Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра мережевих та інтернет технологій

ЗАТВЕРДЖУЮ
завідувач кафедри
мережевих та інтернет технологій

_____ Ю.В. Кравченко

«_____»_____2021 року

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

галузі знань 17 «Електроніка та телекомунікації»
за спеціальністю 172 «Телекомунікації та радіотехніка»

на тему:

АВТОМАТИЗАЦІЯ ВИКОНАННЯ ТА ОЦІНЮВАННЯ ЛАБОРАТОРНИХ ТА ПРАКТИЧНИХ РОБІТ З ПРОГРАМУВАННЯ НА ОСНОВІ ТЕХНОЛОГІЙ БЕЗПЕРЕРВНОЇ ІНТЕГРАЦІЇ ТА РОЗГОРТАННЯ (CI/CD)

Виконав: студент групи МІТ - 41

Науменко Денис Ігорович _____
(прізвище ім'я по-батькові) (підпис)

Керівник: професор кафедри мережевих та інтернет технологій

д.т.н. Миколайчук Р.А. _____
(посада, прізвище ім'я по-батькові) (підпис)

Київ 2021

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра мережевих та інтернет технологій

ЗАТВЕРДЖУЮ
завідувач кафедри
мережевих та інтернет технологій

_____ Ю.В. Кравченко

«_____»_____2021 року

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ

Здобувачу вищої освіти Науменко Денис Ігорович
(прізвище, ім'я, по батькові)

1. Тема роботи:

«Автоматизація виконання та оцінювання лабораторних та практичних робіт з програмування на основі технологій безперервної інтеграції та розгортання (CI/CD)»

затверджена на засіданні кафедри МІТ «11» грудня 2020 р. протокол № 6

2. Термін здачі закінченої роботи «30» травня 2021р

3. Вихідні дані до проекту (роботи)

Застосунок для перегляду статистики та порівняння робіт студентів

Програмне забезпечення та скріпти налаштування конвейерів CI/CD

4. Зміст пояснювальної записки (перелік питань, що їх потрібно розробити, обсяг – 35-50 стор.)

1. Дослідження підходів та інструментів для реалізації CI/CD в предметній області.
2. Проектування системи автоматизації на основі технологій CI/CD в предметній області.
3. Створення системи автоматизації на основі технологій CI/CD в предметній області.

5. Перелік графічного матеріалу 8-12 слайдів

Вибір інструментів для реалізації CI/CD та автоматизації.

Проектування компонентів.

Реалізація проекту.

Дослідження продуктивності конвейерів CI/CD Workflow.

Висновки

Дата видачі завдання

Керівник роботи д.т.н., професор кафедри МІТ Миколайчук Р.А.
(підпис) (посада, прізвище, ім'я, по батькові)

Завдання прийняв до виконання
(підпис) (прізвище, ім'я, по батькові)

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ РОБОТИ

Номер	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Підготовчий	29.01.2020	
2	Розділ 1	01.03.2020	
3	Розділ 2	01.04.2020	
4	Розділ 3	01.05.2020	
5	Доповідь та слайди	27.05.2020	
6	Пояснювальна записка	30.05.2020	

Здобувач вищої освіти _____
(підпис) (прізвище, ім'я, по батькові)

Керівник _____
(підпис) (прізвище, ім'я, по батькові)

РЕФЕРАТ

Пояснювальна записка: 61с., 34 рис., 4 табл., 15 джерел.

Об’єкт дослідження: підходи автоматизації на основі технологій безперервної інтеграції та розгортання CI/CD.

Предмет дослідження: процес проектування та реалізації системи автоматизації на основі технологій безперервної інтеграції та розгортання CI/CD.

Мета роботи (проекту): розробити програмне забезпечення для автоматизації виконання та оцінювання лабораторних та практичних робіт з програмування.

Методи дослідження: системний підхід, методи порівняння, узагальнення.

У спеціальній частині дана характеристика сучасного стану технологій безперевної інтеграції та розгортання.

В роботі проведено аналіз існуючих технологій та підходів автоматизації до процесу виконання та оцінювання лабораторних та практичних робіт.

Запропоновано використати стек технологій безперервної інтеграції та розгортання, а також застосунок для перегляду статистики та порівняння разом з реляційною СУБД.

Розроблено застосунок для перегляду статистики та порівняння робіт студентів.

Розроблено програмне забезпечення та скріпти налаштування конвейерів CI/CD.

Практичне значення роботи полягає у створенні програмного забезпечення для автоматизації процесу оцінювання та виконання лабораторних та практичних робіт студентів з програмування.

Результати здійснених у дипломному проекті досліджень можуть бути використані на першому етапі реалізації системи автоматизації перевірки лабораторних робіт факультету інформаційних технологій, а також у інших системах автоматизації первірки різних факультетів, де викладається програмування та потрібна автоматизована перевірка робіт студентів.

Галузь використання – програмування, первірка лабораторних робіт, оцінювання робіт студентів.

Напрямки подальшого розвитку роботи включають інтеграцію розробленого застосунку та програмного забезпечення і скриптів налаштування конвейерів CI/CD, а також тестування розробленої системи автоматизації та її компонентів у процесі використання.

Ключові слова: ІНФОРМАЦІЙНА СИСТЕМА, АВТОМАТИЗАЦІЯ, ОЦІНЮВАННЯ ЛАБОРАТОРНИХ РОБІТ, ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ, БЕЗПЕРЕРВНА ІНТЕГРАЦІЯ, БЕЗПЕРЕРВНЕ РОЗГОРТАННЯ, БАЗА ДАНИХ, РЕЛЯЦІЙНА БАЗА ДАНИХ, СИСТЕМА АВТОМАТИЗАЦІЇ, ТЕХНОЛОГІЇ АВТОМАТИЗАЦІЇ

ЗМІСТ

Стор.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CI – Continuous Integration (Безперервна інтеграція).

CD – Continuous Deployment (Безперевне розгортання).

ІС – інформаційна система.

ОС – операційна система.

БД – база даних.

СУБД – система управління базами даними.

API – Application Programing Interface (Прикладний програмний інтерфейс).

JDK – Java Development Kit (набір інструментів Java для розробки).

JMH – Java Microbenchmark Harness (інструменти вимірювання продуктивності).

VM – віртуальна машина.

JVM - Java Virtual Machine (віртуальна машина виконання коду Java).

ВСТУП

Методології та практики розробки програмного забезпеченння розвиваються дуже швидким темпом і підвищують якість та швидкість отримання результату від команд програмної інженерії та зменшують затримку для отримання відгуку від замовника. Сьогодні все більше команд розробки ПЗ ставлять за мету створювати більш якісні застосунки та з використанням різних технологій автоматизації. Не виключенням стали і студенти інформаційних факультетів де програмування є невідємною частиною процесу навчання та здобування нових навичок. А тому в нагоді стали технології безпервної інтеграції та розгортання, які покликані пришвидшити процес перевірки.

Автоматизація стрімко проникає у всі сфери людської діяльності, а разом с тим в організацію навчального процесу та перевірку робіт студентів з програмування. Для пришвидшення оцінювання лабораторних та практичних робіт викладач або лаборант повинен за допомогою комп'ютера та розобленого застосунку аналізувати статистику по роботам студентів та автоматизовано порівнювати їх, щоб викрити факт недобросовісного виконання роботи різними студентами. А вже далі на основі цих даних приймати рішення по оцінці.

Метою дипломної роботи є дослідження підходів автоматизації за допомогою технологій CI/CD в предметній області та оцінка продуктивності ковейерів доставки коду CI/CD Workflow. Також метою даної роботи є розробка та дослідження тестового проекту, який використовує технології безперевної інтеграції та доставки CI/CD. Цей проект покликаний автоматизувати певну діяльність викладачів факультету.

Система автоматизації повинна дозволяти перегляд статистики по резпозиторям коду студентів та порівняння коду з іншими студентами групи.

Актуальність теми даної роботи обумовлена тим, що в сучасних умовах перевірка та оцінювання лабораторних та практичних робіт студентів займає багато часу, який можна було б присвятити будь-якій іншій корисній діяльності працівників факультету інформаційних технологій. У цій роботі буде зроблене дослідження використання технологій безперервної інтеграції та безперервної

доставки в загальному контексті та в контексті даної предметної області. Також буде розроблено тестовий застосунок, який дає можливість належно оцінювати роботи студентів та перевіряти їхню оригінальність серед інших робіт студентів групи.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПІДХОДІВ ТА ІНСТРУМЕНТІВ ДЛЯ РЕАЛІЗАЦІЇ CI/CD В ПРЕДМЕТНІЙ ОБЛАСТІ

1.1. Еволюція та шляхи розвитку CI/CD

За останні роки безперервна інтеграція CI та доставка CD стала однією з найбільш ефективних практик в сфері розробки програмного забезпечення. Її основна суть полягає в дуже частому виконанні оновлень програмних додатків. Тобто в той момент як тільки з'являється нова версія коду в репозиторії розробників, всі зміни повинні бути інтегровані. Безперервна доставка CD забезпечує успіх безперервної інтеграції CI та робить розгортання програмного забезпечення значно легшим в майже будь-яке середовище за допомогою одного кліку, або повністю автоматизовано в випадку безперевного розгортання. Незважаючи на те, що це колись звучало як недосяжна ціль, прориви в галузі автоматизації роблять можливим дуже часте розгортання релізів. Ідею безперервної доставки вийнашов і поширив Мартін Фаулер, головний науковий співробітник ThoughtWorks та дуже досвідчений розробник. Він описав передумови практики таким чином [1]: "Кожен розробник часто інтегрує свої зміни зі змінами коду усієї команди. І якщо трапилась якась помилка, ніхто в команді не має важливішої роботи, ніж виправлення цієї помилки". Варто згадати, що кожна інтеграція перевіряється автоматизованою збіркою та тестуванням, щоб виявити помилки інтеграції якомога швидше. Такий підхід дійсно призводить до значного зменшення проблем інтеграції та дозволяє розробникам швидше доставляти покращення ПЗ.

Взагалі, існує три основних підходи до розробки програмного забезпечення: водопад(Waterfall), гнучка методологія(Agile) і CI/CD. Для першої усі етапи розробки від планування до розгортання та обслуговування йдуть послідовно. Водопад довів свою неефективність для продуктів, де потрібні постійні оновлення. Ця особливість методології не дозволяла швидко реагувати на відгуки клієнтів через тривалі цикли розробки. Сьогодні водопад в основному використовується для

коротких проектів із фіксованою вартістю. На рис 1.1 представлені загальньо прийняті кроки методології водопаду.

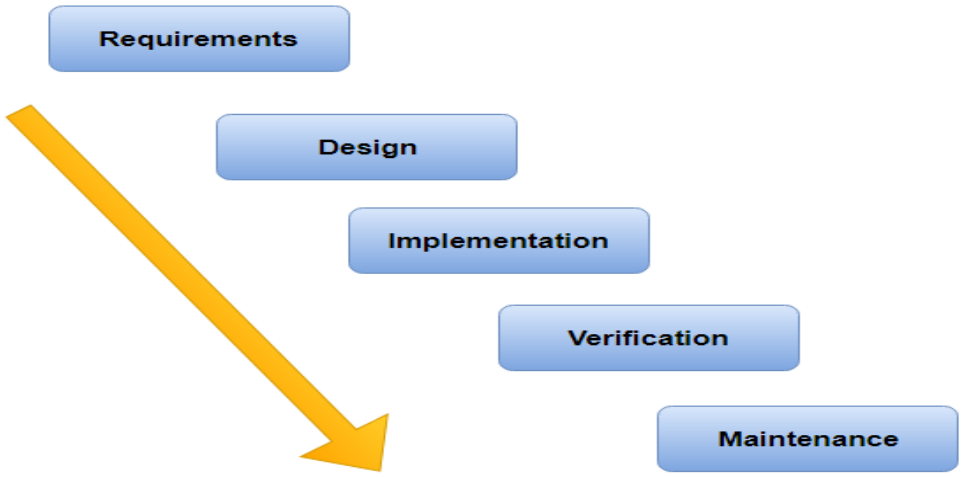


Рисунок 1.1 – Кроки методології водопад

Першим етапом є збір вимог та документація. На данному етапі команда повинна зібрати вичерпну інформацію про те, що вимагає проект замовника. До кінця цього етапу вимоги до проекту повинні бути чітко встановлені, і повинен бути оформлений документ. На другому етапі відбувається встановлення специфікацій, такі як мова програмування та вимоги до обладнання. На третьому етапі відбувається програмування. Програмісти беруть інформацію з попереднього етапу і створюють функціональний продукт. Після того, як все кодування завершено, розпочинається випробовування продукту. Тестувальники систематично знаходять і повідомляють про будь-які проблеми. Якщо виникають серйозні проблеми, то можливо команді доведеться повернутися до першого етапу. На етапі доставки та розгортання продукт завершено, і команда подає результати замовнику в вигляді готових артефактів, які в подальшому потрібно розгорнути. По мірі виникнення проблем команді може знадобитися створити негайне виправлення та оновлення, щоб вирішити їх. Знову ж таки, серйозні проблеми можуть вимагати повернення до першого етапу [3].

На відмніну від моделі водопаду гнучкий підхід визначається ітеративним SDLC і передбачає короткі цикли, які включають усі основні етапи: планування, саму розробку, тестування та розгортання. Кожен цикл займає десь один-два тижні.

Ідея Agile полягає у якомога швидшому розгортанні продукту та поступовому оновленні його на основі відгуків замовників. Agile підхід залишається досить поширеним, оскільки підтримує адаптивність продукту до постійно мінливих потреб ринку та споживачів. На рис 1.2 представлені загальньо прийняті кроки гнучкої методології.

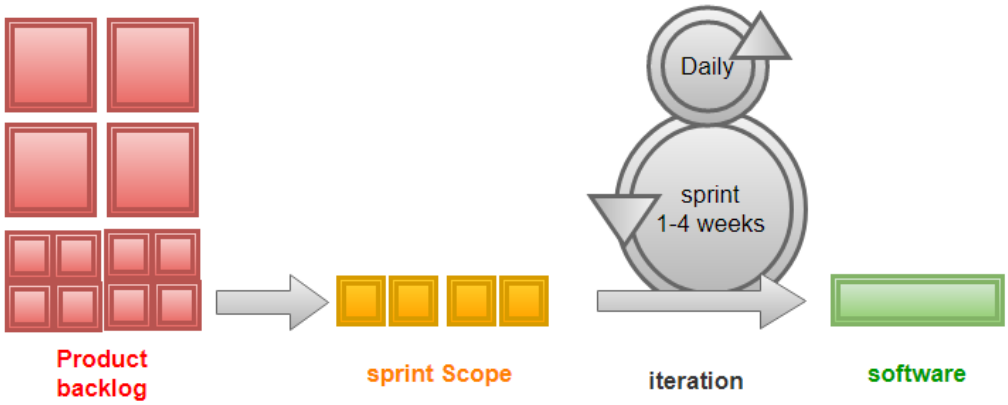


Рисунок 1.2 – Кроки гнучкої методології

В рамках гнучкого SDLC робота поділяється на спрінти, з метою створення або оновлення чи виправлення проблем робочого продукту в кінці кожного спрінту. Спрінт зазвичай триває два тижні або 10 робочих днів. Робочий процес спрінту повинен дотримуватися основного плану:

- планування;
- розробка;
- тестування;
- доставка;
- відгук та оцінка замовника.

