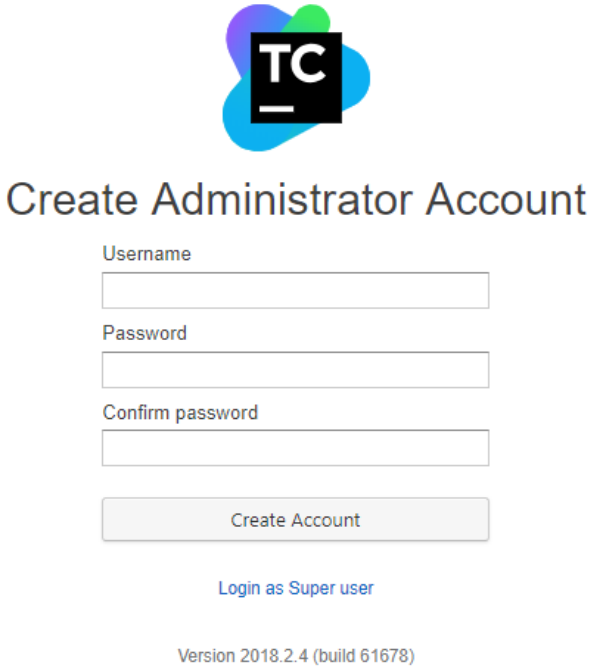

Рисунок Б.16 – Створення аккаунту адміністратора

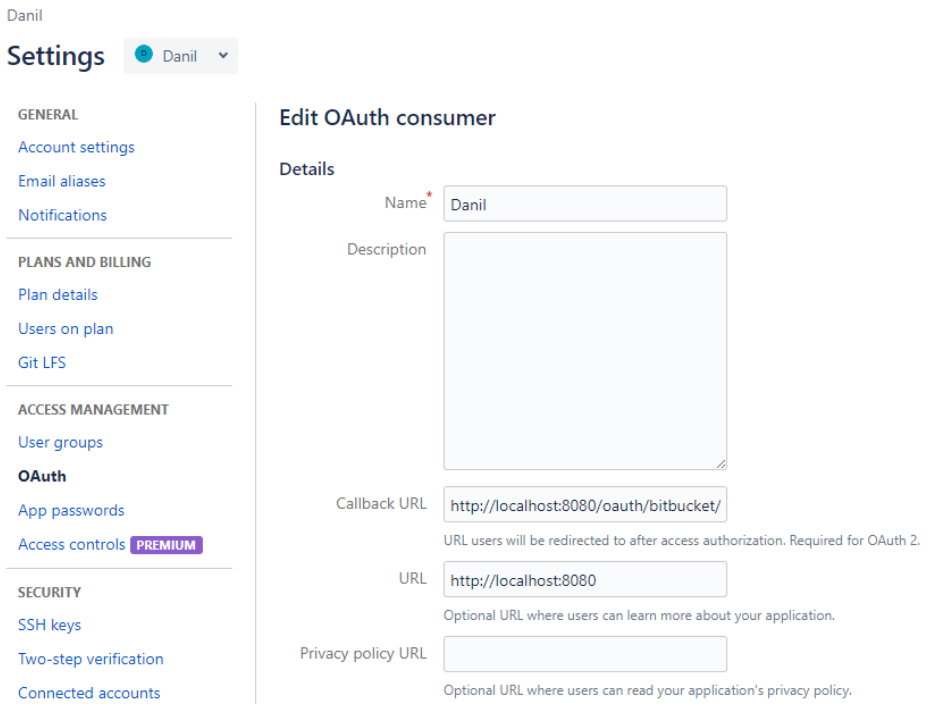

Рисунок Б.17 – Додавання OAuth Customer на BitBucket

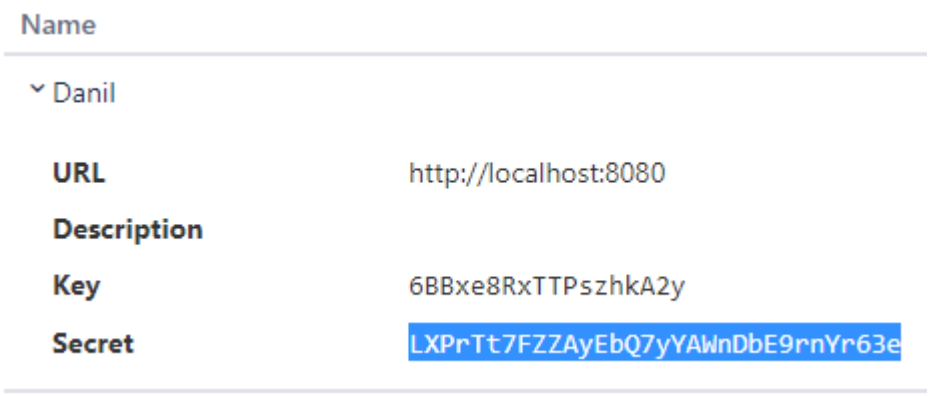

Рисунок Б.18 – Отримання Key та Secret

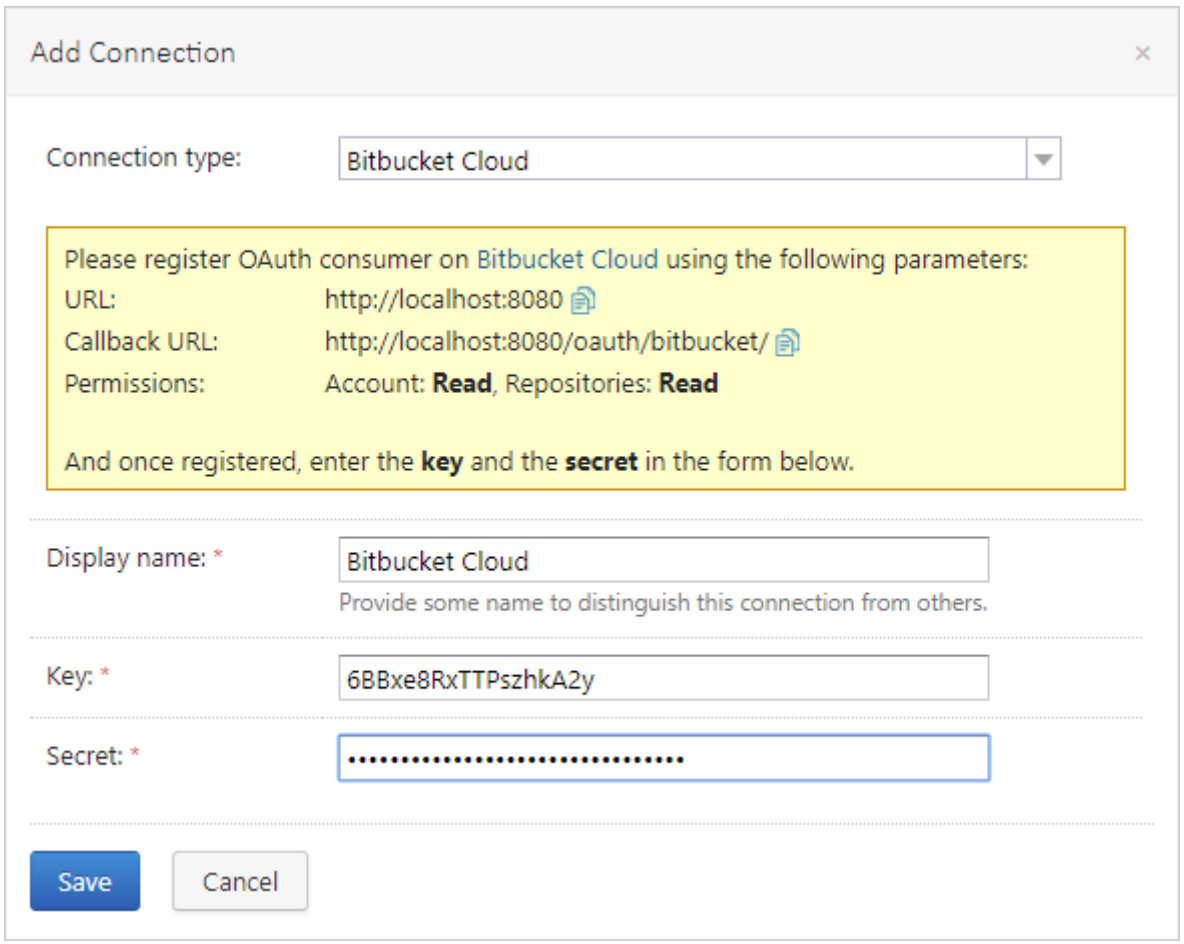

Рисунок Б.19 – Створення підключення до TeamCity

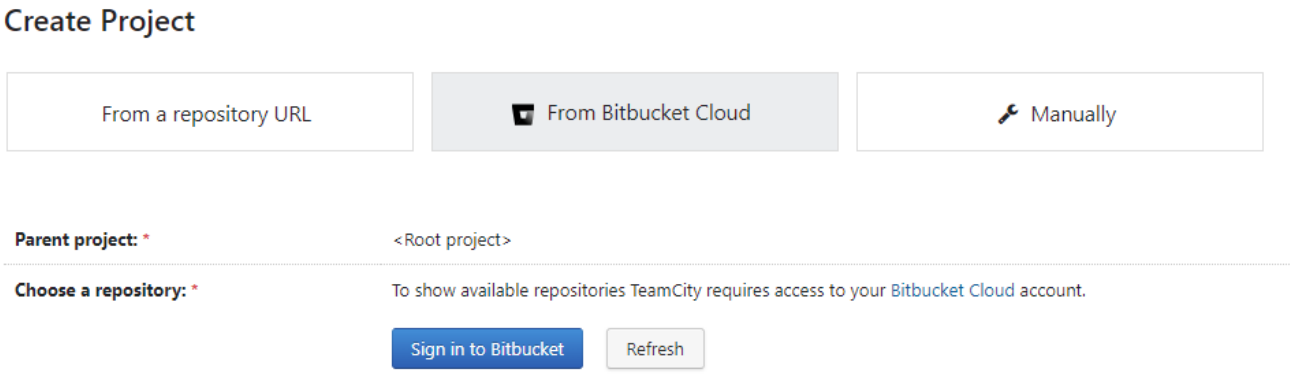

Рисунок Б.20 – Створення проекту за допомогою BitBucket Cloud

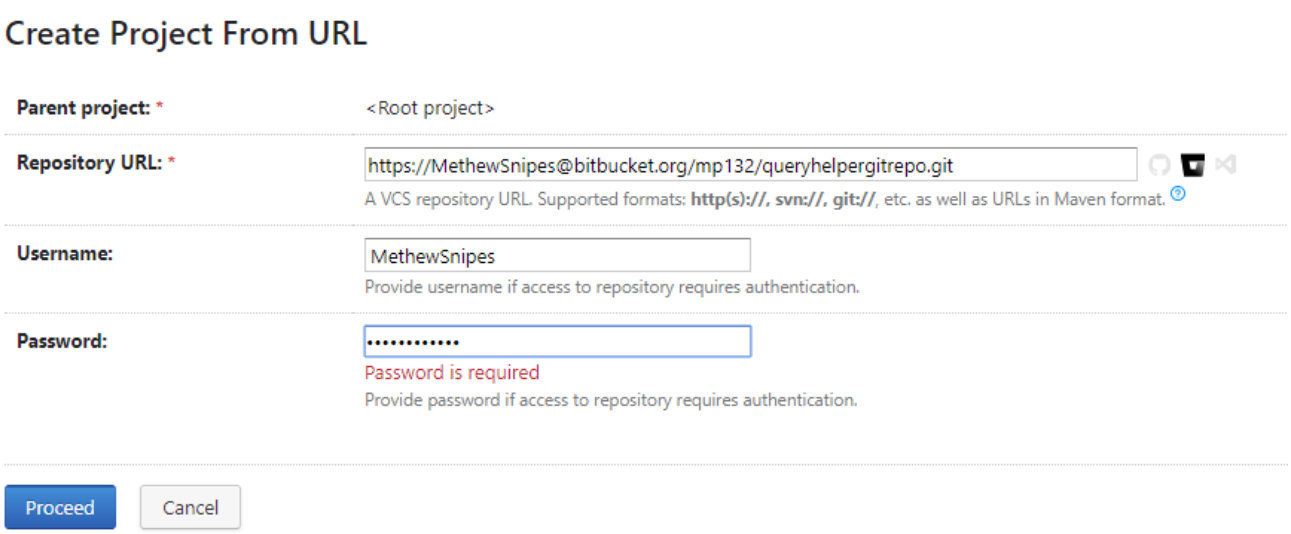

Рисунок Б.21 – Створення проекту
за допомогою раніше створеного з'єднання

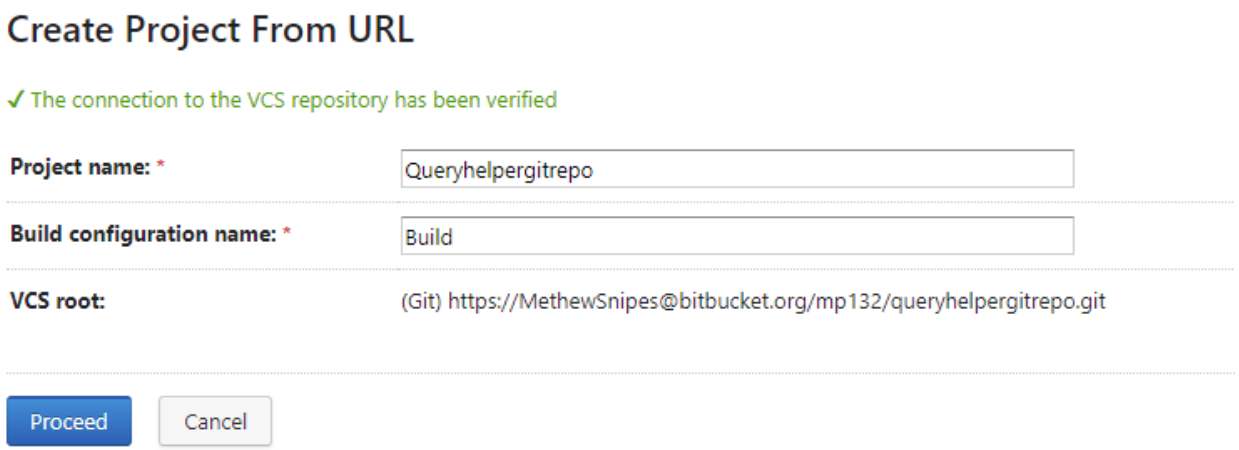

Рисунок Б.22 – Підтверджене з'єднання

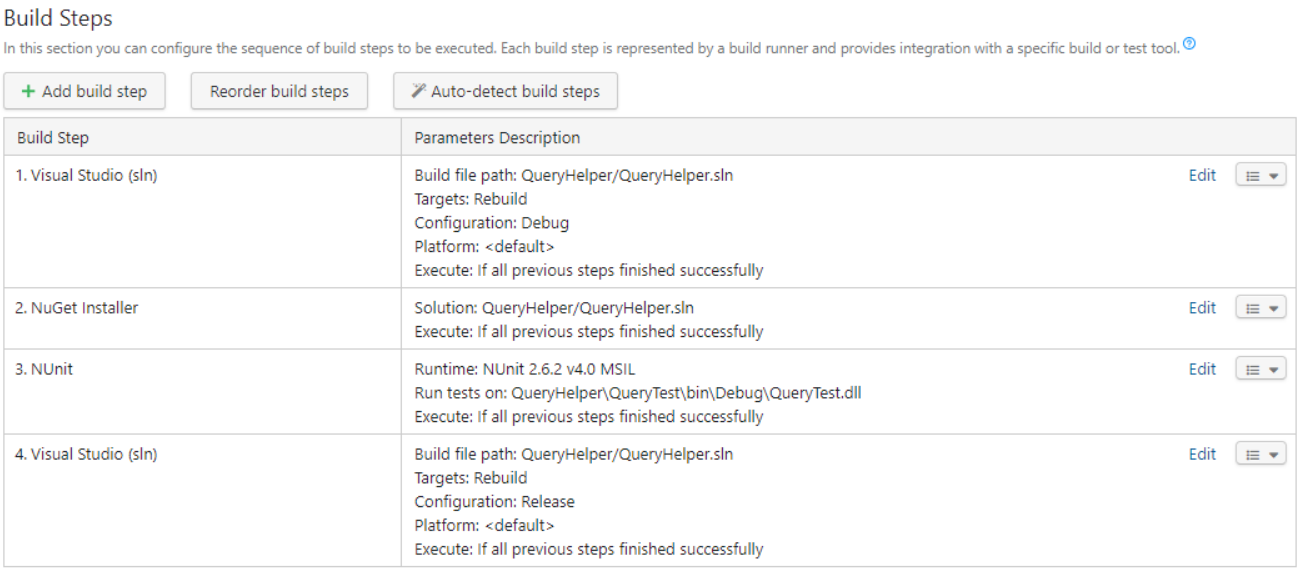

Рисунок Б.23 – Налаштування кроків побудови проекту

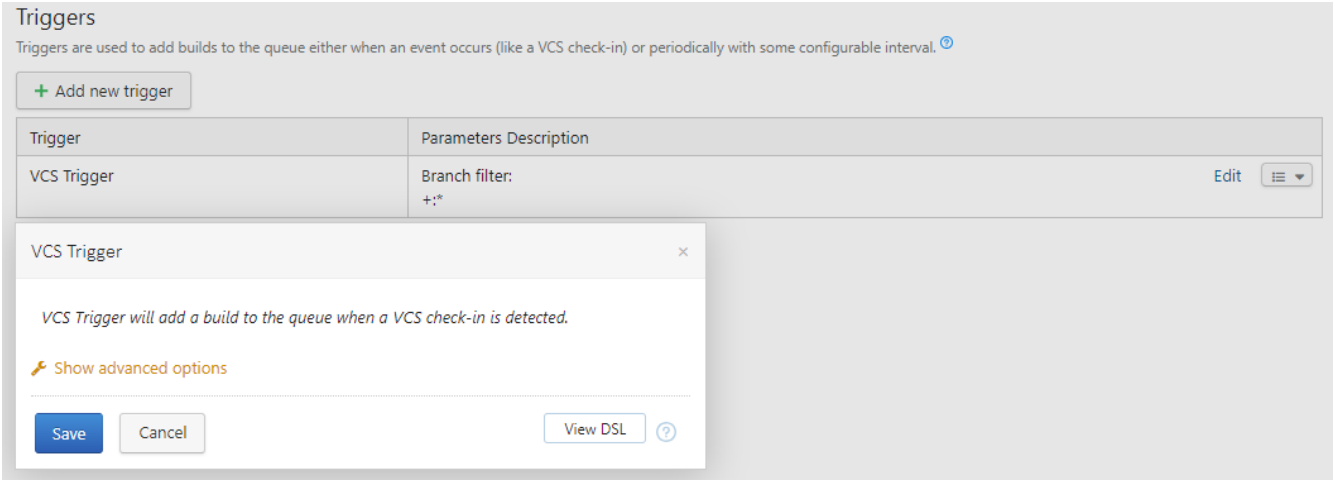

Рисунок Б.24 – Створення тригера запуску збірки

Результати безперервної збірки проекту (див. рисунки Б.25 – Б.27).