Всі три підходи використовуються в розробці програмного забезпечення. Але оскільки методи та засоби з часом вдосконалювались, можна вважати, що ці три підходи еволюціонували в CI/CD. На рис 1.3 представлені загальньо прийняті кроки та їхня інтеграція в методології CI/CD.

Рисунок 1.3 – Кроки та їх інтеграція в методології CI/CD

Етап планування часто поєднує практики Agile, щоб забезпечити часті мікроінкрементальні релізи. Етап програмування зосереджується на основних завданнях розробки в IDE та тестових середовищах для коду застосунка та інфраструктурного коду. Етап побудови зливає воєдино всі зміни зроблені різними розробниками. Тестовий етап фокусується на автоматизованій перевірці удосконалень, часто включаючи тестові практики розгортання на клоні виробничого середовища. Етап випуску зосереджений навколо документації змін та фактичного оновлення кодової бази. Етап експлуатації відбувається після того, як код опублікований, і складається з оркестрації та подальшого моніторингу, що є наступним етапом цієї методології, але в даному контексті може розглядатись як частина етапу експлуатації.

Основними перевагами цієї методології є:

- максимальна автоматизація;
- висока якість;
- низький ризик;
- простіші зміни коду(більш атомарні);
- менші витрати на інфраструктуру;
- безпека;
- кілька середовищ тестування та розгортання.

Якби безперервну інтеграцію, доставку та розгортання можна було коротко описати одним словом, це була б автоматизація. Ця практика стосується автоматизації процесу тестування та розгортання, зведення до мінімуму (або

повного усунення) потреби в людському втручанні, мінімізації ризику помилок та спрощення побудови та розгортання програмного забезпечення.

Якщо розглянути всі інші переваги, то можна виділити також високу якість та часті атомарні зміни коду розроблюваного продукту, а також низький ризик, так як легше виявити дефекти та інші проблеми якості програмного забезпечення на менших диференціалах коду та недопустити потрапляння дефектів на наступні стадії. Команди розробників, які практикують безперервну інтеграцію, використовують різні методи, щоб контролювати, які функції та код готові до виробничого середовища. Відповідно до [3], тестування продуктивності та безпеки, часто інтегруються в цю практику і виконується після доставки збірок до не виробничого середовища, а в різні середовища для тестування нових змін.

1.2. Опис та аналіз предметної області

Першим етапом опису предметної області – є системний аналіз і опис інформаційних технологій та компонентів, які передбачено будуть використані для реалізації проекту даної предметної області. Також на першому етапі визначається відношення між різними суб’єктами інформаційної системи. Наприклад визначаються зв’язки між різними компонентами ІС та користувачами.

Предметна область включає автоматизацію певної діяльності факультету інформаційних технологій. А саме автоматизація оціннювання та виконання лабораторних та практичих робіт, також відображення загальних відомостей та статистики виконання робіт, які потрібні для різних суб’єктів установи, для якої проектується інформаційна система. Студенти виконують лабораторні та практичні роботи, зберігають код в репозиторіях системи контролю версій, відправляють посилання на репозиторій, а викладачі в свою чергу викликають студентів на захист. Під час захисту студент демонструє локальне розгортання та відпрацювання коду і в подальшому отримує оцінку. Ці стадії можуть бути автоматизовані.

Викладачі факультету, зацікавлені в швидкому отриманні інформації про репозиторії коду студентів та статистики, могли б звертатися до інформаційної

системи, яка містить усю відповідну інформацію. Також викладачам необов'язково викликати на захист студентів та просити їх розгорнути та запустити їхній код, ця стадія може бути автоматизована за допомогою технологій CI/CD, таких як авто-збірка, виконанання автотестів написаних самим студентом, але попередньо погоджених з викладачем. Також клонування коду репозиторію та його розгортання може бути автоматизоване. Тобто в момент коли викладачу потрібно перевірити роботи студентів, він може продивитися статистику по репозиторіям, а також запустити виконання збірки, автотестів та розгортання коду в спеціальне середовище в себе на комп’ютері чи на спеціальному сервері. Для цього може бути розроблений додаток, який звертається в інформаційну систему, яка являє собою реляційну базу даних та зберігає інформацію про репозиторії студентів, а також оцінки по завданням та оновлену статистику по комітам в репозиторіях. Сам додаток дає можливість переглянути результати виконання збірок та автотестів, а також розгортання в спеціальному середовищі, такому як віртуальна машина чи орхестратор контейнерів, за допомогою технологій CI/CD [2]. Тобто в додатку має бути веб-інтерфейс, який дозволяє переглядати результати цих стадії для кожного репозиторію. В цілому це значно полегшує та автоматизує процес завантаження коду та його подальші етапи до тестового розгортання. Також в веб-інтерфейсі повинна бути можливість проставити оцінки після швидкого запуску авто-збірки, авто-тестування та авто-розгортання коду.

Отже, основними суб’єктами для яких розробляється ІС та застосунок є:

- викладачі;
- лаборанти;
- студенти.

Основними компонентами даної ІС є:

- застосунок для перегляду статиситики по репозиторіям та порівняння, а також керування застосунком для запуску;
- застосунок для запуску авто-збірки, авто-тестів та авто-розгортання;
- база-даних для зберігання інформації по завданням та зберігання акутальної статистики.

Ці всі компоненти повинні бути зв’язані між собою. Тобто застосунок для перегляду статистики повинен представляти єдиний веб-інтерфейс де є можливість переглядати всі репозиторії в одному місці і статистику по ним, ще в ньому має бути можливість виставляти оцінки для кожного окремого студента по репозиторію. Також він має звертатися до БД для зберігання інформації. В доповнення додаток повинен дати можливість керувати додатком для запуску стадій CI/CD. Застосунок для запуску має бути своєрідним контролером, який виконує різні команди для різних технологій автоматизації на серверах CI/CD. На рис 1.4 представлена архітектурна діаграма першого рівня для даної ІС.

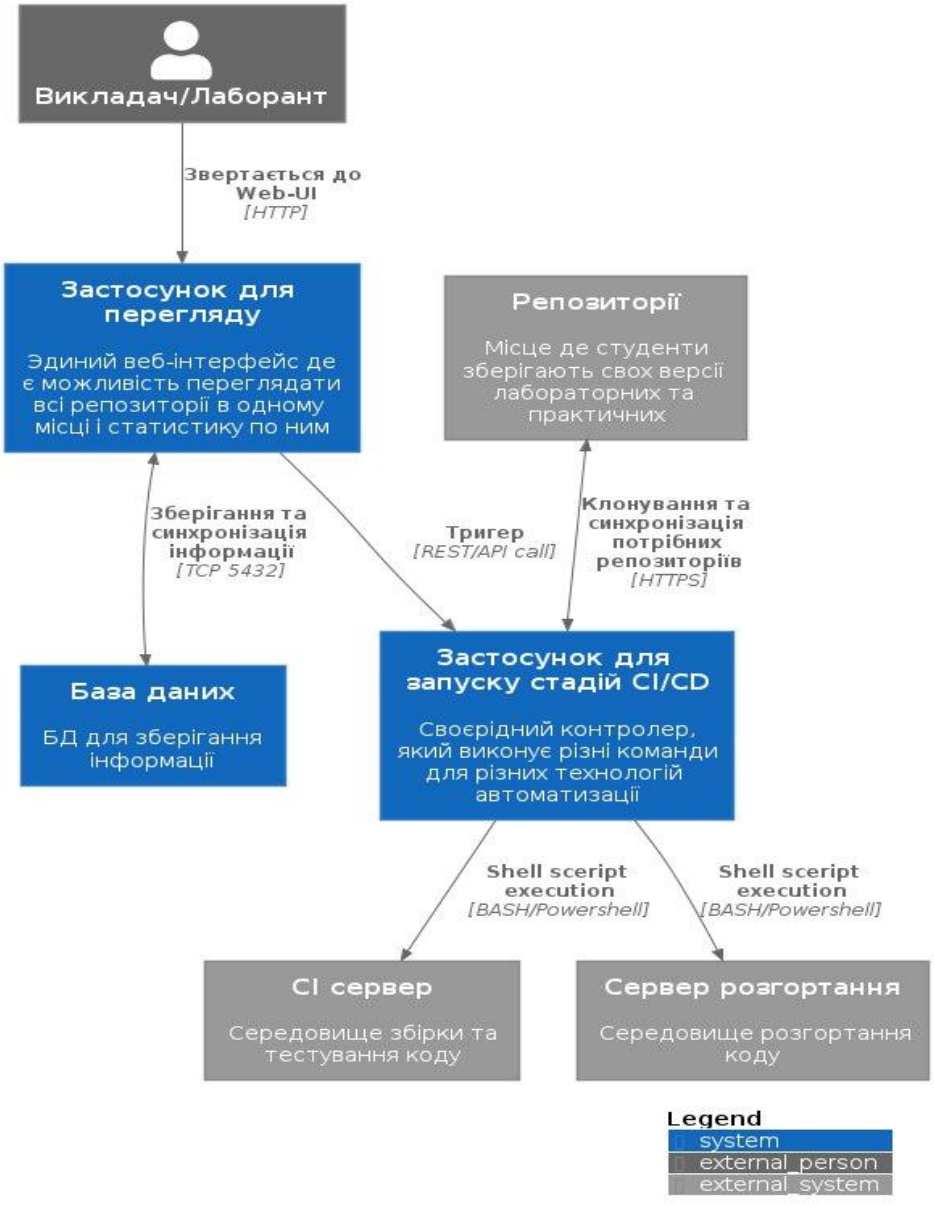


Рисунок 1.4 – Перший рівень деталізації архітектури ІС

Продовжуючи перший етап аналізу предметної області доцільно детально описати розроблюваний функціонал та головні особливості кожного компоненту даної ІС та додатків. Так як це є етапом опису базової моделі, то на ньому потрібно вирішити питання зі зберіганням та структурування інформації. Перший компонент з якого було б краще всього почати це база даних для зберігання інформаціїї в ІС. Далі буде наведено концептуальну модель зберігання інформації в БД. Створення концептуальної моделі складається з кількох етапів:

- визначення сутностей;
- визначення атрибутів;
- визначення унікальних ідентифікаторів;
- визначення відношень та їх властивостей.

Концептуальна модель складається з наступних сутностей:

1. Студенти – Завдання: тип відношень – багато до багатьох. Кожен студент має зробити багато завдань, а також багато завдань мають бути присвоєні до багатьох студентів. Роль проміжної таблиці в даному відношенні виконує таблиця “репозиторії”. На рис 1.5 ER-модель, яка показує відношення між сутностями «Студенти» і «Завдання».

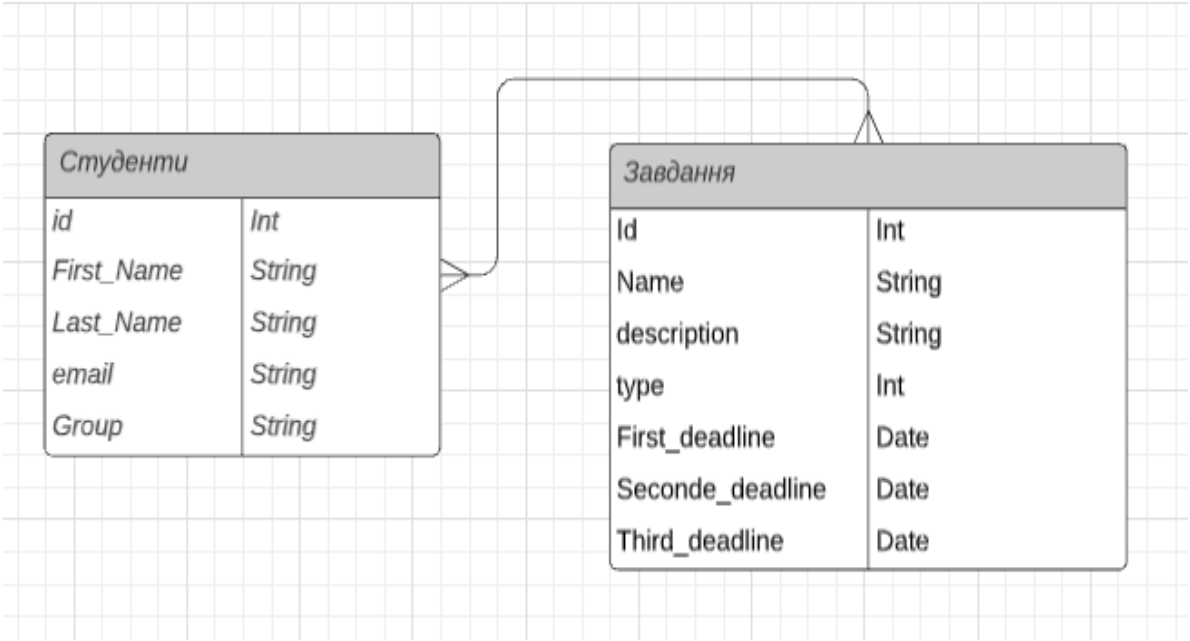


Рисунок 1.5 – ER-модель зв’язку сутностней «Студенти» і «Завдання»

2. Студенти – Репозиторії: тип відношень – один до багатьох. Студент створює репозиторій під кожне завдання. На рис 1.6 ER-модель, яка показує відношення між сутностями «Студенти» і «Репозиторії».