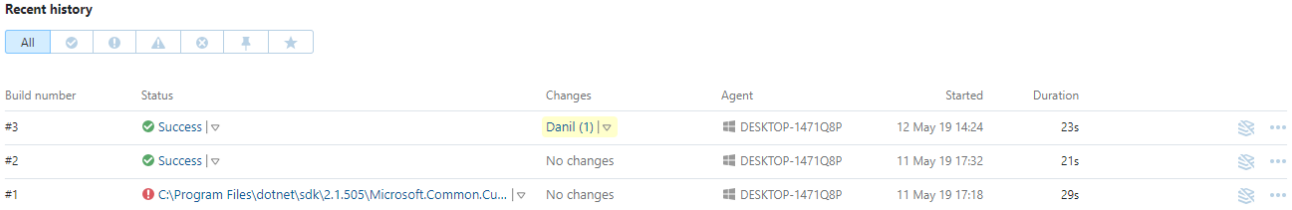

Рисунок Б.25 – Журнал збірок

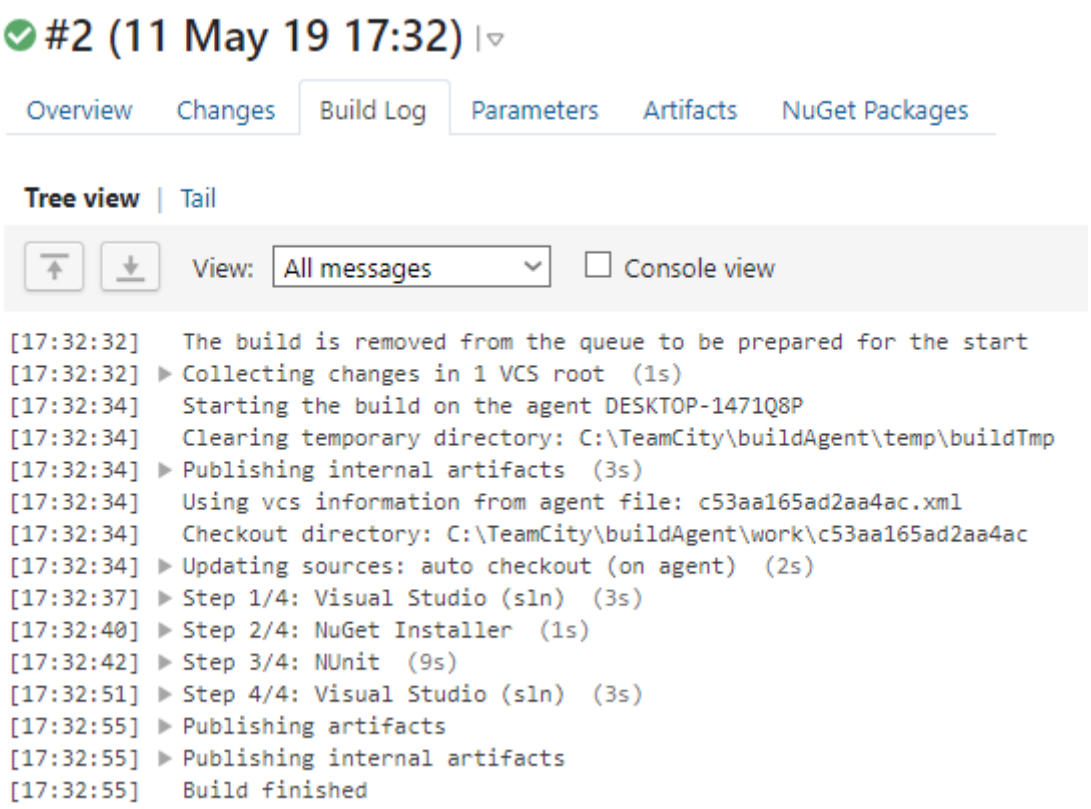

Рисунок Б.26 – Логи першої збірки

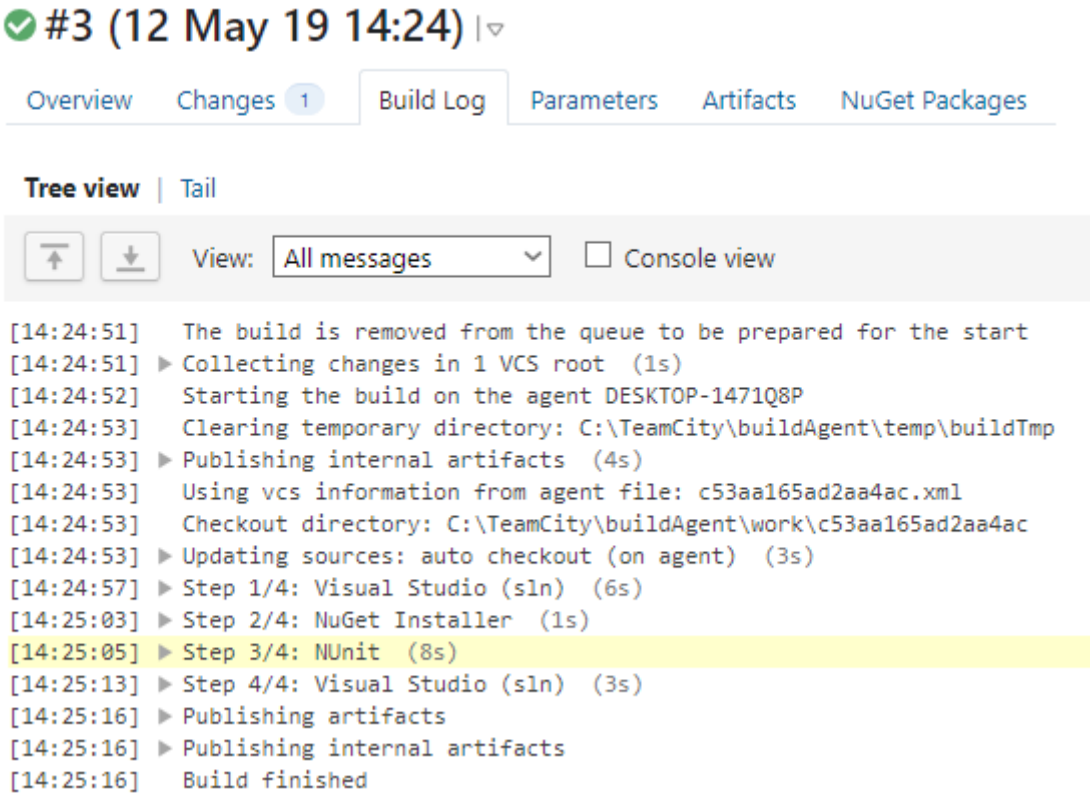

Рисунок Б.27 – Логи збірки після зміни проекту

Приклад № 2. Використовуючи інструментальні засоби, зазначені викладачем (наприклад, сервер збірки Jenkins), продемонструвати його настройку. Система контролю версій – Git, веб-хостинг – GitHub (або GitLab).

Розгорнути середу для збірки, тестування додатків з використанням сервера безперервної інтеграції.

Установка Jenkins. Для виконання роботи будемо використовувати Jenkins CI. Розміщується CI буде на VDS сервері Amazon з операційною системою Linux Ubuntu. При установці будемо використовувати ssh консольний доступ, так як Linux на сервері працює без графічного інтерфейсу.

Для початку встановимо ключ сховища, з якого Linux буде завантажувати Jenkins, для цього в консолі виконаємо команду

wget -q -O - https://pkg.jenkins.io/debian/jenkins-ci.org.key | sudo apt-key add -

Далі додамо посилання на сховище з кодом командою

echo deb http://pkg.jenkins.io/debian-stable binary / | sudo tee /etc/apt/sources.list.d/jenkins.list

Оновимо локальний список sudo apt-get update і встановимо Jenkins на наш сервер командою sudo apt-get install jenkins.

Запуск Jenkins. Для автоматичного запуску Jenkins при старті системи ми будемо використовувати systemctl який за замовчуванням вбудований в Linux. Для запуску використовуємо команду sudo systemctl start jenkins ця команда не повертає результат виконання означає перевіримо стан вручну написавши команду sudo systemctl status jenkins в результаті ми бачимо, що Jenkins буде автоматично запускається при старті системи (рисунок Б.28).

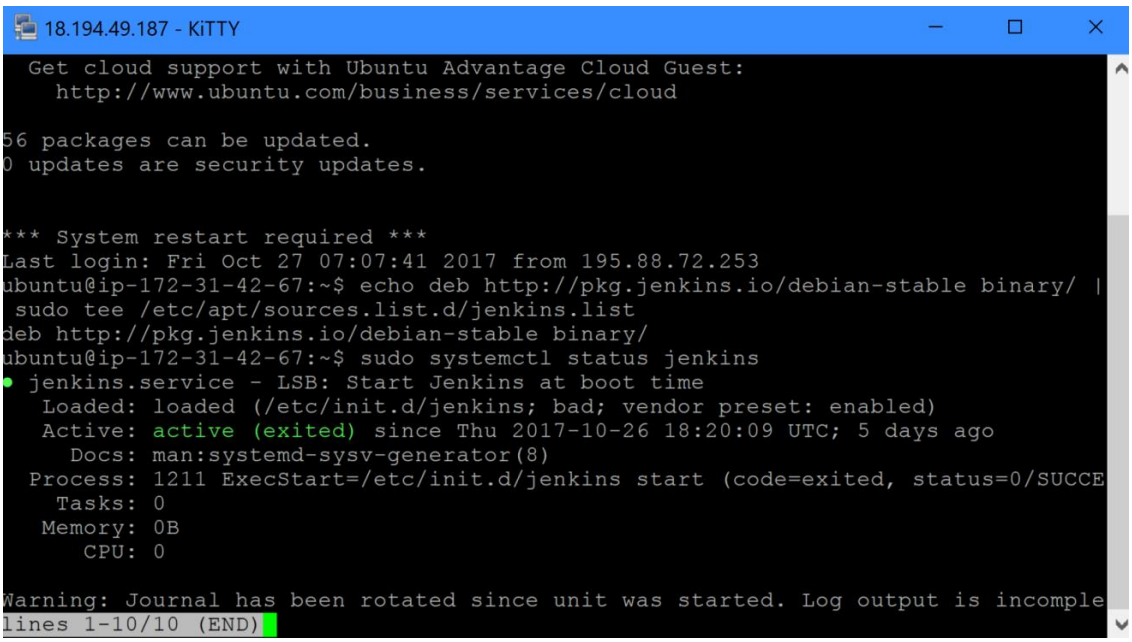

Рисунок Б.28 – Інформація в systemctl

Далі для того що б можна було увійти в web панель Jenkins потрібно дозволити доступ до порту, на якому він працює для цього виконаємо команду sudo ufw allow 8080 яка і відкриє доступ. На цьому запуск закінчений.

Базова настройка Jenkins. Для настройки Jenkins потрібно зайти в Web панель вона знаходиться по url адресою типу "http: // ip адреса або доменне ім'я: 8080 "в моєму випадку до сервера прив'язане доменне ім'я lorem.ga так що повна адреса виходить http://lorem.ga:8080. При першому заході в Web панель нам буде запропоновано активувати Jenkins за допомогою секретного пароля, який знаходиться на сервері (рисунок Б.29).