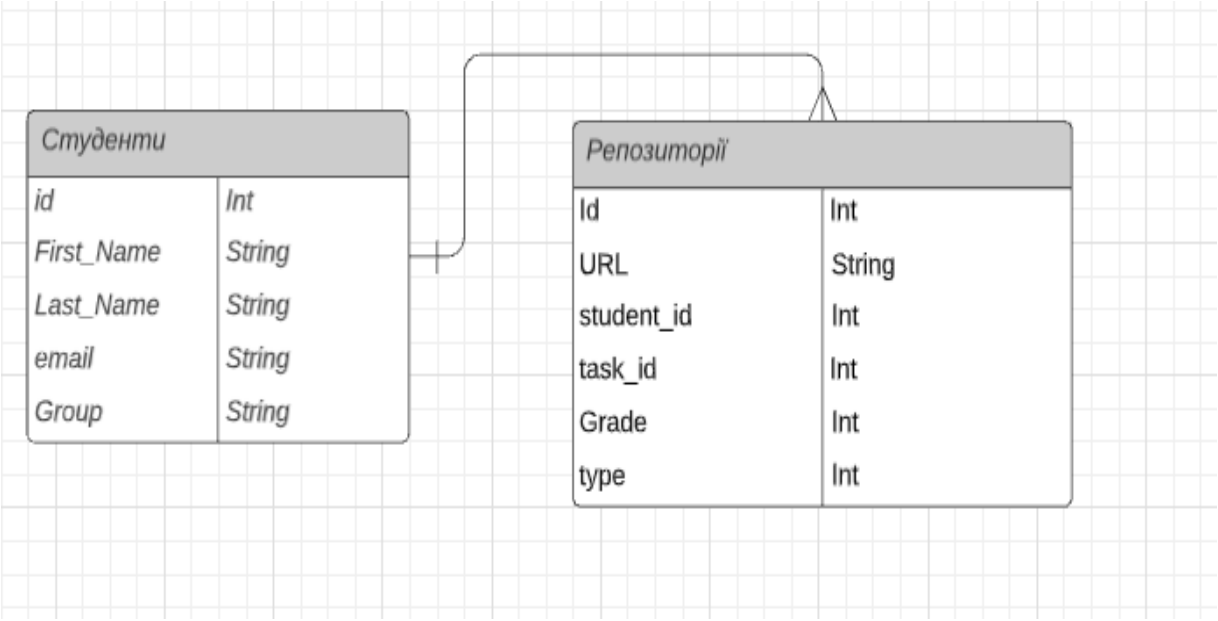


Рисунок 1.6 – ER-модель зв’язку сутностней «Студенти» і «Репозиторії»

3. Завдання – Репозиторії: тип відношень – один до багатьох. Кожне завдання може бути зроблено в кількох репозиторіях та багатьма студентами, кожен з них виділи під завдання окремий репозиторій. На рис 1.7 ER-модель, яка показує відношення між сутностями «Завдання» і «Репозиторії».

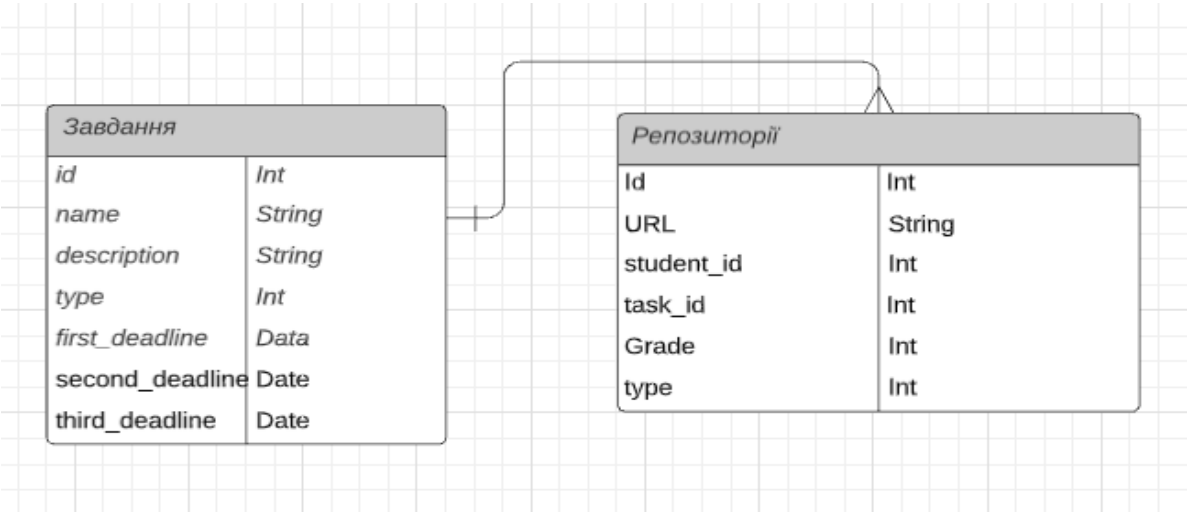


Рисунок 1.7 – ER-модель зв’язку сутностней «Завдання» і «Репозиторії»

Наступну проблему, яку потрібно вирішити це перегляд та зміна інформації про студентські репозиторії та самих студентів. Для цього потрібно розробити додаток, який дозволяє заносити та редагувати данні. Завантаження даних повинне відбуватися, як вручну так і автоматизовано з файлу excel чи csv. Тобто в ці файли попередньо була занесена інформація викладачами та студентами, а застосунок повинен дозволяти завантажувати вміст даних файлів до бази данних для подальшої маніпуляції, також додатковою функцією даного застосунку має бути пергляд статистики по коміттам та порівняння коду студентів з іншими репозиторіями інших студентів. Також цей застосунок повинен відправляти запити до контролера запуску стадій CI/CD. Це потрібно для автоматичної перевірки збірки, тестування та розгортання. Другий компонт даної ІС це застосунок для перегляду та маніпулювання даними про студентів та репозиторії

Аналізуючи концептуальну модель даних, цей застосунок повинен мати як мінімум 3 контролери, 3 представлення та 3 сутності моделей. Тобто по кожному контролеру на кожну модель даних, які передбачають різні методи. Представлень може бути більше, але мінімум по кожному на кожен контролер. Кожен контролер повинен передбачати кілька методів, які реалізовують функціональність CRUD, а також методи для зв’язку з іншим застосунком для запуску стадій CI/CD. Тобто в данному контексті розглядаються наступні сутності моделей:

- class students;
- class repos;
- class tasks.

Для цих сутностей можна виділити мінімальний набір контролерів, які роблять можливими CRUD операції:

- students controller;
- repos controller;
- tasks controller.

Також для цих контролерів та моделей можна виокремити мінімальний набір представлень:

- students templates;

- repos templates;
- tasks templates.

В процесі розробки може виникнути необхідність додавати нові контролери та представлення. Але моделі залишатимуться незмінними. В цілому такий підхід дозволяє створювати нові контролери та представлення за потреби. У подальшому це може знадобитися для інтеграції з іншими компонентами даної ІС.

І останім компонентом даної ІС є застосунок запуску CI/CD стадій. Він представляє собою консольний додаток, який при спрацьовуванні тригеру повинен запускати стадії CI та розгортати код на сервер чи локально. Цей компонент є своєрідним контролером, який запускає різні скріпти, які потрібні для різних технологій автоматизації та зберігання коду. Прикладами таких технологій з якими взаємодіє контролер:

- системи контролю версій(віддаленні репозиторії);
- сервер безпервної інтеграції(CI сервер);
- сервер для розгортання.

За допомогою VCS можна легко відстежувати історію змін файлів і коду, а також робити порівняння різних гілок та репозиторіїв. Можна виділити кілька поколінь таких систем.

Наступним компонентом даного проекту є сервер CI. За допомогою серверу безперервної інтеграції можна легко зробити автоматизацію збірки, тестів. Сервер безперервної інтеграції (іноді його називають сервером збірки), а також він виконує функцію рецензента для коду. Коли розробники здійснюють оновлення віддаленої гілки локальними комітами, сервер CI ініціює збірку та виводить логи, тобто документує результати збірки [6].

Віртуальні машини та контейнери також можна використовувати разом в контексті даного проекту. В подальших пунктах даного розділу буде описана можливість використовувати контейнери для виконання стадій CI в таких дитсрибутивах контролю версій як Gitlab та GitHub Actions.

1.3. Вибір технологій та інструментів автоматизації

Першим компонент ДП для якого потрібно вибрати оптимальну технологію є база даних. Так як в ній зберігаються представлені дані про студентів та їхні репозиторії, а також завдяння з описом.

Бази даних – це інструменти для організації даних у різні структури та подальшого зберігання, отримання. Наразі існує величезна кількість імплементацій технологій, які завдяки своїм перевагам та недолікам, можуть бути використані для різних цілей:

- зберігання складних транзакцій;
- легкого подальшого масштабування та реплікації;
- створення гнучкої схеми даних та типів;
- обробки багатьох неперервних запитів для читання та запису;
- оперування широким спектром шаблонів доступу та типів даних;
- завдань постійної автономної звітності, логування;
- реалізації цілісності;
- зберігання та кешування BLOB.

Відповідно до [13], наразі існує два популярних типи баз даних: NoSQL та SQL. Для деяких цілей, зазначених вище, підходить тільки один тип. Для першої цілі добре підходить другий тип баз даних, оскільки в даному випадку неможливо дозволити втратити дані або пошкодити їхню цілісність. Саме для реалізації цілісності даних призначені реляційні БД, але тут втрачається гнучкість, тобто неможливо створити запис з різними типами даних полів та їхньою кількістю в одній таблиці, але в NoSQL це можливо. Для масштабування та реплікації можуть бути вибрані два типи БД, тобто саме для підтримки балансування навантаження, автоматичного sharding, а також для забезпечення відмовостійкості. Четверта ціль – випадок, коли дані можуть бути мінливими, саме для цього випадку потрібно розглянути документно-орієнтовну БД, чи ключ значення БД. Також у випадку обробки багатьох неперервних запитів, потрібно розглядати БД, які пропонують швидкий доступ до пам’яті, також SSD, коли є потреба замовляти в хмарних

провайдерів. Щоб оперувати широким розмаїттям шаблонів доступу та типів даних, потрібно використовувати документно-орієнтовану, вона, як правило, гнучка та продуктивна в порівнянні з іншими типами. Отже, існує кілька типів баз даних для різних цілей, однак, як правило, SQL є найбільш відомим типом та перевіреним, тобто може бути використаний в різних фінансових установах та там де необхідна точність та цілісність транзакцій. MongoDB або будь-яка інша БД не підходить, але там де потрібно працювати з мінливими даними, все ж таки потрібно дивитися в бік NoSQL. Ще більш специфічною ціллю є підтримка різних шаблонів запису та читання. Існують системи, оптимізовані для 50% читання, 50% записів, 95% записів або 95% читань.

Створення бази даних - це процес, який є послідовністю переходів від неформального опису предметної області до створення моделей різного рівня. В попередньому розділі було визначено базову схему, яка включає в себе 3 сутності, а також було визначено зв'язки між ними. В цьому розділі буде визначено різновид СУБД, який краще всього підходить для реалізації проекту, а також конкретний дистрибутив та різні технології оптимізації

Порівнюючи бази даних SQL та NoSQL, бази даних SQL є базами даних на основі таблиць, тоді як бази даних NoSQL є базами даних на основі документів, тобто пари ключ-значення. Бази даних SQL є вертикально масштабованими, тоді як бази даних NoSQL - горизонтально. Бази даних SQL мають заздалегідь визначену схему, тоді як бази даних NoSQL використовують динамічну схему для неструктурованих даних.

У випадку, коли потрібно організувати базу даних для інформаційної системи студентів та репозиторіїв, згідно з аналізу двох типів, гарантовано підходить реляційна схема.

Для цього ДП було вибрано PostgreSQL, як основну технологію. Вона забезпечує відповідність всім належним вимогам та добре підходить під дану ІС. Але ключовими перевагами є безкоштовність, та велика кількість інтерфейсів користувачів, тобто це великі можливості керування за невеликий бюджет.

Наступним компонентом для реалізації ДП є застосунок, який потрібен для маніпулюванням даними, які зберігаються в БД, а також для перегляду статистики по репозиторіям системи контролю версій. Для даних цілей добре б підійшов фреймворк, який має функціональність REST-API для взаємодії з іншими застосунками, а також який має вбудований функціонал чи можливість підключити різні бібліотеки для роботи з API систем контролю версій. В даному проекті використовується система контролю версій GIT. Студенти зазвичай використовують дистрибутив Github, так як він безкоштовний. Але також на момент 2021 року в цьому дистрибутиві з'явився аналог CI сервера, який має назву GitHub Actions. Він також має різні способи взаємодії через API. В цьому пункті буде описана його функціональність. Як зазначено у [8], для цілей даного проекту добре підходить фреймворк Flask на мові програмування Python, так як він має наступні переваги:

- підтримка GitHub API;
- підтримка Rest Full;
- сумісність с контейнерезацією;
- простота підтримки;
- простота розробки;
- модульність.

Основною перевагою Flask є те що він прагне зберегти ядро своєї основи легким, але розширюваним. Це робить написання програм або розширень гнучким, і надає розробникам можливість вибирати різні модульні конфігурації, які вони хочуть для свого додатку, не накладаючи жодних обмежень на вибір бази даних, механізму шаблонування.

Наступний компонент для реалізації даного проекту є CI сервер. Для його реалізації найкраще підходять дві технології GitHub Actions та Gitlab. Дії GitLab CI / CD та GitHub дозволяють створювати робочі процеси, які автоматично створюють, тестують, публікують, випускають та розгортають код. GitLab CI / CD та GitHub Action мають подібні особливості конфігурації.

У GitLab CI / CD процеси налаштовуються за допомогою інтерфейсу користувача, тоді як у GitHub Action можна запускати робочий процес із запланованим інтервалом. Але так як переважна більшість студентів використовують github, то перевагу було віддано GitHub Actions. Його переваги:

- простота конфігурації;
- логування;
- автоматизація CI/CD;
- підтримка Github API.

Висновки до розділу 1

У цьому розділі було розглянуто різні стадії життєвого циклу ПЗ, визначено переваги та недоліки використання тих чи інших методогій розробки ПЗ, а також було наведено опис предметної області, яка потребує автоматизації. Було визначено основні етапи в які є можливість інтегрувати автоматизацію. Також були визначені основні вимоги для впроваждення певною мірою CI/CD. Дана інформаційна система буде орієнтована на викладачів та лаборантів різних факультетів, які мають дисципліни пов’язані з розробкою ПЗ.

Автоматизація оцінювання та виконання лабораторних та практичних робіт за допомогою технологій CI/CD дозволить покращити якість перевірки завдань та швидкість отримання результатів по перевірці. Також дана розроблювана система дозволить отримувати статистичні данні по різним репозиторіям різних студентів в єдиному інтерфейсі.

Узагальнючи, у даному розділі було проведено аналіз предметної області, визначено основні моделі та бізнес логіку, а також був проведений поверхневий опис архітектури розроблюваної системи. Це дозволить систематизувати всю необхідну інформацію, яка потрібна для створення інформаційної системи та закласти базові та фундаментальні знання, які знадобляться для написання інших розділів. Наступний розділ буде присвячений більш детальнішому проектуванню

системи, а також наведенню прикладів застосування вибраних технологій для реалізації CI/CD в даній предметній області.

РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ АВТОМАТИЗАЦІЇ НА ОСНОВІ ТЕХНОЛОГІЙ CI/CD В ПРЕДМЕТНІЙ ОБЛАСТІ

2.1. Проектування бази даних для зберігання інформації про студентів та репозиторії

У попередньому розділі вже було наведено концептуальну модель зберігання даних в БД, що розробляється, а також було з’ясовано мету використання БД для даного проекту. У цьому розділі буде зроблено більш детальніше проектування цього компоненту для вирішення загальної проблеми створення системи автоматизації CI/CD. Адже добре спроєктована БД належним чином забезпечує:

- економію місця для зберігання;
- підтримку точності та цілісності даних;
- надання швидкого доступу до даних;
- ефективне виконання запитів;
- високопродуктивну взаємодію з додатками.

В контексті реалізації даного проекту ці та інші особливості можуть стати в нагоді під час подальшої підтримки даної системи. Витрати часу на планування бази даних економить можливий витрачений час на раптові неприємності під час розробки, а добре розроблена база даних забезпечує легкий пошук інформації. Безумовно ця БД потрібна для зберігання даних про студентів та репозиторії, а також про завдання та проекти, які передбачені навчальною програмою факультета. А отже тут неможливо обійтись без чіткого планування та проектування даного компоненту.

Подальше проектування схеми БД включає створення наступних моделей:

- створення концептуальної схеми;
- створення логічної схеми;
- створення, чи отримання за допомогою конвертації, фізичної схеми.

Так як концептуальна модель була зроблена в попередньому розділі, то залишилось зробити логічну та фізичну модель. Наступним етапом є створення логічної моделі. Моделювання логічної схеми допомагає аналізувати вимоги до

даних, щоб створювати добре розроблені бази даних. Це модель, яка є тільки в інструментарії для проектування. Іншими словами це ніщо інше, як план для архітектора БД. Подібним чином, як до концепутальної моделі так і для логічної моделі, визначаються атрибути сутностей. Нижче буде приведено логічну схему таблиць даної БД.

Таблиця 2.1 – Атрибути сутності «Студенти»

Атрибут	**Опис**
ID	Унікальний номер екземпляру сутності.
First_name	Ім’я студента.
Last_name	Прізвище студента.
Email	Електронна пошта студента.
Group	Група в якій знаходиться студент.

Таблиця 2.2 – Атрибути сутності «Завдання»

Атрибут	**Опис**
ID	Унікальний номер екземпляру сутності.
Name	Назва завдання.
Description	Опис завдання.
Type	Тип завдання. Визначаєть тип завдання проект, лабораторна, практичне завдання.
First_deadline	Перший термін здачі.
Second_deadline	Другий термін здачі.
Third_deadline	Третій термін здачі.

Таблиця 2.3 – Атрибути сутності «Репозиторії»

Атрибут	Опис
ID	Унікальний номер екземпляру сутності.
URL	Посилання на веб-ресурс, що вказує місцезнаходження репозиторія студента.
Ідентифікатор студента	Зовнішній ключ, який вказує на екзмепляр сутності студента.
Ідентифікатор завдання	Зовнішній ключ, який вказує на екзмепляр сутності завдання.
Оцінка	Відмітка, яка виставляється по завршенню перевірки роботи студента.
Тип репозиторію	Тип проекту, який визначається структурою Maven, Gradle, Java condole app.

Отримавши всі атрибути сутностей бази даних, можна застосувати правила нормалізації, щоб переконатися, що таблиці правильно структуровані. Також в майбутньому слід налаштувати базу даних для перевірки даних відповідно до базових правил створення схеми в PostgreSQL. Багато систем управління базами даних застосовують деякі з цих правил автоматично. Правило цілісності сутності стверджує, що первинний ключ ніколи не може мати значення NULL. Якщо ключ складається з декількох стовпців, жоден з них не може мати значення NULL. Інакше не вдасться однозначно ідентифікувати запис. Також він має бути цілочисельним типом даних, а не будь-яким іншим. Його назва повинна складатися з назви вторичної таблиці, а також з префіксу ідентифікатора. На рис 2.1 зображена логічна модель БД для зберігання даних про студентів та репозиторії.

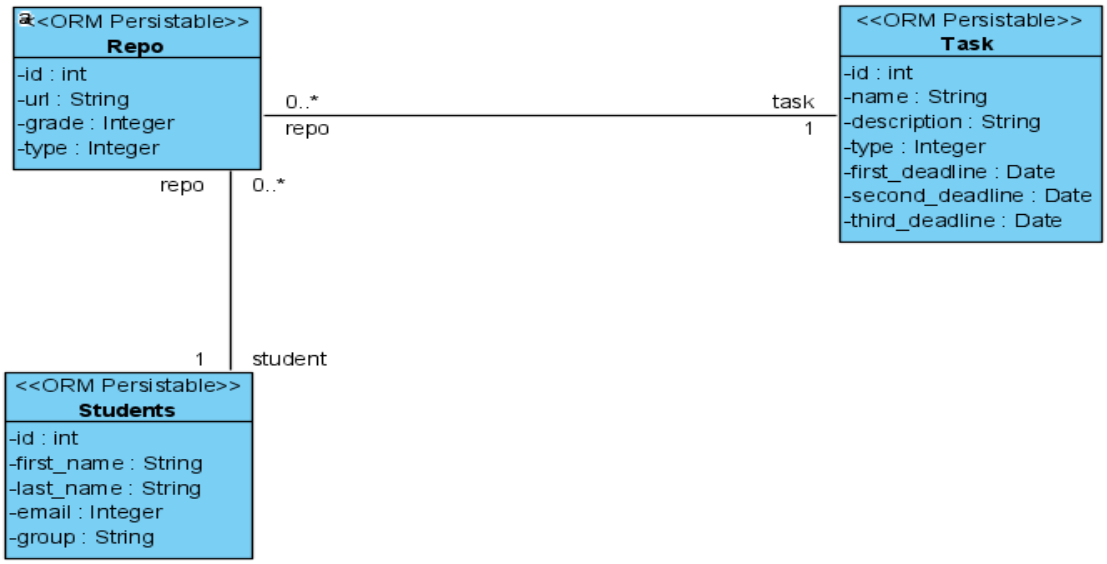


Рисунок 2.1 – Логічна модель БД

Але є ще один різновид представлення БД, який пов’язаний із конкретною реалізацією. Це фізична модель – те, що насправді реалізовано в СУБД, яка враховує конкретну технологію баз даних. Вона визначає точні типи даних атрибутів, які наявні в конкретних СУБД [14]. На рис 2.2 зображена фізична модель БД даного проекту.

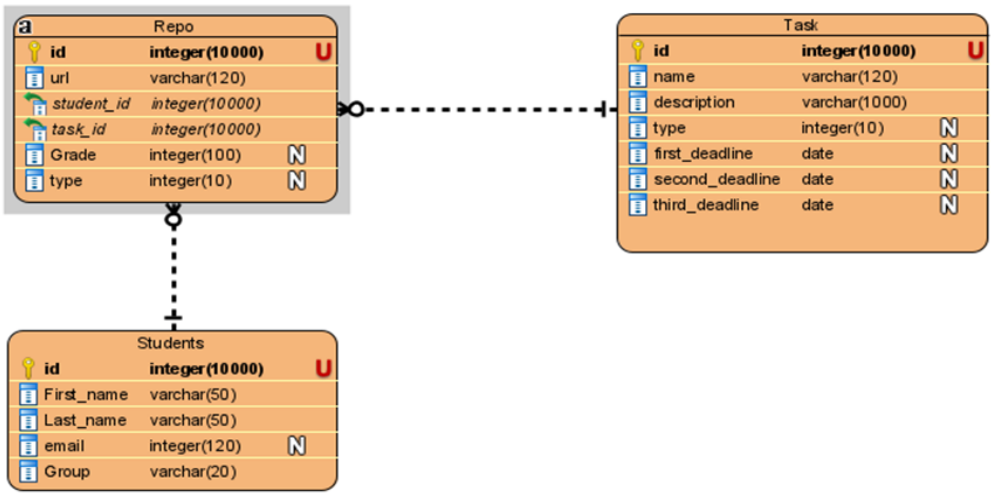


Рисунок 2.2 – Фізична модель БД

2.2. Проектування застосунку для перегляду статистики та оцінювання лабораторних та практичних робіт

У цьому пункті даного розділу буде представлено інформацію про те, як дані передаються між веб-інтерфейсом користувача та бекендом даного застосунку. Також буде представлено технічні подробиці про формат JSON, який є основою для REST-API застосунка на Flask. Тобто в цьому пункті розділу буде спроектовано дизайн RESTful API для бекенду. Ця платформа дозволить лаборантам та викладчам додавати інформацію про студентів. У той же час вони також зможуть дивитися статистику по комітам в кожному репозиторії і швидко та автоматично завантажувати проект і відкривати його в IDE.

API в загальному розумінні означає інтерфейс прикладного програмування, тобто це інтерфейс для веб-додатку або мобільного додатку для зв'язку з бізнес-логікою сервера. Це також стандартний інтерфейс, який може взаємодіяти з різними типами кількох бекенд застосунків, це можуть бути мобільні програми або веб-сайти. Простіше кажучи, це як єдиний веб-інтерфейс, який приймає запит від користувачів чи інших застосунків і відправляє запит до серверної системи. Однією з переваг використання API, яку варто згадати, є інкапсуляція, вона потрібна для захисту від несанкційованого доступу до інформації, щоб неавторизовані суб'єкти зовні не змогли її побачити. За допомогою цього принципу великі компанії з конфіденційною інформацією готові надавати послуги через API, впевнені, що їх внутрішня інформація не буде доступною.

REST-API розшифровується як передача стану представлення. Вперше цей термін був визначений в дисертації доктора Роя Філдінга (Архітектурні стилі та дизайн мережевих архітектур програмного забезпечення) ще в 2000-ому році. Ця дисертація вважається основоположним документом. REST не є стандартом або протоколом, а він є архітектурним стилем веб-додатків. Багато інженерів дотримуються цього архітектурного стилю для створення своїх додатків. Ці веб-програми обслуговують величезну кількість трафіку щосекунди, тому можна зробити висновок, що REST - це справді масштабований стиль архітектури. І коли

говорять про RESTful API, то мають на увазі API, який відповідає обмеженням та принципам REST.

Існує п’ять важливих обмежень та принципів для архітектурного стилю REST:

- клієнт-серверна архітектура;
- stateless принцип;
- кешування;
- многорівнева архітектура застосунку;
- уніфікований інтерфейс.

По першому принципу, в даному архітектурному стилі існує інтерфейс між клієнтом і сервером. Клієнт і сервер спілкуються за допомогою цього інтерфейсу і не залежать один від одного. Будь-яку сторону можна замінити, якщо інтерфейс залишається незмінним. Запити завжди надходять від клієнта в даній архітектурі. Застосунок для перегляду статистики та маніпуляції даними також буде реалізовувати данну архітектуру. На рис 2.3 зображене представлення клієнт серверної архітектури [13].

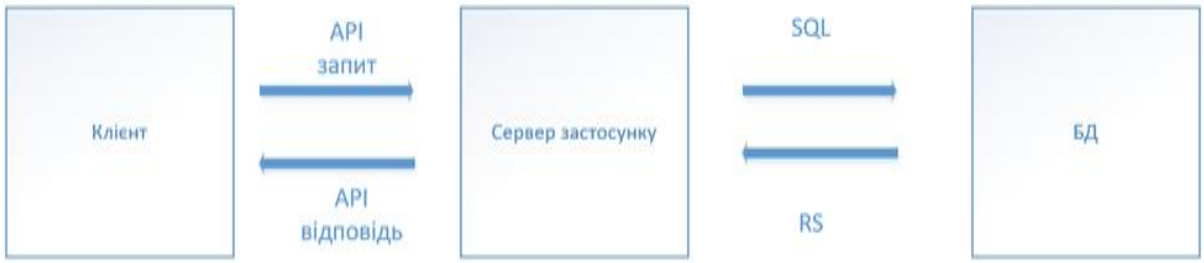


Рисунок 2.3 — архітектура “Клієнт-сервер”

По другому обмеженню, кожен запит вважається незалежним та повним. Немає ні залежності від попереднього запиту, ні залежності від сеансу для підтримання стану з'єднання.

Також запити можна кешувати на сервері або на стороні клієнта для підвищення продуктивності.

У системі може бути декілька рівнів, і мета тут полягає в тому, щоб приховати фактичну логіку та ресурси. Ці рівні можуть виконувати різні функції, такі як кешування та шифрування. Це також допомагає розділити логіку клієнта та сервера. Ця технологія почала розвиватися в останні десятиліття. Сервісно-орієнтована архітектура (SOA) почала формуватися для розподіленого дизайну, і програми почали переходити від монолітної до сучасної n-рівня архітектури, де зовнішній сервер, сервер додатків та база даних знаходились на різних віртуальних машинах чи комп'ютерах. SOA проектувались за допомогою протоколу обміну повідомленнями на основі XML, який є простим протоколом доступу до об'єктів (SOAP). Це в основному слідує моделі клієнт-сервер для створення служб.

В останні роки дизайн рішень на основі мікросервісів стає все більш популярним, що базується на обміні повідомленнями об’єктів JavaScript (JSON) та службі передачі представницького стану (REST). Це веб-API, які не потребують протоколів веб-служб на основі XML (SOAP) для підтримки своїх інтерфейсів. Вони покладаються на веб-протоколи HTTP та методи, такі як POST, GET, UPDATE, DELETE.

Однозначною перевагою таких сервісів є те, що послуги є автономними. Їх можна будувати без зовнішніх залежностей. В контексті даного проекту застосунок на Flask повністю підтримує взаємодію через API з іншими сервісами, такими як сервер CI та сервер для розгортання. Від CI серверу застосунок повинен отримувати відповідь у вигляді JSON масиву, тобто успішність стадій авто-збірки, авто-тестування можна буде перевірити через веб інтерфейс. Від серверів дистрибутива контроля версій в даному випадку через API можна отримувати статистику по комітах, а також зроблені зміни в файлах лабораторних в кожному з них.

В цьому застосунку будуть використовуватись різні методи HTTP. Наприклад, можна використовувати метод GET для запиту http://localhost:/students для отримання всіх студентів, а також цей же метод для отримання всіх даних про завдання та репозиторії але з іншим URL. Також можна використовувати метод POST для запиту http:// localhost:/students/addstudent, щоб додати інформацію про нового студента. У наступних таблицях буде приведо кілька методів для розроблювлюваних контролерів API для CRUD операцій.

Ці контролери допоможуть заносити та обробляти данні про студентів в відповідні таблиці БД. В даному контексті CRUD моделює життєвий цикл управління записами про студентів, репозиторії та завдання до бази даних. Також в майбутньому будуть розроблені контролери, які включають функціонал перегляду статистики по різним репозиторіям та взаємодії зі стороніми API.

2.3. Проектування CI процесів для репозиторіїв студентів

CI - це практика за допомогою якої код постійно проходить через стадії збірки та тестування автоматично з найменшим ручним втручанням. Для цього потрібні додаткові інструменти, як правило це спеціалізована платформа, яка обробляє код під час виконання стадій. Якщо використовувати GitLab чи Github Actions, то ці платформи предоставляють всі інструменти, необхідні для швидкого запуску цих стадій, а також практики CI/CD в цілому [15]. Далі в цьому розділі буде проведено порівняння цих платформ та дослідження продуктивності.

Реалізація платформи безперервної інтеграції та постійного розгортання від GitLab та GitHub Actions втілює ідею програм, які встановлюються на інших серверах та допомогають запускати всі стадії безперевної інтеграції та безперервного розгортання. Ці програми називаються runners. Сама програма може представляти собою віртуальну машину, контейнер чи віддалений сервер.

Для реалізації даного проекту краще всього підійте контейнер з оточенням Ubuntu в яке під час підготовчих стадій може бути встановлене додаткове ПЗ для запуску програм написаних на різних мовах програмування.