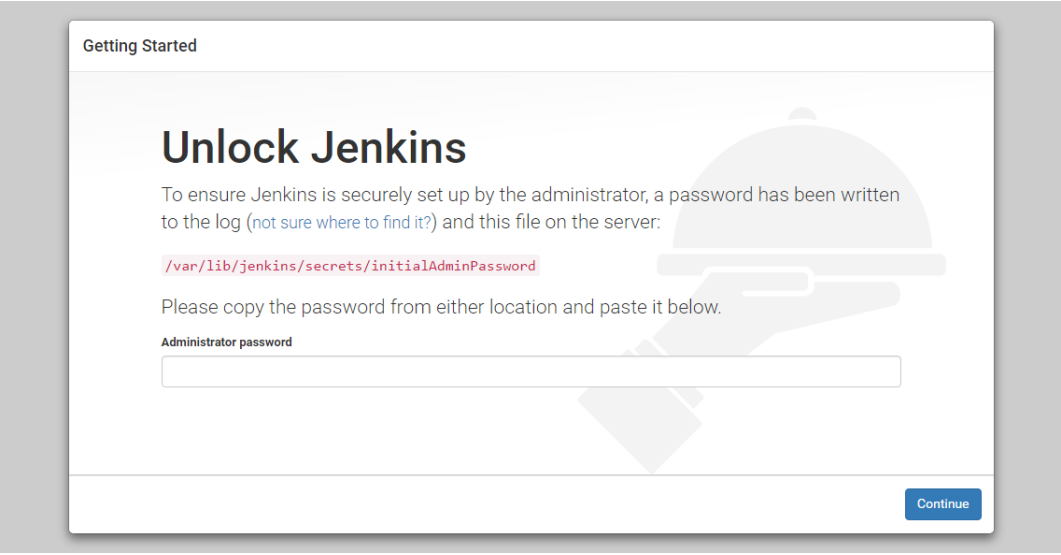

Рисунок Б.29 – Запит пароля

Дізнаємося пароль, виконавши команду

sudo cat / var / lib / jenkins / secrets / initialAdminPassword

Копіюємо його в поле і натиснемо кнопку Continue. Далі відкриється сторінка, яка запропонує встановити плагіни, є два варіанти установки:

1) встановити стандартні плагіни;

2) встановити вибрані користувачем плагіни.

Виберемо перший варіант і почекаємо поки завершиться процес установки (рисунок Б.30).

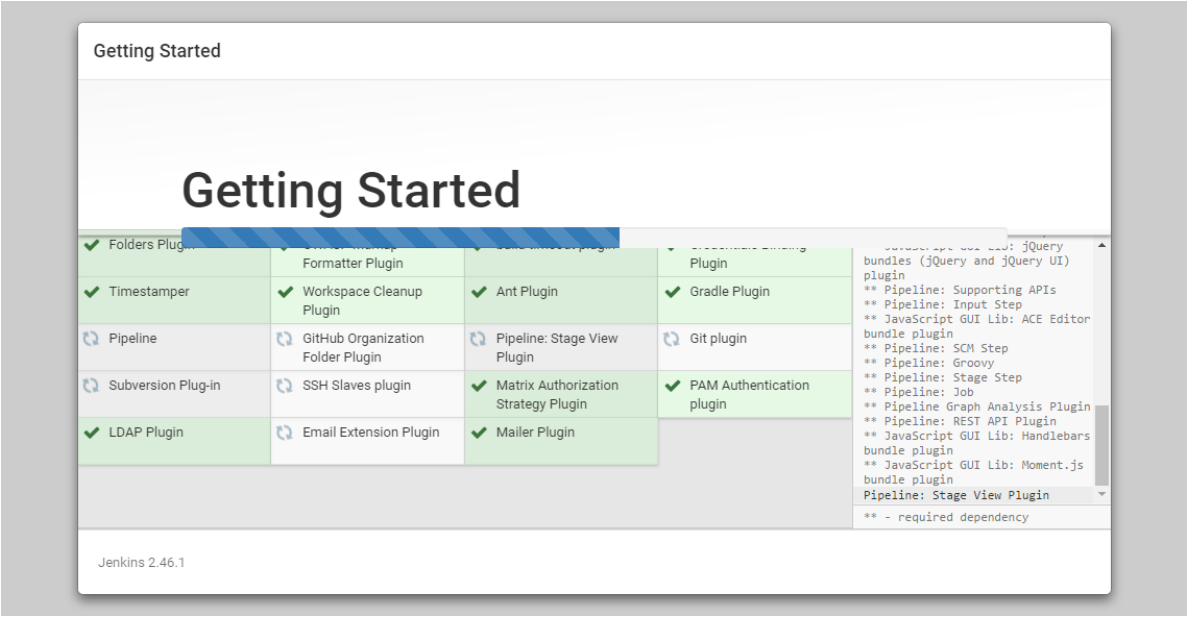

Рисунок Б.30 – Процес установки плагінів

Після установки плагінів відкриється сторінка, на якій буде запропоновано створити обліковий запис адміністратора (рисунок Б.31).

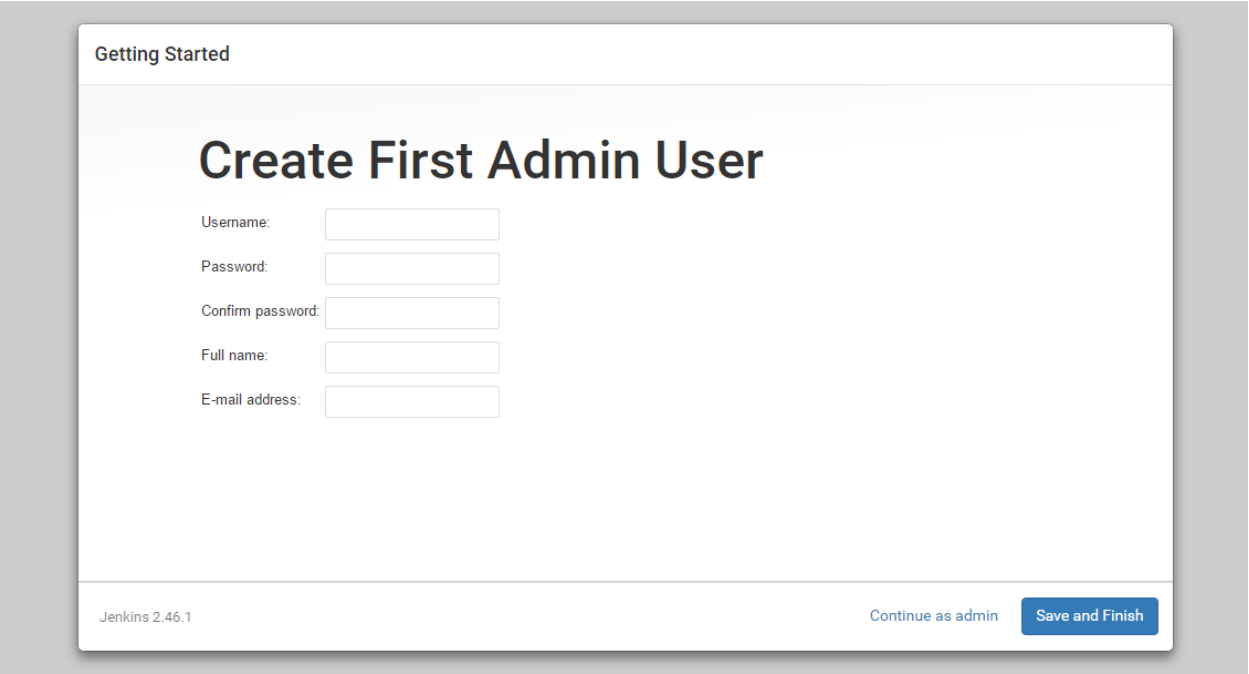

Рисунок Б.31 – Створення облікового запису адміністратора

Після заповнення всіх полів і натискання кнопки Save and Finish відкриється сторінка з підтвердженням що Jenkins готовий до роботи (рисунок Б.32).

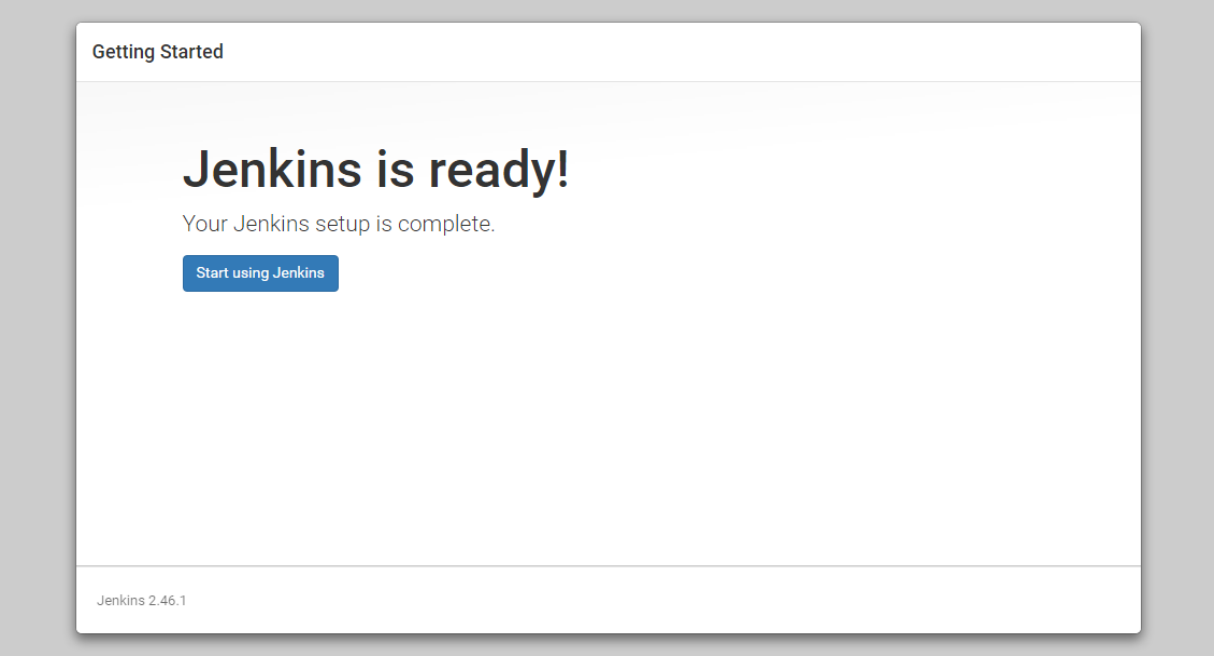

Рисунок Б.32 – Jenkins готовий до роботи

Далі при натисканні кнопки Start using Jenkins відкриється сторінка Web панелі управління (рисунок Б.33).