Наступним етапом проектування робочих процесів CI є вибір архітектури конвеєра. Конвеєр у світі програмної інженерії розуміється як ланцюг подій чи процесів, який автоматично запускає стадії відповідно до їх опису та передає вихідні дані до наступного елемента

Ланцюг подій запускається комітом або через тригер в GitLab чи GitHub Actions. Система збірки отримує повідомлення про нову версію та проводить компіляцію в вихідний код і запускає модульне тестування чи будь яке інше тестування. Встановлення та налаштовування конвейеру завжди починається з налаштовувань завдань [9]:

- завдання є найважливішим елементом конвеєру;
- завдання створюються з обмеженнями, які визначають, за яких умов вони повинні виконуватися;
- завдання можуть мати довільну назву і повинні містити елемент сценарію як мінімальну;
- кількість завдань для одного конвеєра є необмеженою.

Конвейери, як правило, працюють на основі певних умов, які виконуються. Графіки конвейерів можуть використовуватися для виконання стадій через певні проміжки часу. Наприклад:

- щомісяця;
- якогось числа;
- раз на день;
- cron графік;
- кожну годину.

У наступному розділі буде наведено приклад конкретної реалізації опису завдань, а в цьому розділі приведено загальний шаблон для проектування конвейеру. У контексті даного проекту студент, завдання якого проходять перевірку, має сам описувати всі стадії в файлі репозитрію свого проекту, а далі вже викладач має переглядати результати через застосунок, який витягує результати тестування через API GitHub Actions.

Загальний шаблон проектування для даного проекту представлений в таблиці нижче.

Таблиця 2.4 – Базовий проект для пайплайну, який має бути описаним в репозиторіях студентів

Стадії	Опис	Вхідні дані	Вихідні дані
Підготовка	Встановлення всіх необхідних пакетів в середовище програми GitLab та GitHub runner.	Команди CLI before install.	Образ ОС віртуальної машини чи контейнера зі встановленими залежностями.
Збірка	Запуск Maven build чи Gradle build.	Команди CLI та образ ОС.	Артефакт збірки, архів для розгортання(JAR).
Тестування	Запуск тестів, які були вказані.	Команди CLI та образ ОС, артефакт збірки	Журнал логів тестів.

Дані завдання ідуть чітко один за одним і не можуть виконуватись поки попередні завдання не завершаться успішно. Це є прикладом базового конвейеру, який краще всього підходить для реалізації даного проекту. Взагалі існує три основні способи побудови конвейерів, кожен із яких має свої переваги. Ці методи можна змішувати та підбирати, якщо це необхідно [4]:

- базовий - підходить для простих проектів, де вся конфігурація знаходиться в одному легко знайти місці;
- направлений ациклічний граф - підходить для великих, складних проектів, які потребують ефективного виконання;
- конвейер дочірніх та батьківських стадій - підходить для монорепозиторіїв та проектів з великою кількістю самостійно визначених компонентів.

В даному проекті буде використано найпростіший конвеєр GitHub Actions. Він одночасно запускає все на стадії збірки, і як тільки все закінчується, він запускає все на стадії тесту. Він не найефективніший, але його простіше підтримувати. На рис 2.4 зображене представлення даного виду конвейера

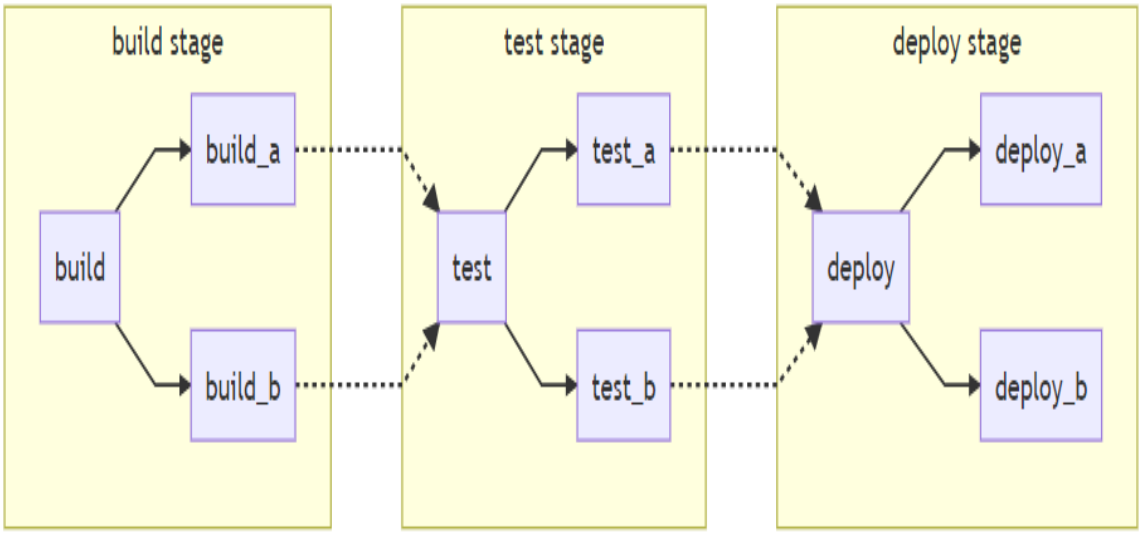


Рисунок 2.4 — Представлення базового конвеєру CI/CD GitHub Actions

2.4. Проектування взаємодії компонентів CI/CD системи

У попередніх пунктах даного розділу було зроблено проектування компонентів системи автоматизації CI/CD, а в цьому пункті даного розділу буде спроектовано їхню взаємодію. Впровадження даної системи вимагає детального аналізу та сбору інформації про кожен компонент, адже інтеграція різних стадій процесу розробки програмного забезпечення у системи CI / CD може значно скоротити час циклу розробки та підвищити продуктивність, а також скоротити помилки при перевірці належного виконання розробки програмного забезпечення. Розуміючи різні частини конвеєру програмного забезпечення та автоматизуючи процеси SDLC, можна дозволити розробникам, а в даному випадку студентам, працювати значно швидше та ефективніше.

Далі буде приведено результат поєднання компонетів у вигляді діаграми, яка відображає взаємозв’язки компонентів. На рис 2.5 приведено архітектурну діаграму C4 третього рівня, яка відображає взаємодію компонентів системи автоматизації оцінювання та виконання лабораторних та практичних робіт.

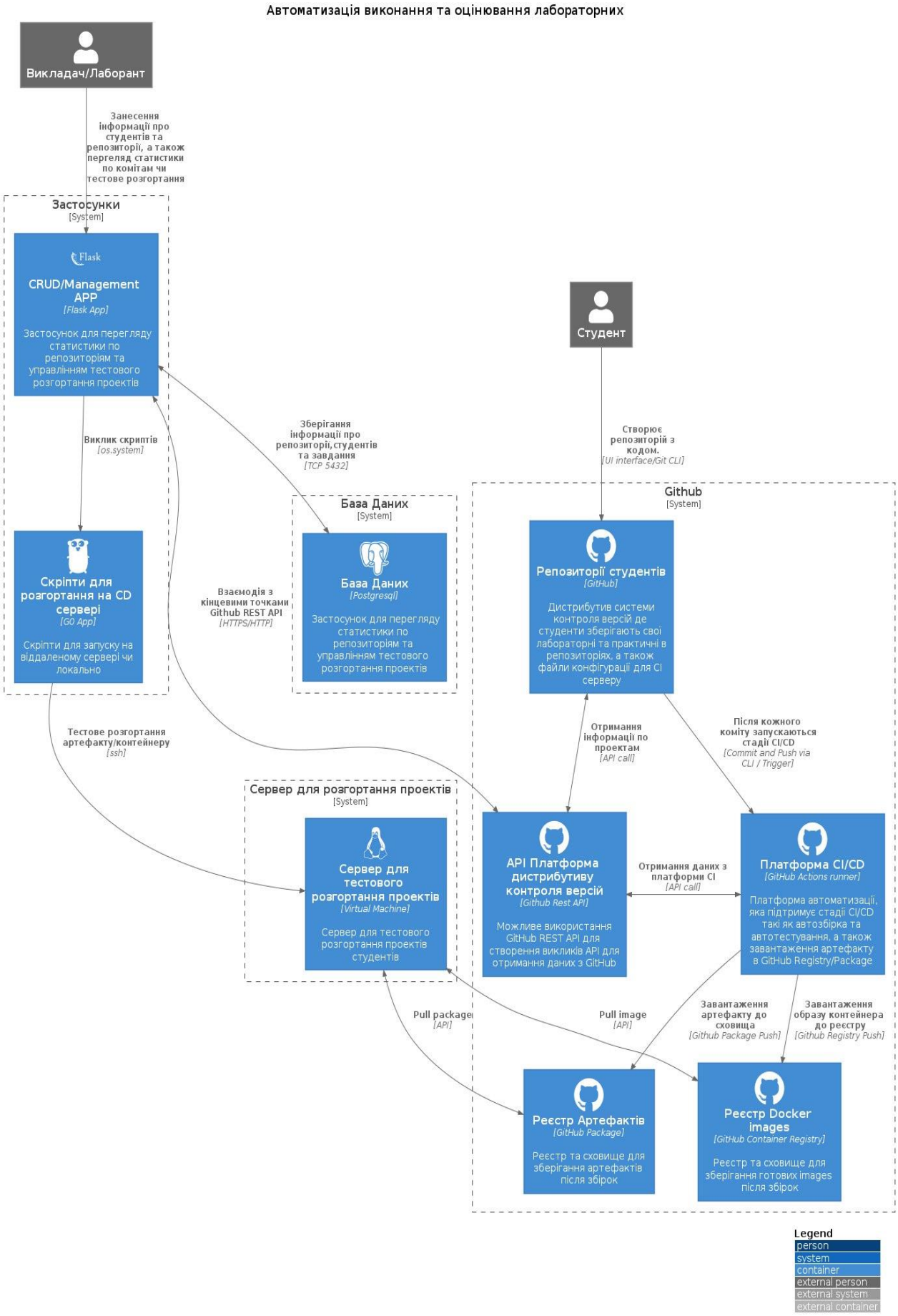


Рисунок 2.5 — Взаємодія різних компонентів проекту

2.5. Дослідження продуктивності CI платформи GitHub Actions для різних проектів

В попередніх пунктах розділу було проведено проектування системи автоматизації оцінювання та виконання лабораторних та практичних робіт, а цей пункт зосереджений на дослідженні продуктивності CI платформи GitHub Actions. В загальному плані, ця платформа є компонентом даної CI/CD системи та покликана автоматизувати тестування та інші стадії перевірки для не тільки кількісної, але й якісної оцінки робіт. Але дана платформа потрібна впершу чергу автоматизувати діяльність розробників ПЗ, а отже данне дослідження потрібне для визначення продуктивності в контексті розробки будь-яких застосунків, тобто з використанням популярних мов програмування та фреймворків. Даний метод дослідження базується на включенні автоматизованого тестування продуктивності базового конвейеру GitHub Actions та оцінювання продуктивності стадій безпервної інтеграції.

В загальному розумінні, це процес вимірювання продуктивності та збору інформації про стан системи. На відміну від моніторингу, порівняльний аналіз має на меті відповісти на конкретне питання і зазвичай використовується для порівняння різних версій систем, конфігурацій, альтернатив або розгортань. Більше того, порівняльний аналіз, як правило, вимірює якість невиробничого середовища з довільними показниками в певний час під час декількох тестових запусків та аналізує його результати в подальшому офлайн-аналізі. Він може включати запущені мікротести, які вимірюють дуже малі окремі характеристики аж до продуктивності одного методу. Проблема таких тестів автоматизації в тому що зазвичай перевірка грунтується на функціональних тестах або невеликих мікроефективних показниках, таких як продуктивність одної стадії, тоді як загальна якість обслуговування (QoS) ігнорується. Але було знайдено технологію, яка може допомогти визначити в цілому продуктивність CI процесів та забезпечити належну перевірку QoS. Це є розширенням GitHub Marketplace, яке має назву Continuous Benchmark. Також вона забезпечує відслідковування споживаних системою оцінювання ресурсів, які є накладними витратами на оцінювання

продуктивності. Для різних мов прграмування ця технологія представлена в різних бібліотеках даних мов програмування, також постачальник гарантує сумісність с GitHub Actions та іншими платформами автоматизації CI. Ця технологія підтримує наступні мови програмування та бібліотеки:

- Rust проекти;
- Go проекти;
- JavaScript/TypeScript проекти;
- Python проекти;
- C++ проекти.

Для мови програмування Java є власний фреймворк, який допомагає значно краще оцінити продуктивність конвейеру для застосунків написаних на даній мові програмування. JMH - це набір інструментів, який допомагає правильно впровадити вимірювання для різних фреймворків Java. JMH розробляється тими ж програмістами, які розробляють віртуальну машину Java.

Для даного дослідження буде проведено тест продуктивності однієї й ж тої самої функції для знаходження перших десяти та двадцяти чисел послідовності Фібоначі.

Даний підхід вимагає певних показників для прийняття рішення про успіх або невдачу вимірювання. Залежно від характеристик серверу CI, це рішення може базуватися на фіксованих значеннях, наведених у SLA. Тобто це рішення буде відхиляти збірку через раптове та значне падіння або стрибок значень вимірювання х порівняно з останньою збіркою, або виявляти негативну тенденцію протягом декількох збірок. Тут буде представлено алгоритми прийняття рішень, які згодом будуть використані при оцінці. Найпростіший метод виявлення небажаних збірок - застосовувати фіксовані межі, наприклад з SLA чи встановленого інтервалу відхилення. Збірка відхиляється, якщо визначена метрика х не знаходиться в певному інтервалі.

$$f(x) = \begin{cases} 1, & \text{якщо FVmin} < x < FVmax \\ 0, & \text{інакше} \end{cases}$$

Де f(x) – функція на основі якої приймається рішення про відхилення чи прийняття результатів вимірювання. FVmin – нижня межа інтервалу допустимих значень вимірювання. FVmax – верхня межа інтервалу допустимих значень вимірювання.

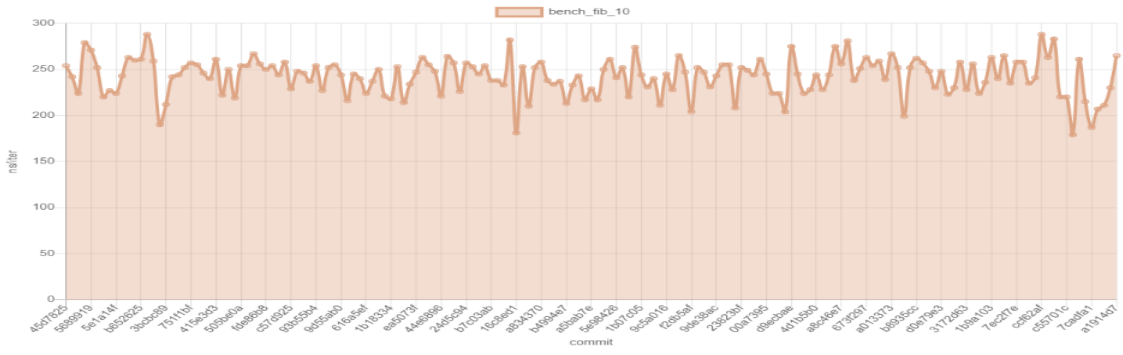
Відносне порівняння поточної метрики часу збірки x до попередньої метрики часу збірки y може використовуватися для відстеження відхилення збірок, в яких значення вимірювань x відхиляється від останньої збірки більш ніж на t відсотків.

$$f(x,y) = \begin{cases} 1, & \text{якщо } t > 100(\frac{x}{y} - 1) \\ 0, & \text{інакше} \end{cases}$$

Далі буде приведено результати порівняльного аналізу продуктивності конвейерів GitHub Actions для різних мов програмування. Було зроблено серію комітів, які потрібні для відображення загальної статистики продуктивності всіх стадій конвейерів. Порівняльний аналіз серії комітів двух функцій, які реалізовані в одному проекті, може бути використаний для цілей знаходження правильних метрик QoS. Вони представлені в вигляді діграм продуктивності для різних фреймворків та мов програмування. Алгоритм уточнення даних полягає в тому що після кожного коміту та завантаження вмісту локального репозиторія у віддалене сховище запускається конвейер, який кожен раз після прооходження всіх стадій формує нову діаграму базуючись на результатах попередніх виконань конвейерів, а також на результаті виконання даного конвейеру. Ці діаграми включають в себе:

– фіксовані хеші комітів;
– значення тривалості кожної ітерації під час збірки та тестування.

На рисунках 2.6-2.10 зображено графіки результатів порівняльного аналізу двух функцій fib_10 та fib_20 для різних мов програмування.



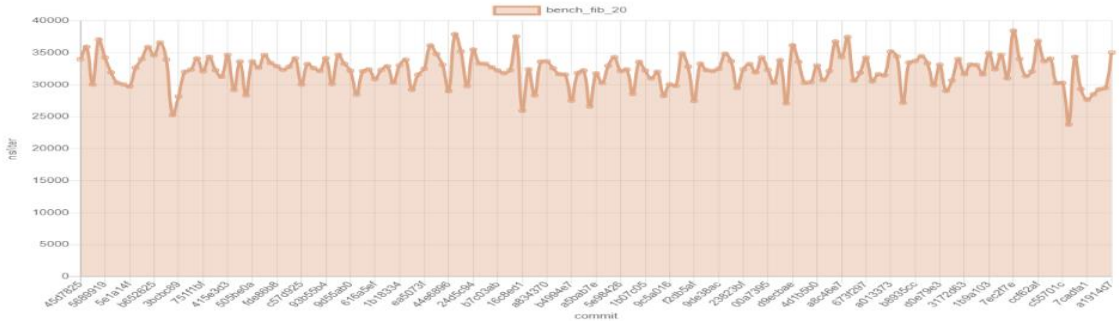


Рисунок 2.6 — Результати тетсування продуктивності ковейеру GitHub Actions для Rust проекту fib_10 та fib_20

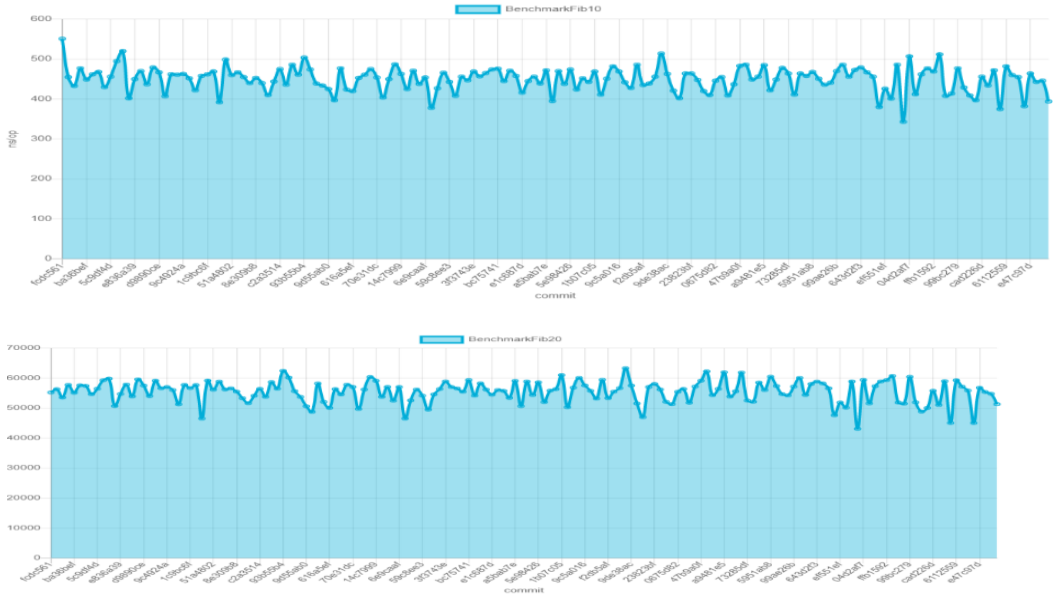


Рисунок 2.7 — Результати тетсування продуктивності ковейеру GitHub Actions для Go проекту fib_10 та fib_20

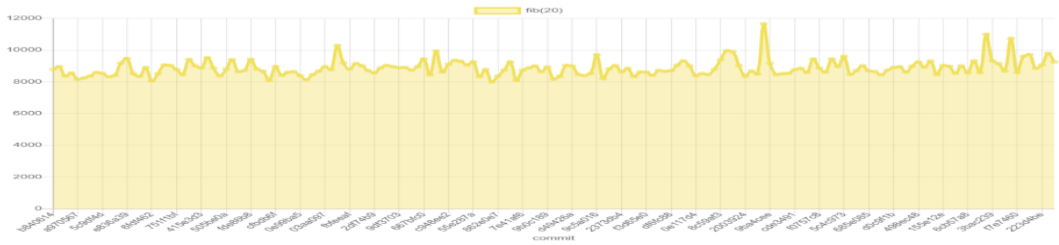


Рисунок 2.8 — Результати тетсування продуктивності ковейеру GitHub Actions для JavaScript проекту fib_20

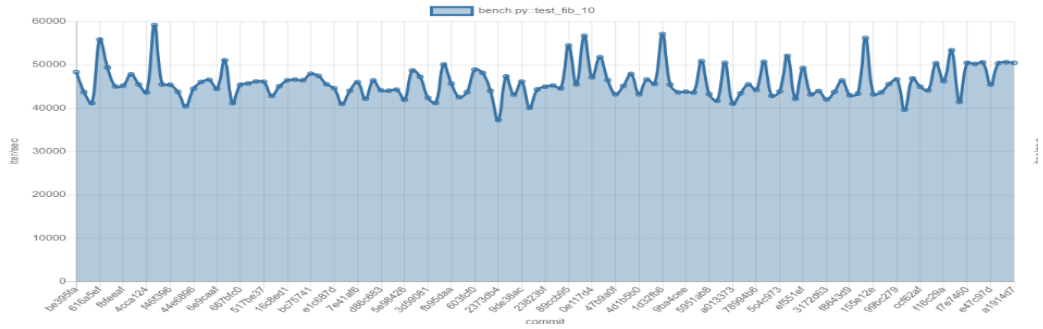


Рисунок 2.9— Результати тетсування продуктивності ковейеру GitHub Actions для Python проекту fib_20

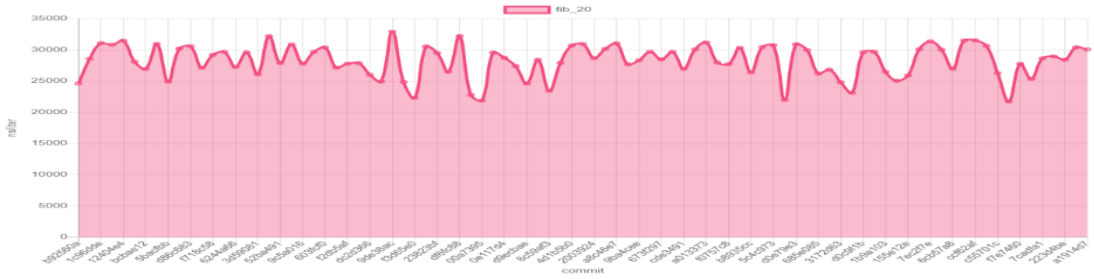


Рисунок 2.10 — Результати тетсування продуктивності ковейеру GitHub Actions для C++ проекту fib_20

Далі буде приведено діаграми порівняльного аналізу двух функцій для мови програмування Java, але для різних реалізацій VM. Так як переважна більшість лабораторних та практичних завдань виконуються на мові програмування Java, то доречно порівняти реалізації віртуальних машин. Станом на поточний момент Java має кілька популярних реалізацій VM:

- HotSpot VM;
- OpenJDK VM;
- GraalVM.

Для різних функцій fib, які мають виконуватися на різних реалізаціях JVM, було проведено окремий аналіз продуктивності конвейерів. Але даний графік має інший прицип порівняння, тобто тут порівнюються декілька різних алгоритмів пошуку перших 20 чисел послідовності Фібоначі на різних реалізаціях Java. Також замість тривалості усіх стадій CI, тут порівнюється кількість операцій в секунду. Це обумовлене тим, що набір інструментів JMH має таку методику тестування. На рисунку 2.11 зображено діаграму порівняння.

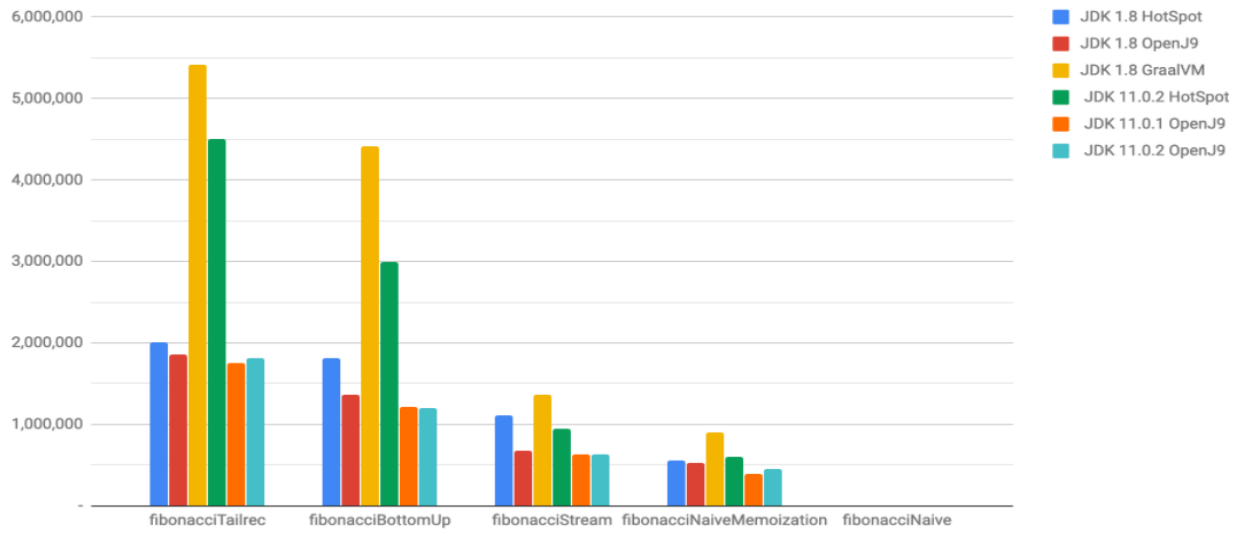


Рисунок 2.11 — Результати тестування продуктивності конвейерів для різних реалізацій JVM та різних алгоритмів пошуку перших 20 чисел фібоначі

Це розширення може бути використане для виявлення проблем продуктивності в конвейерах різних проектів, які використовують систему контролю версій Git та CI сервер Github Actions. В даному контексті було протестовано прості алгоритми знаходження послідовності перших чисел фібоначі, але дане розширення може бути використане для тетстування кінцевих точок HTTP або виконання більш складної логіки проектів.

Висновки до розділу 2

Проектування системи автоматизації – це один із початкових етапів впровадження та дослідження, який необхідним для подальшого та належного створення будь-якої інтеграції різних компонентів.

У цьому розділі було описано модель даних для СУБД, яка має зберігати інформацію про студентів, репозиторії завдання. Також був спроектований веб-застосунок, який буде предоставляти можливість переглядати різні репозиторії студентів та порівнювати їх між собою.

Також було спроектовано основні стадії конвейеру для репозиторіїв студентів, а також був продуманий процес налаштування всіх стадій CI/CD в GitHub Action Workflow. Ці стадії допоможуть студентам проводити автоматизоване тестування свого коду різних проектів, а також отримувати на

виході артефакт, який в подальшому може бути розгорнутим локально чи на віддаленому сервері.

У даному розділі наявна архітектурна діаграма інтеграції всіх компонентів, які потрібні для реалізації даної системи автоматизації. У наступному розділі буде описано процес створення та інтеграції усіх цих компонентів.

І в останьому пункті було проведено дослідження продуктивності конвейерів для різних мов програмування. Для цих цілей було використано один і той же самий алгоритм, але реалізований на різних мовах програмування.

РОЗДІЛ 3. СТВОРЕННЯ СИСТЕМИ АВТОМАТИЗАЦІЇ НА ОСНОВІ ТЕХНОЛОГІЙ CI/CD В ПРЕДМЕТНІЙ ОБЛАСТІ

3.1. Створення бази даних для зберігання інформації про студентів та репозиторії

Самим першим компонентом з якого потрібно починати створювати майже будь-яку інформаційну систему є база даних. У попередніх розділах було спроектовано належним чином схему БД, а також було приведено моделі даних, які вона повинна зберігати. Є кілька методів створення БД: автоматичне створення в різних інструментах, які підтримують конвертацію фізичної схеми в БД, а також за допомогою скриптів, ще існує імпорт, але за наявності згенерованого та експортованого файлу схеми БД. Також є метод полуавтоматичного створення БД з подальшим створенням всіх таблиць через застосунок при першому запуску. Це реалізується за допомогою ORM. Для різних фреймворків існує кілька реалізацій маперів сутностей:

- Hibernate;
- JDO;
- Oracle;
- TopLink;
- iBATIS SQL;
- Maps and JPA;
- SQLAlchemy;
- Django ORM.