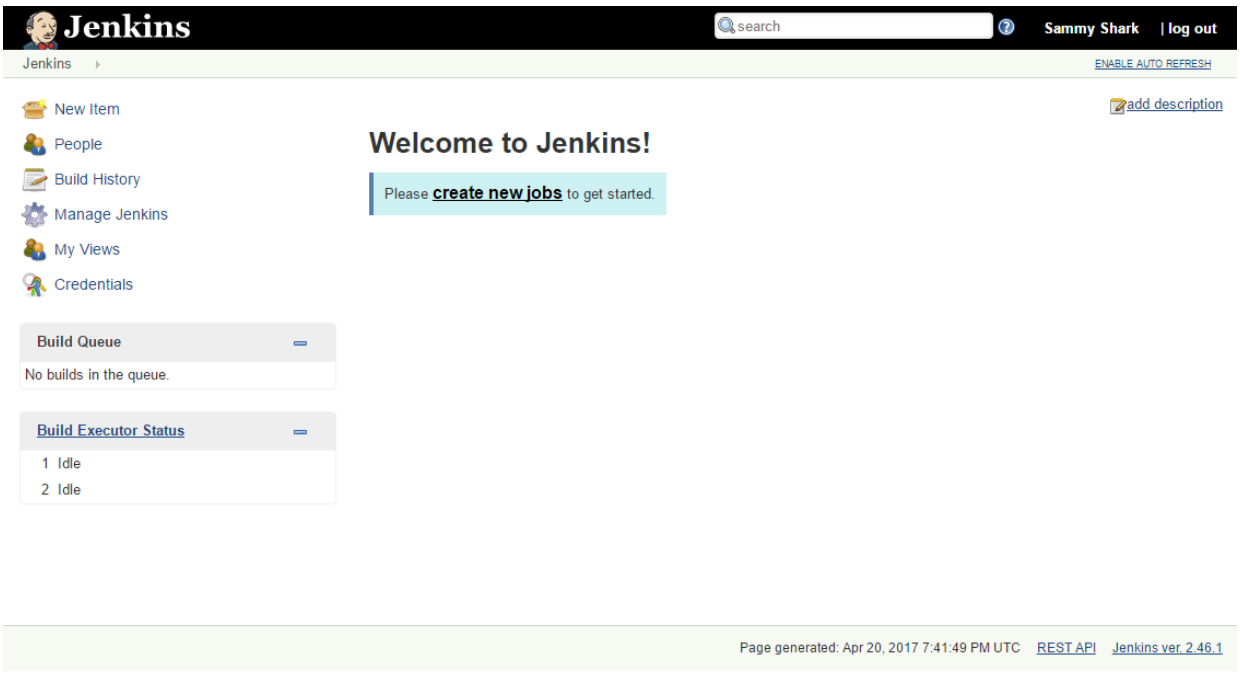

Рисунок Б.33 – Головна сторінка панелі керування

Налаштування git hook в Jenkins CI. Налаштуємо збірку проекту NodeJS в Jenkins CI. Вихідний код проекту буде знаходиться на сервері системи контролю версій github, при будь-якій зміні на github за допомогою web хука Jenkins буде отримувати інформацію що відбулися зміни і завантажувати проект на сервер, а потім виконувати дії, описані у файлі Jenkinsfile, який знаходиться в корені проекту.

Заходимо на github далі settings (рисунок Б.34).

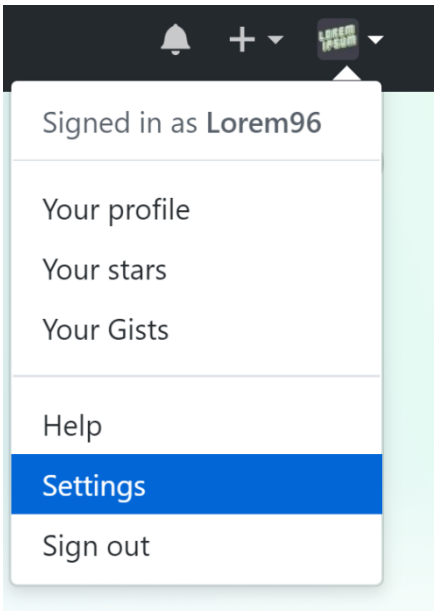

Рисунок Б.34 – Налаштування GitHub

Далі переходимо у вкладку Developer setting далі Personal access tokens і натискаємо кнопку Generate new token (рисунок Б.35).

Рисунок Б.35 – Генерація токена

Далі нам необхідно вписати опис токена і поставити потрібні для Jenkins галочки (Рисунок Б.36).

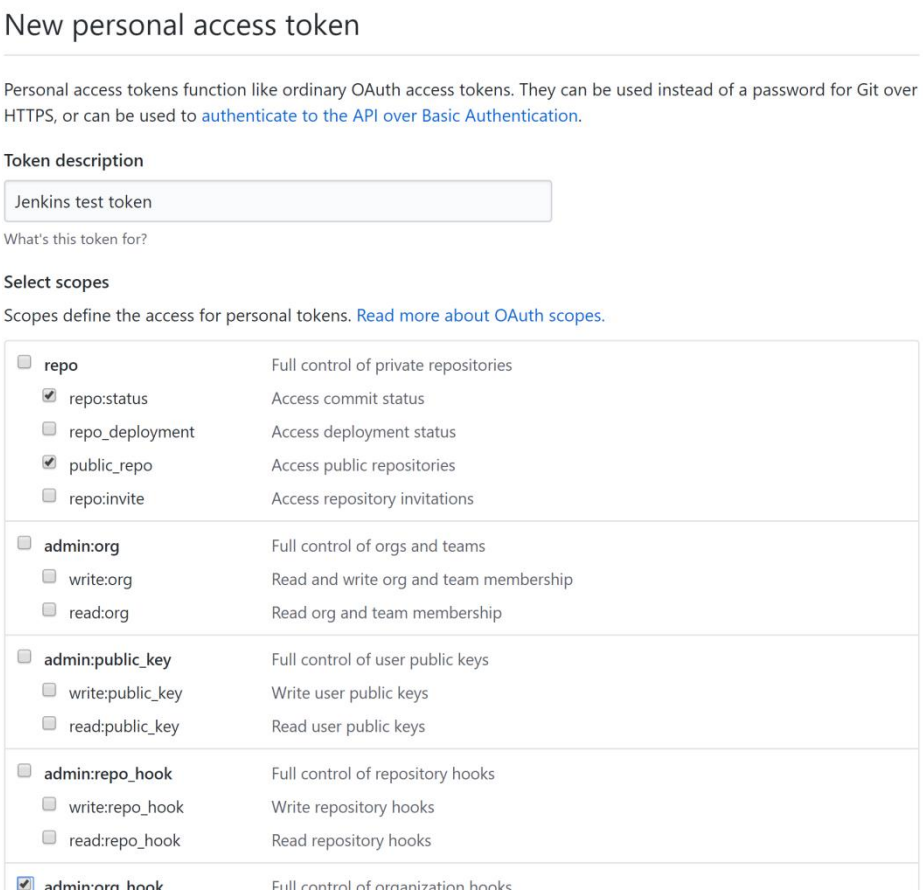

Рисунок Б.36 – Створення токена

І натискаємо кнопку Generate token далі копіюємо згенерований токен 51b204653bc02863f1d2e4beb52201471215e7ba. Після того як ми отримали свій токен додамо його як глобальний в Jenkins. Для додавання заходимо в пункт Credentials (рисунок Б.37).

Рисунок Б.37 – Меню Jenkins

Натискаємо на global і вибираємо Add credentials (рисунок Б.38).

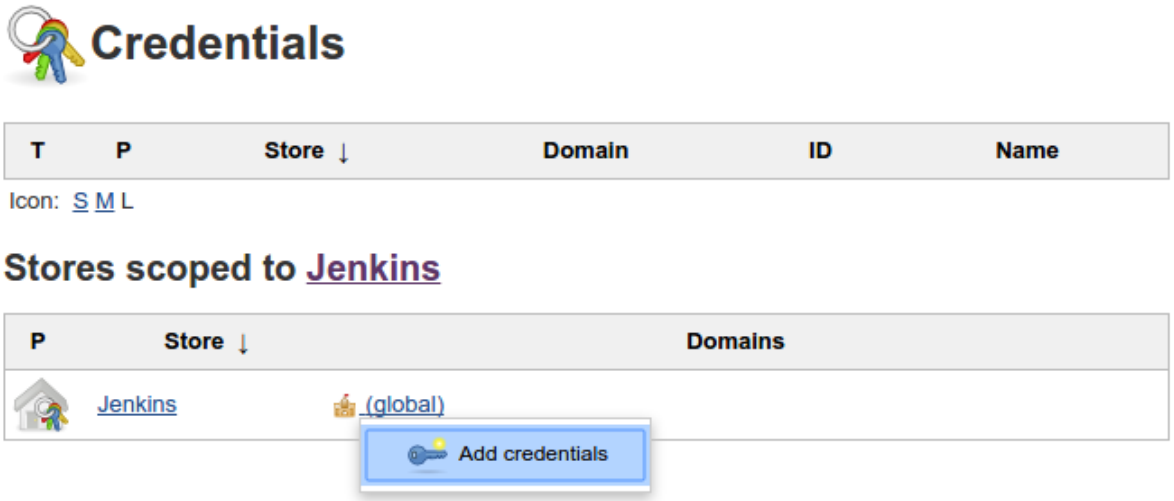

Рисунок Б.38 – Додавання credentials

Вказуємо що це секретний текст, в поле Secret вписуємо наш токен і натискаємо Ok (рисунок Б.39).