Для реалізації даного проекту в ході попередніх розділів було вибрано фреймворк Flask, а він в свою чергу підтримує SQLAlchemy, який дуже схожий на всі інші вищезазначені ORM. Так як в усіх цих ORM є можливість декларувати таблиці за допомогою сутностей, які являють собою класи з атрибутами, які в свою чергу представляють колонки таблиць в БД. Кожен атрибут має тип даних, який відображає тип даних в таблиці. В загальному розумінні, SQLAlchemy - це API для різних дистрибутивів БД, яке виконує реляційне відображення об’єктів на

найвищому рівні. ORM - це інструмент, який дозволяє розробникам зберігати та отримувати дані за допомогою об'єктно-орієнтованих підходів та вирішувати об'єктно-реляційні питання. Реляційні та об’єктно-орієнтовані моделі настільки різні, що для їх ефективної спільної роботи потрібні додатковий код та свої функціональні можливості. На рис 3.1 зображений код декларативного опису таблиць БД для зберігання даних про студентів та репозиторії.

```
class Student(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    First_name = db.Column(db.String(50))
    Last_name = db.Column(db.String(50))
    email = db.Column(db.String(120), nullable=True)
    Group = db.Column(db.String(20))
    repositories = db.relationship('Repository',
                                   backref='student',
                                   lazy=True,
                                   primaryjoin="Student.id == Repository.student_id"

    def __init__(self, First_name, Last_name, email, Group):
        self.First_name = First_name
        self.Last_name = Last_name
        self.email = email
        self.Group = Group

    def __repr__(self):
        return "{0} {1} {2} {3}".format(self.First_name,
                                        self.Last_name,
                                        self.email,
                                        self.Group)

class Repository(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    url = db.Column(db.String(120))
    student_id = db.Column(db.Integer, db.ForeignKey('student.id'),
        nullable=False)
    task_id = db.Column(db.Integer, db.ForeignKey('task.id'),
        nullable=False)
    Grade = db.Column(db.Integer,  nullable=True)
    type = db.Column(db.Integer, nullable=True)

    def __init__(self, url, student_id, task_id, Grade, type):
        self.url = url
        self.student_id = student_id
        self.task_id = task_id
        self.Grade = Grade
        self.type = type
```

Рисунок 3.1 – Декларативний опис таблиць Students та Repository за допомогою сутностей SQLAlchemy

В даному коді поля атрибутів класа є полями таблиць в БД, а різні види конструкторів потрібні при створенні об'єктів сутності для створення записів в таблицях баз даних. Також метод repr потрібен для швидкого отримання запису БД в належному виді. Далі на рис 3.2 буде зображений код декларативного опису таблиць БД для зберігання даних про завдання.

```
class Task(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(50))
    description = db.Column(db.String(200))
    type = db.Column(db.Integer, nullable=True)
    first_deadline = db.Column(db.Date, nullable=True)
    second_deadline = db.Column(db.Date, nullable=True)
    third_deadline = db.Column(db.Date, nullable=True)
    repositories = db.relationship('Repository',
                                   backref='task',
                                   lazy=True,
                                   primaryjoin="Task.id == Repository.task_id")

    def __init__(self, name, description,type, first_deadline,second_deadline,third_deadline)
        self.name = name
        self.description = description
        self.type = type
        self.first_deadline = first_deadline
        self.second_deadline = second_deadline
        self.third_deadline = third_deadline




    type_dict = {
        0: 'Lab',
        1: 'PZ',
        2: 'Project',
    }

    def type_string(self):
        return self.type_dict[self.type]
```

Рисунок 3.2 – Декларативний опис таблиць Task за допомогою сутностей SQLAlchemy

На попередніх сніпетах коду було приведено централізований опис сутностей застосунку в файлі models.py. Після опису сутностей в ORM потрібно створити базу даних в СУБД. Але перед цим потрібно створити нове віртуальне середовище для всіх залежностей додатку, а також потрібно буде встановити певні пакети для бази даних PostgreSQL, які будуть виконувати роль драйвера для SQLAlchemy.

3.2. Сворення застосунку для перегляду статистики та порівняння лабораторних та практичних робіт

Для отримання та маніпуляції даними, а також для збору статистики чи порівння лабораторних та практичних робіт потрібен застосунок, який дозволяє переглядати всі зміни внесені студентам в репозиторії в одному місці, а також виставляти їм оцінки. Також цей застосунок певною мірою повинен дозволяти виконувати різні дії над проектами, такі як відкриття в IDE кожної версії лабораторної чи практичної, а також розгортання на віддаленому сервері чи локально.

Спочатку потрібно реалізувати функціонал для редагування та маніпуляцією даними. Для цього потрібно створити CRUD. У кожному механізмі зберігання даних існує чотири основних типи функцій: створення, читання, оновлення та видалення. Вони дозволяють виконувати всі основні способи обробки та перегляду даних, необхідних для різних веб-програм. Для реалізації потрібно використовувати методи контролерів, які є компонентами MVC.

У даному фремворку на якому буде написано застосунок, blueprint є методом розширення існуючої програми. Він забезпечує спосіб поєднання груп контролерів із загальною функціональністю і дозволяє розробникам розбити додаток на різні компоненти. У архітектурні схемі даного застонку в контролерах будуть знаходитися різні функції CRUD для маніпуляції даними. Як було зазначено в розділі для проектування, для даного застосунку потрібно створити мінімум 3 контролери для CRUD, а також кілька інших контролерів для перегляду статистики та інших корисних дій.

Контролери MVC відповідають за контроль потоку виконання програми. Коли робиться запит до програми MVC, контролер відповідає за повернення відповіді на цей запит. Контролер може виконувати одну або кілька дій. Далі буде приведено сніпети коду для реалізації контролерів Students, Repo та Tasks. На рис 3.3 зображений сніпет коду контролера Students.

```python
from flask import render_template, request, redirect

from models import Student, db


def students():
    students = Student.query.all()
    return render_template('students/viewStudent.html', students=students)


def addStudent():
    if request.method == 'GET':
        return render_template('students/addStudent.html')
    elif request.method == 'POST':
        student = Student(request.form['First_name'],
                    request.form['Last_name'],
                    request.form['email'],
                    request.form['Group'],
                    )
        db.session.add(student)
        db.session.commit()
        return redirect("/addStudent")



def editStudent():
    students = Student.query.all()

    return render_template('students/editStudents.html', students=students)


def updateStudent(id):
    student = Student.query.filter_by(id=id).first()
    if request.method == 'POST':
        if student:
            student.First_name = request.form['First_name']
            student.Last_name = request.form['Last_name']
            student.email = request.form['email']
            student.Group = request.form['Group']

            db.session.commit()
            return redirect('/editStudents')


def deleteStudent(id):
    student = Student.query.filter_by(id=id).first()
    if request.method == 'POST':
        if student:
            db.session.delete(student)
            db.session.commit()
            return redirect("/students")
    if request.method == 'GET':
        return id
```

Рисунок 3.3 – Сніпет коду контролеру Student CRUD

Вищевказаний код реалізує можливість додавання, редагування та видалення даних про студентів. Він звертається до об’єкту сутності Students та присвоює значення атрибутів. Тобто кожний метод під час отримання даних з форм присвоює атрибути екземпляру моделі та зберігає ці зміни. Далі буде приведно фрагмент коду для реалізації моделі CRUD. На рис 3.4 зображений сніпет коду контролера Repositories.

```
def addRepo():
    if request.method == "GET":
        students = Student.query.all()
        tasks =  Task.query.all()
        return render_template('repos/addRepo.html', students=students, tasks=tasks)
    elif request.method == 'POST':
        if request.form['Grade'] == '':
            repo = Repository(request.form['url'],
                              request.form['student_id'],
                              request.form['task_id'],
                              None,
                              request.form['type'], )
        else:
            repo = Repository(request.form['url'],
                        request.form['student_id'],
                        request.form['task_id'],
                        request.form['Grade'],
                        request.form['type'],)


        db.session.add(repo)
        db.session.commit()
        return redirect("/addRepo")



def editRepo():
    repositories = Repository.query.all()
    students = Student.query.all()
    tasks = Task.query.all()

    return render_template('repos/editRepos.html',repositories=repositories, students=students, tasks=tasks)
```

Рисунок 3.4 – Сніпет контролеру Repositories CRUD

Код для маніпуляції даними репозиторіїв має подібний функціонал до інших контролерів та дозволяє виконувати всі дії, які є в моделі манпуляції даними CRUD. Після того як було проведено певні операції над даними, контролери генерують шаблони відображення, які можуть включати форми для користувачів. Також контролери можуть перенаправляти на інші методи.

На рис 3.5 зображений сніпет коду контролера Tasks.

```python
def addTask():
    if request.method == 'GET':
        today = date.today()
        return render_template('tasks/addTask.html', today=today)
    elif request.method == 'POST':
        task = Task(request.form['name'],
                    request.form['description'],
                    request.form['type'],
                    request.form['first_deadline'],
                    request.form['second_deadline'],
                    request.form['third_deadline'],
                    )
        db.session.add(task)
        db.session.commit()
        return redirect("/addTask")



def editTasks():
    tasks = Task.query.all()

    return render_template('tasks/editTasks.html', tasks=tasks)




def updateTask(id):
    task = Task.query.filter_by(id=id).first()
    if request.method == 'POST':
        if task:
            task.name = request.form['name']
            task.description = request.form['description']
            task.type = request.form['type']
            task.first_deadline = request.form['first_deadline']
            task.second_deadline = request.form['second_deadline']
            task.third_deadline = request.form['third_deadline']

            db.session.commit()
            return redirect('/editTasks')
```

Рисунок 3.5 – Сніпет контролеру Task CRUD

Ці вищезазначені контролери допоможуть заносити дані та виводити їх предеставлення, а також маніпулювати ними. Але процес занесення даних про студентів, репозиторії та завдання потрібно автоматизувати, саме для цих цілей було розроблено контролер dataupload. Він дозволяє зчитувати дані з файлів Excel, CSV та додавати їх до БД. На рис 3.6 зображений сніпет коду контролера DataUpload.

```python
@app.route('/upload', methods=['GET', 'POST'])
def uploadTasks():
    return render_template('Upload/Upload.html')


@app.route('/data', methods=['GET', 'POST'])
def data():
    if request.method == 'POST':
        file = request.form['upload-file']
        data = pd.read_excel(file).to_dict()
        for i in range(len(data['Завдання'])):
            task = Task(data['Завдання'][i],
                data['Опис'][i],
                data['Тип'][i],
                data['Перший дедлайн'][i],
                data['Другий дедлайн'][i],
                data['Третій дедлайн'][i],)
            db.session.add(task)
            db.session.commit()

        return redirect('/upload')

@app.route('/dataStudentsRepo', methods=['GET', 'POST'])
def dataStudentsRepo():
    if request.method == 'POST':
        file = request.form['upload-file']
        data = pd.read_excel(file).to_dict()
        for i in range(len(data['ПІБ'])):
            student = Student(
                data['ПІБ'][i].split(' ')[1],
                data['ПІБ'][i].split(' ')[0],
                data['email'][i],
                data['Група'][i],
            )
            db.session.add(student)
            db.session.commit()
            repo = Repository(
                data['URL'][i],
                Student.query.order_by(Student.id.desc()).first().id,
                data['Task(id)'][i],
                None,
                data['Type'][i],
            )
            db.session.add(repo)
            db.session.commit()


        return redirect('/upload')
```

Рисунок 3.6 – Сніпет контролеру DataUpload(File Upload)

Для перегляду статистики по комітам та її візуалізації було розроблено наступні контролери:

- stat;
- chart.

Код для даних контролерів приведено в додатку до пояснювальної записки, а в цьому розділі буде приведено результат графічного представлення зібраних даних GitHub API. Ця бібліотека API сумісна зі специфікаціями Open API. Існує багато варіантів використання OpenAPI, таких як:

- створення власного клієнту API;
- перевірка та тестування кінцевих точок API GitHub REST;
- дослідження та взаємодія з GitHub REST API;
- використання сторонніх інструментів;
- використання інструментів таких як Postman.

GitHub-Flask - це розширення, яке дозволяє аутентифікувати користувачів в GitHub за допомогою протоколу OAuth і викликати методи API. На відміну від звичайного використання методів роботи з API, дане розширення дозволяє на безлімітній основі використовувати запити API до серверів GitHub. В даному застосунку за допомоги використання токену OAuth та username, відбувається аунтефікація в Github.

Дані контролери використовують методи для запитів до централізованих серверів Github для отримання таких даних як:

- хеш коміту;
- дата;
- вітка(номер лабораторної);
- опис коміту(повідомлення);
- статистика по правкам файлів для кожного коміту;
- статистика змінненних ліній коду;
- статистика доданих ліній коду;
- статистика по кількості доданих, змінненних, видалених файлів.

Ці всі дані збираються в одному місці для полегшення перегляду статистики та прийняття рішення по оцінці.

На рис 3.7 зображене представлення для контролера stat.

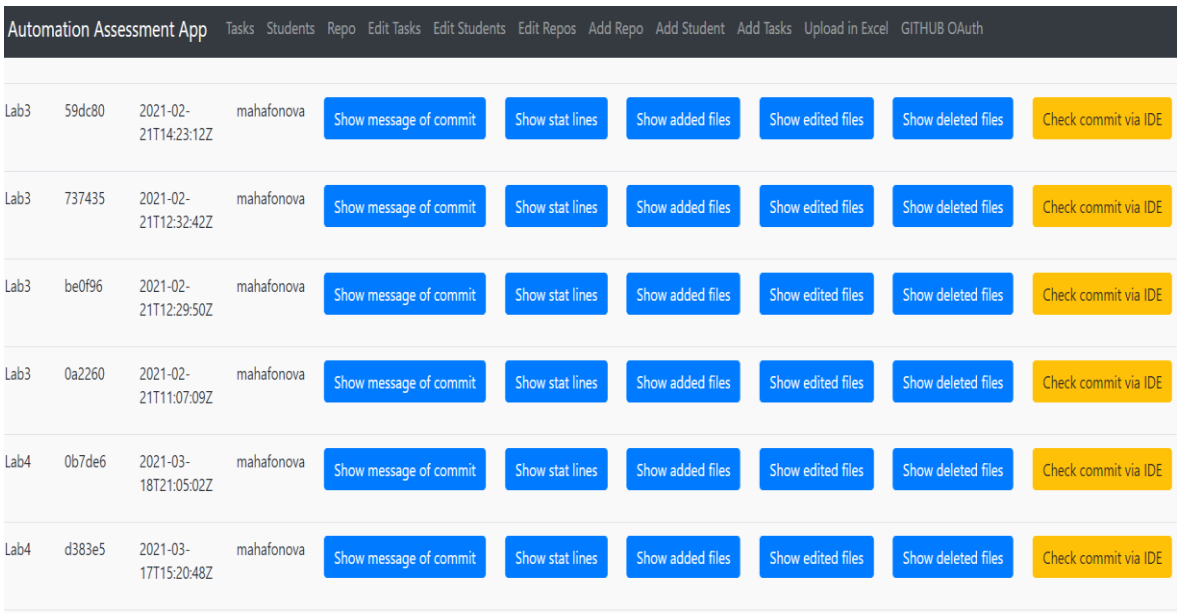


Рисунок 3.7 – Представлення для контролера stat

Для візуалізації статитики було обрано наступні dashboards:

– кількість комітів за кожну дату;

– кількість змін та нововведень під час кожного коміту.

На рис 3.8 зображене представлення контролера chart.