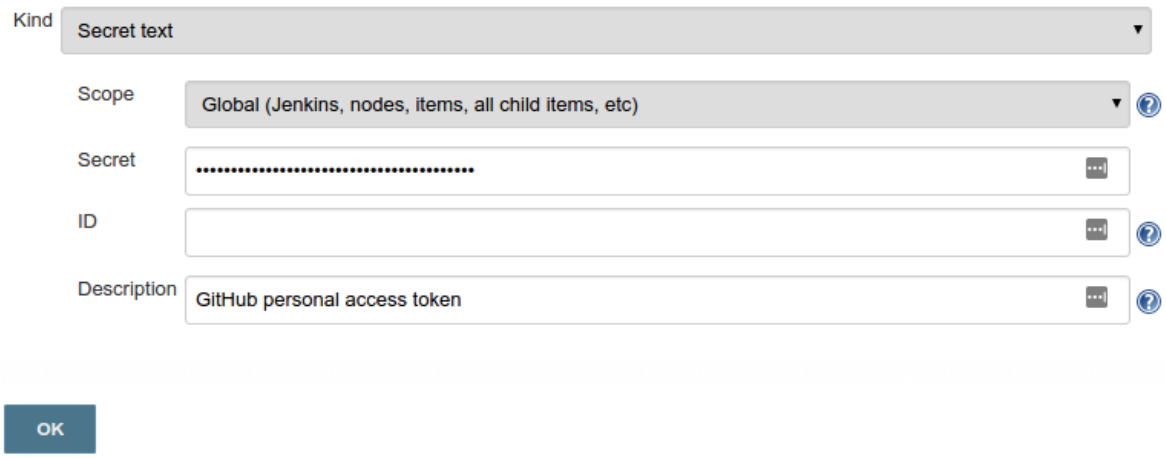

Рисунок Б.39 – Створення токена

Далі налаштуємо Jenkins на наш github сервер для цього заходимо в налаштування далі настройки системи і в пункті GitHub Server вказуємо посилання на api github при цьому також встановлюємо в credentials наш секретний токен який ми додавали до цього і ставимо галочку Manage hooks. Можна відразу ж протестувати все ми правильно вказали, натиснувши на кнопку Test connections, якщо все вірно, то результатом буде Credentials verified for user Username, rate limit: 4993 (рисунок Б.40).

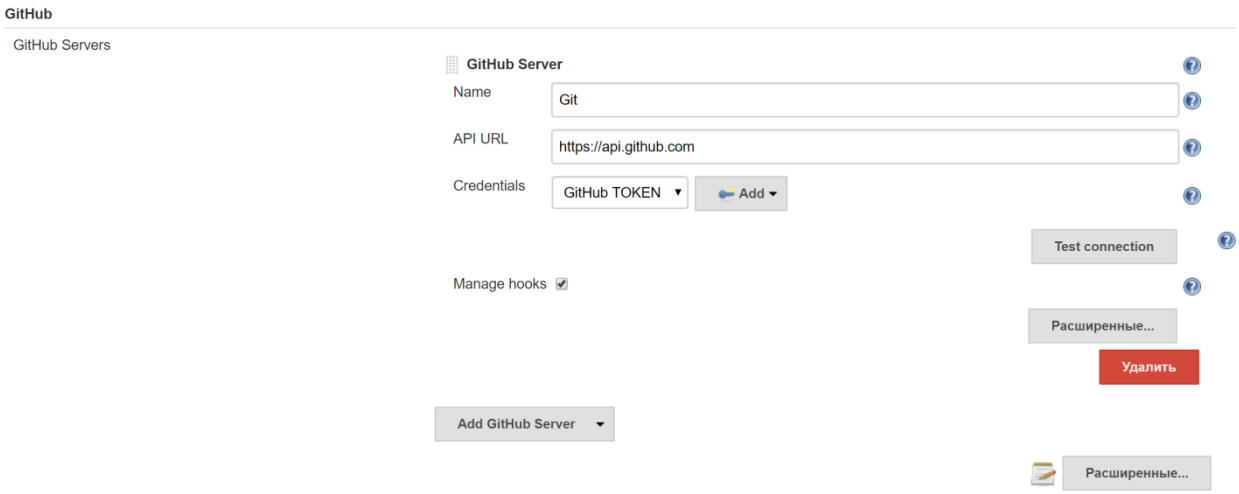

Рисунок Б.40 – Налаштування github сервера

Налаштування проекту NodeJS в Jenkins CI. Далі налаштуємо скачування і запуск проекту NodeJS з репозиторію github при зміні.

Для початку додамо файл конфігурації для нашого проекту, в файлі будуть описані кроки, які Jenkins повинен виконати після того як завантажить наш проект, файл називається Jenkinsfile і знаходиться в корені проекту. Вміст файлу таке:

```
#!/usr/bin/env groovy
pipeline {
  agent any
   stages {
      stage('Install') {
         steps {
            echo 'Install..'
            sh 'npm install'
         }
      }
      stage('Stop all') {
         steps {
            echo 'Stop all..'
            sh 'forever stopall'
         }
```

```
}
 stage('Deploy') {
    steps {
        echo 'Deploying.'
        sh 'BUILD_ID=dontKillMe pm2 start app.js'
    }
}    }}
```

Далі в головному меню Jenkins заходимо в Створити item вводимо ім'я і вибираємо Pipeline далі натискаємо кнопку Ok (рисунок Б.41).

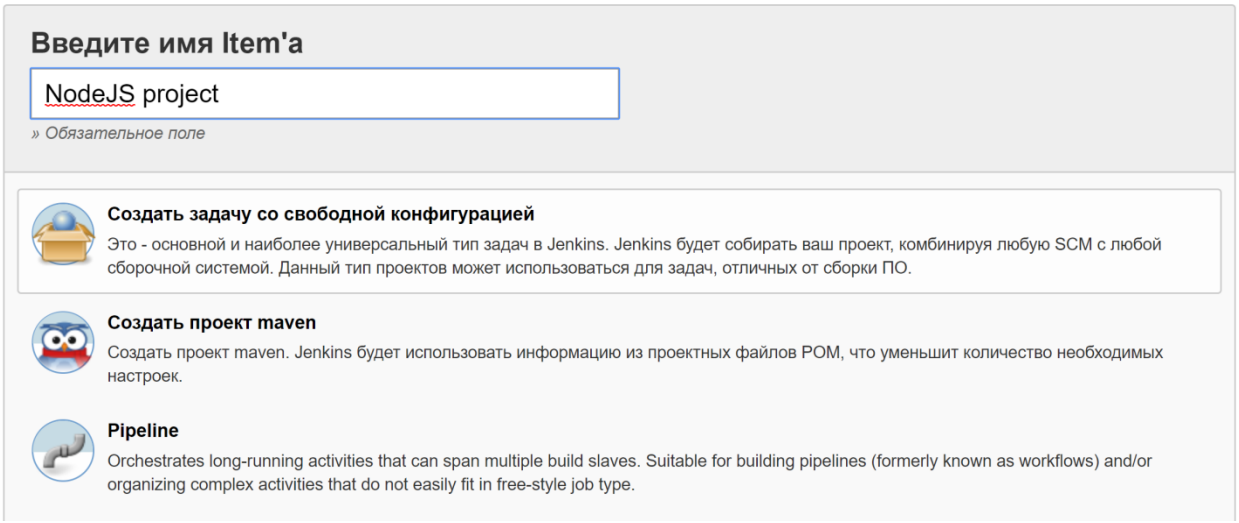

Рисунок Б.41 – Створення pipeline

Далі відкривається сторінка налаштування проекту в якій необхідно поставити галочку GitHub project і в поле Project url вказуємо адресу проекту на github (рисунок Б.42).

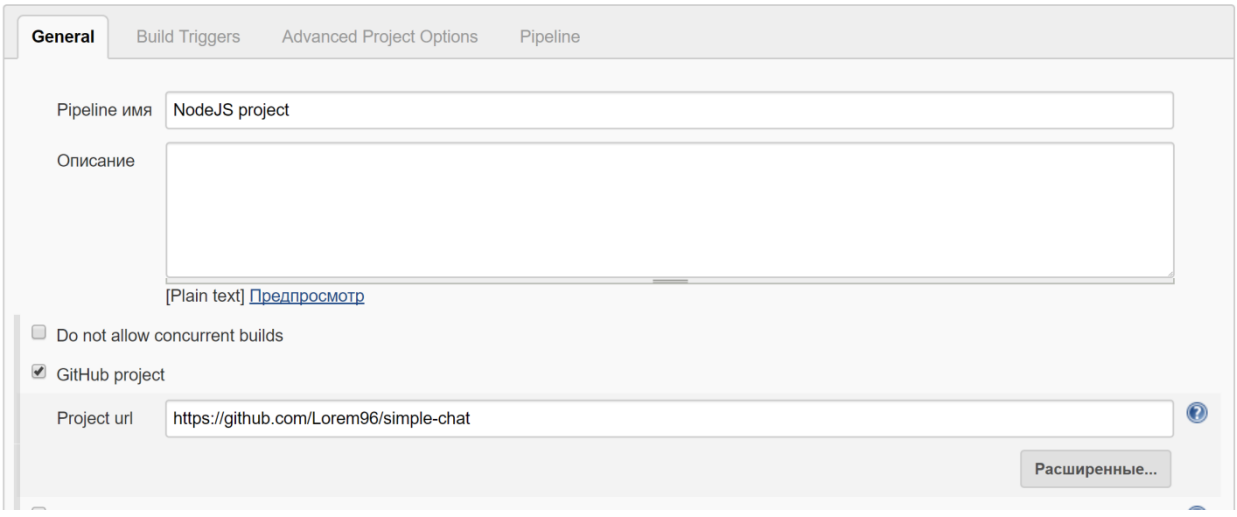

Рисунок Б.42 – Налаштування pipeline

Також необхідно встановити галочку в пункті Build Triggers для того що б Jenkins використовував хук при оновленні сховища (рисунок Б.43).

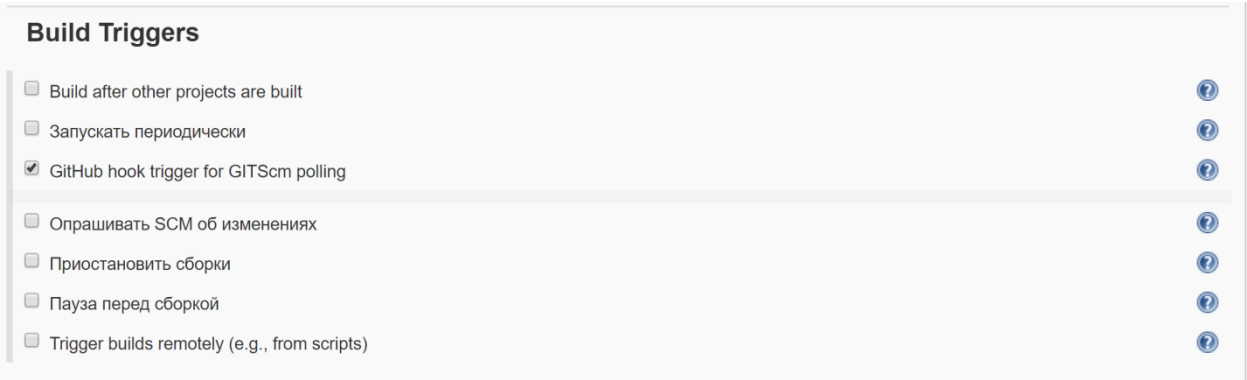

Рисунок Б.43 – Налаштування тригера

Далі в розділі Pipeline необхідно вказати що ми будемо використовувати конвеєр, конфігурація якого розташована в файлі Jenkinsfile, для цього в Definition вибираємо Pipeline script from SCM і в SCM вибираємо GIT і вказуємо посилання на репозиторій і вказуємо гілку яку хочемо використовувати (рисунок Б.44).

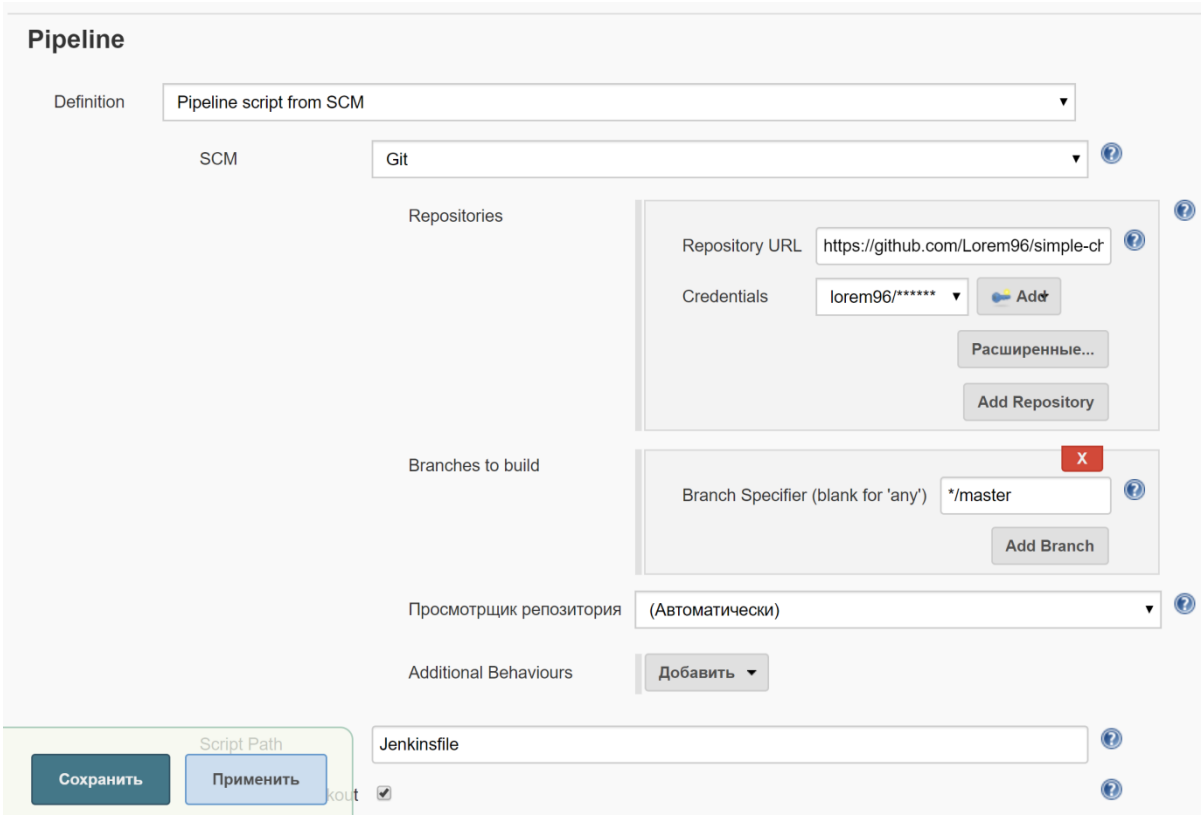

Рисунок Б.44 – Налаштування сховища для тригера

У Credentials необхідно вказати аккаунт, який ми додамо, натискаючи на кнопку Add. У вікні необхідно вказати логін і пароль від github (рисунок Б.45).

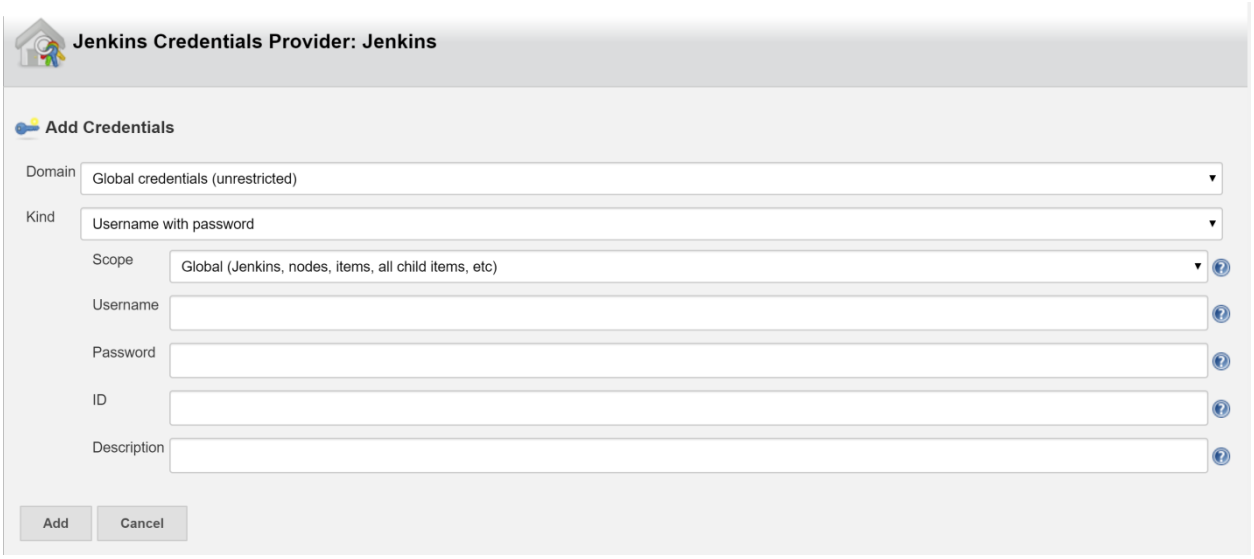

Рисунок Б.45 – Створення Credentials з аккаунтом

Після натискання кнопки зберегти відкриється сторінка нашого конвеєру в якій нам потрібно натиснути на кнопку Зібрати зараз (рисунок Б.46).

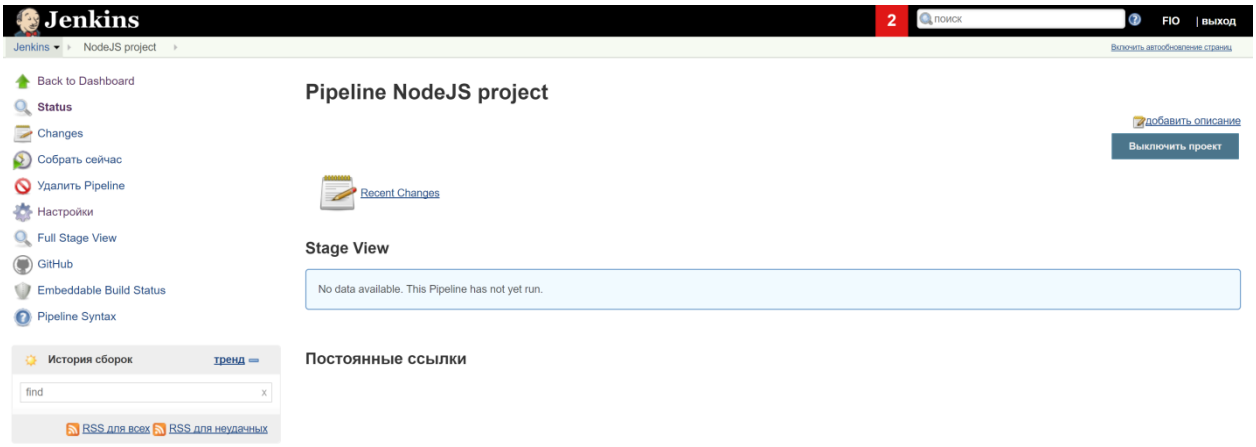

Рисунок Б.46 – Головна сторінка конвеєра

Далі в історії збірок ми можемо побачити, що збірка почалася (рисунок Б.47).

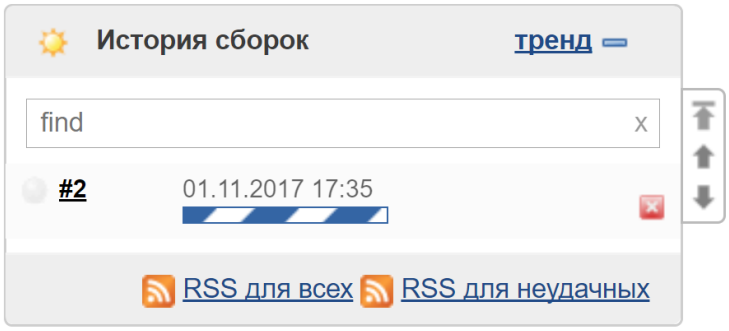

Рисунок Б.47 – Історія збірок

В Stage View ми можемо бачити хід виконання збірки (рисунок Б.48).

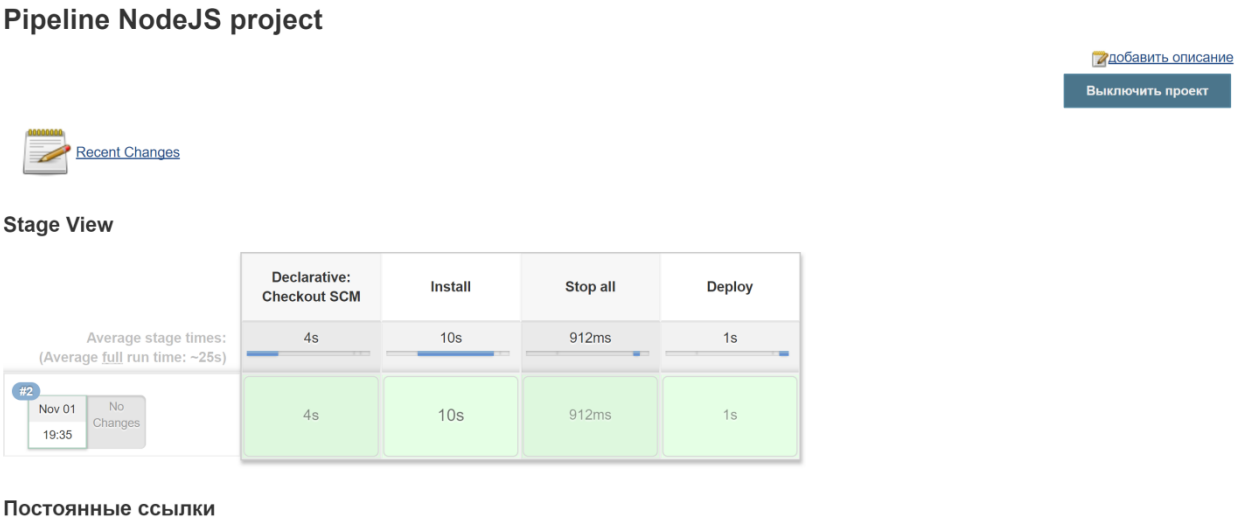

Рисунок Б.48 – Хід виконання збірки

Далі постійна ручна активація більше не обов'язкова Jenkins сам буде довантажувати змінені файли при Ком на сервері github і виконувати збірку. Для перевірки змінимо що ні будь в локальній копії нашого проекту і закоммітім на сервер github з назвою коммітов "TEST JENKINS". Після коммітов Jenkins скачав змінені файли і виконав складання, можемо переконається в цьому, подивившись в Stage View і навівши на поле з коммітов побачити назву останнього коммітов (рисунок Б.49).

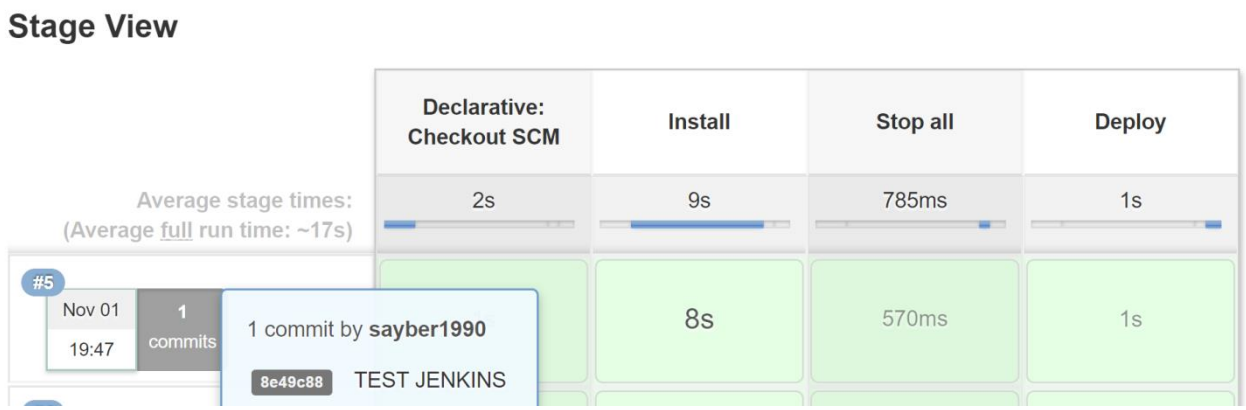

Рисунок Б.49 – Інформація про комміт

Так само в Stage View ми можемо побачити інформацію про кожен етап збірки проекту час, витрачений на складання і лог при виконанні кожного етапу (рисунок Б.50).

Рисунок Б.50 – Інформація про крок збірки

В результаті була протестована робота сервера безперервної інтеграції шляхом запуску кількох збірок і доповнень змін в репозиторій Git.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. Анализ требований и создание архитектуры решений на основе Microsoft .NET, Русская редакция, 2004.
2. У. Ройс. Управление проектами по созданию программного обеспечения. Унифицированный подход. – Лори, 2002.
3. К. Бек. Экстремальное программирование. – Питер, 2002.
4. S. Guckenheimer, J.Perez. Software Engineering with Microsoft Visual Studio Team System. – Addison-Wesley Professional, 2006.
5. Шторгина Елена Сергеевна. Системы контроля версий. Сетевые технологии. Технологии Интернет. Лекция 2. – 82 с.
6. Тестирование Дот Ком, или пособие по жёсткому обращению с багами в интернет-стартапах. – М.: Дело, 2007. – 312 с.
7. Сергей Орлик. Программная инженерия. Тестирование программного обеспечения. – 2004 – 2005. – 16 с.
8. Марк Симан. Внедрение зависимостей. – Изд-во «Питер», 2013. – 624 с.
9. Ericsson A. B. Secure Socket Layer. – ERLANG, September 20, 2016. – 35 p.
10. Липаев В. В. Сертификация программных средств : учеб. пособ. – М. : СИНТЕГ, 2010. – 348 с.
11. PMBOK – Project Management Body of Knowledge, USA, 2005.
12. Е. М. Лаврищева, В. А. Петрухин. Методы и средства инженерии программного обеспечения. – Библиотека учебных курсов MSDN AA, 2007.
13. Ф. Брукс. Мифический человеко-месяц, или как создаются программные системы. – Символ-Плюс, 2006.
14. Э. Йордан. Путь камикадзе. – Лори, 2001.
15. В. В. Липаев. Программная инженерия. Методологические основы. – ТЕИС, 2006.
16. А. Н. Терехов. Технология программирования. – Интуит Бином, 2007.
17. Pro Git. Scott Chacon. 2012-08-31. This is the PDF file for the Pro Git book contents. It is licensed under the Creative Commons Attribution-Non Commercial-Share Alike 3.0 license. I hope you enjoy it, I hope it helps you learn Git, and I hope you'll support Apress and me by purchasing a print copy of the book at Amazon: http://tinyurl.com/amazonprogit

18. Ben Collins-Sussman. Version Control with Subversion. For Subversion 1.7 (Compiled from r5206). Ben Collins-Sussman Brian W. Fitzpatrick C. Michael Pilato. 2011, 433 p. This work is licensed under the Creative Commons Attribution License. To view a copy of this license, visit http://creativecommons.org/licenses/by/2.0/ or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.
19. Бен Коллинз-Сассмэн, Брайан Фитцпатрик. Идеальная IT-компания. Как из гиков собрать команду программистов. – ООО Издательство «Питер», 2014. – 230 с.
20. Mercurial: The Definitive Guide by Bryan O'Sullivan http://hgbook.red-bean.com/read/
21. https://hgbook.bacher09.org/html/index.html
22. https://dou.ua/lenta/articles/mercurial-step-by-step-dvcs-intro/
23. Ben Lynn. Волшебство Git. Руководство выпущено под GNU General Public License 3-й версии. Исходный текст находится в хранилище Git и может быть получен командой: $ git clone git://repo.or.cz/gitmagic.git #.
24. http://www-cs-students.stanford.edu/~blynn/gitmagic/intl/ru/index.html
25. http://tortoisehg.bitbucket.org/screenshots.html#mq
26. http://tortoisehg.bitbucket.org/
27. https://habrahabr.ru/post/154255/
28. http://rus-linux.net/MyLDP/BOOKS/Architecture-Open-Source-Applications/Vol-2/git-02.html
29. Лекции по инструментам управления конфигурацией, Иртегов Д.В, http://parallels.nsu.ru/~fat/subversion.ppt
30. https://books.google.com.ua/books?id=kZ5nAAAAQBAJ&pg=PA75&lpg=PA75&dq=%D1%80%D0%B0%D1%81%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5+%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85+%D0%B2+%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B5+%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D1%8F+%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D0%B9&source=bl&ots=iCnBVHOYMg&sig=u9TdcTMnkUhnl6qmL6BOSdt9CHE&hl=ru&sa=X&ved=0ahUKEwjyvtbC_t3QAhXJhiwKHS0DBiYQ6AEIKzAD#v=onepage&q=%D1%80%D0%B0%D1%81%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5%20%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%

85%20%D0%B2%20%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B5%20%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D1%8F%20%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D0%B9&f=false

31. https://ru.wikipedia.org/wiki/Subversionhttps://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%8F%D0%BC%D0%B8
32. Управление версиями в Subversion, http://svnbook.red-bean.com/
33. Сайт с материалами курса: http://www.inteks.ru/PM/
34. https://confluence.atlassian.com/bamboo/mercurial-289277014.html
35. https://developer.mozilla.org/en-US/docs/Mercurial/Using_Mercurial
36. Обзор систем контроля версий http://all-ht.ru/inf/prog/p_0_1.html
37. Рейтинг систем контроля версий http://tagline.ru/version-control-systems-rating/
38. Сравнение систем контроля версий. http://mitra.ru/events/news/website_3.html
39. Куликов Святослав. Тестирование программного обеспечения. Базовый курс. – EPAM Systems, 2016. – 289 с. http://svyatoslav.biz/software_testing_book/
40. https://msdn.microsoft.com/ru-ru/library/stxxt3fd(v=vs.90).aspx
41. Нумерация версий https://habrahabr.ru/post/119400/
42. S. Guckenheimer, J.Perez. Software Engineering with Microsoft Visual Studio Team System. – Addison-Wesley Professional, 2006.
43. http://wixtoolset.org/documentation/manual/v3/howtos/updates/major_upgrade.html
44. https://msdn.microsoft.com/en-us/library/windows/desktop/aa370037(v=vs.85).aspx
45. https://www.safaribooksonline.com/library/view/wix-36-a/9781782160427/ch13s04.html
46. http://helpnet.flexerasoftware.com/installshield22helplib/helplibrary/MajorMinorSmall.htm – 2015

Навчальне видання

Данова Марія Олександрівна
Кузнецова Юлія Анатоліївна
Сьомочкін Максим Олександрович

ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Редактор Н. В. Мазепа

Зв. план, 2021
Підписано до видання 08.06.2021
Ум. друк. арк. 5. Обл.-вид. арк. 5,625. Електронний ресурс

Видавець і виготовлювач
Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»
61070, Харків-70, вул. Чкалова, 17

Видавничий центр «ХАІ»
61070, Харків-70, вул. Чкалова, 17

Свідоцтво про внесення суб'єкта видавничої справи
до Державного реєстру видавців, виготовлювачів і розповсюджувачів
видавничої продукції сер. ДК № 391 від 30 03.2001