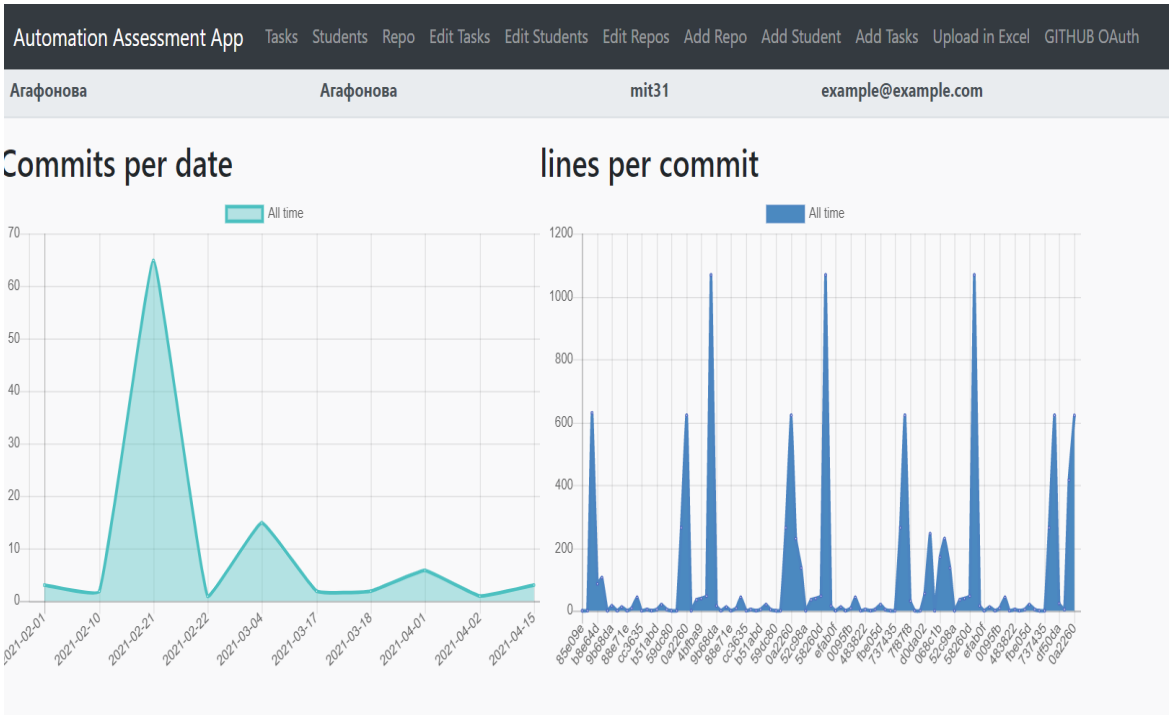


Рисунок 3.8 – Представлення для контролеру chart

Також в даному розробленому застосунку підтримується можливість відкривати проекти студентів в IDE. За дані дії відповідає контролер ide.py. Він зчитує та завантажує відалений репозиторій та відкриває в VS code на заданій версії

проекту. Ці вищезазначені дії робляться через бібліотеку запитів os на мові програмування Python. На рис 3.9 зображений сніпет коду для реалізації заданих дій.

```
import os

from flask import request, redirect

from models import Repository


def openide(id):
    if request.method == 'POST':
        if os.name == 'nt':
            url = Repository.query.filter_by(id=id).first()
            os.system('cd .\\labs\\ && git clone {}'.format(url.url))
            repo=url.url.split('/')[4].replace('.git', '')
            os.system("code .\\labs\\{}\\".format(repo))
            return redirect("/simple_chart/{}/".format(id))


def openidesha(id,sha):
    if request.method == 'POST':
        if os.name == 'nt':
            url = Repository.query.filter_by(id=id).first()
            os.system('cd .\labs\ && git clone {}'.format(url.url))
            repo=url.url.split('/')[4].replace('.git', '')
            os.system('mkdir .\\labs\\{}\\{}\\'.format(sha, repo))
            os.system('cd .\\labs\\{}\\{}\\ && git clone {} && cd .\\{}  && git checkout {}'.format(sha,repo,url.url,repo,sha))
            os.system('code .\\labs\\{}\\{}\\{}\\'.format(sha,repo,repo))
            return redirect("/statistics/{}/".format(id))
```

Рисунок 3.9 – Контролер, який дозволяє відкривати код на будь-якому коміті та переглядати зміни

Наступним кроком в створенні програмного додатку є реалізація порівняння різних робіт за подібністю та виведення списку в відсортованому порядку. Є кілька шляхів для впровадження цьої функції. В тому числі такими способами як:

- скрапінг веб інтерфейсу та пошук файлів в GitHub;
- використання GitHub API;
- клонування репозиторіїв та перевірка їх на локальному комп'ютері.

Для перших двох способів потрібно мати дуже добре та швидке інтернет підключення. Тому що кожен раз при порівнянні двох файлів в проекті потрібно підключатися до віддалених веб серверів та отримувати відповідь від них, яку потрібно обробляти. Також операції зв’язані з підключенням займають значного часу. На відміну від перших двох способів, останій має багато переваг. Одна із них, це те що не потрібно кожен раз для порівняння підключатись до віддалених серверів, достатньо всього один раз клонувати ці всі репозиторії та робити порівняння локально [15]. В даному проекті використовуються скріпти написанні

на мові програмування Go, які можуть бути використані для швидкого порівняння директорій. На рис 3.10 зображений сніпет коду для реалізації цих скріптів для порівняння двох директорій та файлів в них.

```
const has_mmap = true

var win_mapper_mutex sync.Mutex
var win_mapper_handle = make(map[uintptr]syscall.Handle)

// Implement mmap for windows
func map_file(file *os.File, offset, size int) ([]byte, error) {

        // create the mapping handle
        h, err := syscall.CreateFileMapping(syscall.Handle(file.Fd()), nil, syscall.PAGE_READONLY, 0, uint32(size), nil)
        if err != nil {
                return nil, err
        }

        // perform the file map operation
        addr, err := syscall.MapViewOfFile(h, syscall.FILE_MAP_READ, uint32(offset>>32), uint32(offset), uintptr(size))
        if err != nil {
                return nil, err
        }

        // store the mapping handle
        win_mapper_mutex.Lock()
        win_mapper_handle[addr] = h
        win_mapper_mutex.Unlock()

        // Slice memory layout
        sl := reflect.SliceHeader{Data: addr, Len: size, Cap: size}

        // Use unsafe to turn sl into a []byte.
        bp := *(*[]byte)(unsafe.Pointer(&sl))

        return bp, err
}
```

Рисунок 3.10 – Сніпет коду скриптів для порівняння

Також в застосунку була реалізована можливість викликаки задані скріпти та бачити результати порівняння в представленні контролера comparison.py. Цей контролер викликає даний скріпт через метод system бібліотеки os мови програмування Python. Потім для кожного репозиторія відбувається порівняння з іншими репозиторіями. Тобто працює цикл в якому відбувається передача параметрів шляхів до двух директорій в скріпт [11]. На рис 3.14 зображене представлення цього контролеру.

Automation Assessment App Tasks Students Repo Edit Tasks Edit Students Edit Repos Add Repo Add Student Add Tasks Upload in Excel GITHUB OAuth

Lab	Percent	Suspicious student	Target	RepoName
Lab4	19.49 %	Manilo	Kateryna	SampleJava2021
Lab4	19.48 %	Manilo	Kateryna	SampleJava2021
Lab4	5.69 %	Manilo	Oleksandr	sampleNevm
Lab4	5.68 %	Manilo	Oleksandr	sampleNevm
Lab4	2.59 %	Manilo	Ivan	JavaWeb
Lab4	2.33 %	Manilo	Myroslav	javarepos
Lab4	1.96 %	Manilo	Dmytro	labsjava
Lab4	1.88 %	Manilo	Myroslava	Ahafonova
Lab4	1.82 %	Manilo	Khomik	DP2021
Lab4	1.75 %	Manilo	Leonid	DBrepo
Lab4	1.7 %	Manilo	Nikita	JavaFIT
Lab4	1.51 %	Manilo	Danylo	web
Lab4	1.18 %	Manilo	Illia	JavaProject_by_Illia
Lab4	1.06 %	Manilo	Danylo	ProjectBaida
Lab4	1.05 %	Manilo	Marharyta	RubanMargo
Lab4	1.0 %	Manilo	Vladyslav	JavaMIT

Рисунок 3.11 – Представлення для контролеру comparison.py

В даному представленні у відсортованому по спаданню вигляді зображені результати порівняння конкретного репозиторію с репозиторіями інших студентів. Ця інформація може стати корисною під час підведення підсумків по оцінкам для викладачів. Наприклад, якщо в стовпчику Percent буде виведено результат більше 70%, то роботу можна вважати не справжньою, яку студент явно робив не самостійно та не без копіювання чужої. Також в даний застосунок в майбутньому може бути додана можливість розгортати проекти локально чи на віддаленому сервері. Для цього потрібно створити додатково контролер, який має зв'язується з віддаленим GitHub Package Registry та робити pull артефакту, який потім автоматично копіювати на сервер. Далі в наступному пункті розділу буде приведено приклади налаштування реєстру артефактів та стадій Github Actions Workflow.

3.3. Приклад налаштування процесів GitHub Actions для репозиторіїв студентів

У даному пункті розділу буде описано процес автоматизації стадій CI за допомогою інструмента GitHub Actions. Ця платформа виконує роль CI серверу, який використовує за основу ОС Ubuntu, яка вказується декларативним описом в yaml файлу маніфесту в директорії .github/workflows/. У даному прикладі буде показано, як завдання GitHub Action можуть автоматично запускатися, де вони виконуються та як вони можуть взаємодіяти з кодом у репозиторіях студентів. Для належного виконнання всіх стадій CI в GitHub Actions, студентам перед цим потрібно запустити весь цикл Maven на локальному комп'ютері та описати конфігурацію maven проекту в pom.yaml.

Нищевказані залежності потрібні для виконання тестів Junit 3-5 проекту Maven. Також в придаток до них потрібно вказати плагіни maven для компіляції версії не нижче 3.1 та плагін maven для створення артефакту WAR. На рис 3.12 зображений сніпет розмітки xml для конфігурації проекту maven та його залежності для стадій компіляції та створення артефакту.

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.1</version>
    <configuration>
        <source>1.7</source>
        <target>1.7</target>
        <compilerArguments>
            <endorseddirs>${endorsed.dir}</endorseddirs>
        </compilerArguments>
    </configuration>
</plugin>
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-war-plugin</artifactId>
    <version>2.3</version>
    <configuration>
        <failOnMissingWebXml>false</failOnMissingWebXml>
    </configuration>
</plugin>
<plugin>
```

Рисунок 3.12 – Залежності maven проекту, які потрібні для виконання стадій компіляції в GitHub Actions Workflow

Далі наведено найважливіші етапи життєвого циклу збірки за замовчуванням:

- validate - перевірити, чи доступна вся інформація, необхідна для збірки;
- compile - компіляція коду;
- test-compile - скомпілюйте тестовий код;
- test – запуск усіх модульних тестів;
- package – скомпілювати вихідний код у формат, що розповсюджується;
- integration-test - обробити та розгорнути пакет, якщо потрібно для запуску інтеграційних;
- install - встановити пакет до локального сховища розгортання;
- deploy – завантажити пакет до спеціального регістру GitHub Package.

Усі стадії, окрім integration-test та deploy необхідно запустити локально, а вже потім сконфігурувати конвейер в файлі з роширенням .yaml для запуску на CI сервері. Це потрібно для попередньої перевірки перед тим як робити коміт та завантажувати код до віддаленого репозиторію та потім запускати стадії віддалено. Локально потрібно запустити всі стадії окрім двох останіх site і deploy. На рис 3.13 зображений цикл.

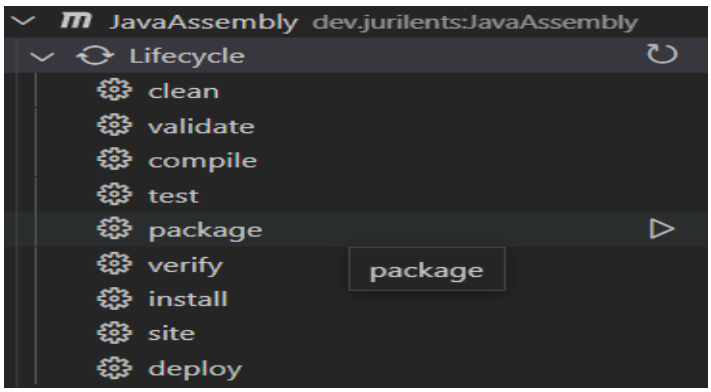


Рисунок 3.13 – Стадії, які необхідно запустити локально перед виконанням git push

Після цих всіх дій потрібно описати manifest з розширенням yaml. А в ньому вказати дані стадії. Також попередньо потрібно задати image ОС та налаштування запуску даних стадій, а саме тригери. На рис 3.14 зображений вміст маніфесту файлу .github/workflows/main.yaml, а саме сніпет коду для двох стадій test та build.

```yaml
name: Java CI with Maven

on:
  push:
    branches: ["**"]

jobs:
  validate:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2
    - name: Set up JDK 11
      uses: actions/setup-java@v2
      with:
        java-version: '11'
        distribution: 'adopt'
    - name: Validation with Maven
      run:  mvn validate -f pom.xml

  build:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2
    - name: Set up JDK 11
      uses: actions/setup-java@v2
      with:
        java-version: '11'
        distribution: 'adopt'
    - name: Build with Maven
      run:  mvn compile -f pom.xml


  test:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2
    - name: Set up JDK 11
      uses: actions/setup-java@v2
      with:
        java-version: '11'
        distribution: 'adopt'
    - name: Running Tests with Maven
      run:  mvn test -f pom.xml
```

Рисунок 3.14 – Декларативний опис маніфесту main.yaml для стадій build та test

На рис 3.15 зображений вміст маніфесту файлу .github/workflows/main.yaml, а саме сніпет коду для двох стадій package та deploy.

```
package:
  runs-on: ubuntu-latest
  steps:
  - uses: actions/checkout@v2
  - name: Set up JDK 11
    uses: actions/setup-java@v2
    with:
      java-version: '11'
      distribution: 'adopt'
  - name: Packaging with Maven
    run:  mvn package -f pom.xml && ls -la target


deploy:
  runs-on: ubuntu-latest
  steps:
    - uses: actions/checkout@v2
    - name: Set up JDK 11.0.2
      uses: actions/setup-java@v1
      with:
        java-version: 11.0

    - name: Deploy
      run: sed -i 's/personal-access-token/${{ secrets.GH_PACKAGE_REPO_PASSWORD }}/g' settings.xml && sed -i 's/username/denaumenko/g' settings.xml && mvn -B -Pgithub deploy
      env:
        GITHUB_TOKEN:  ${{ secrets.GH_PACKAGE_REPO_PASSWORD }}
```

Рисунок 3.15 – Декларативний опис маніфесту main.yaml для стадій package та deploy

Дві останні стадії ствоюють та завантажують артефакт до віддаленого сховища GitHub Package Registry. Далі ці артефакти можна розгорнути на будь якому сервері чи на локальному компёютері. На рис 3.16 зображений інтерфейс реєстру пакетів jar maven з завантаженими артефактами лабораторних робіт після відпрацювання всіх стадій конвейеру CI/CD.

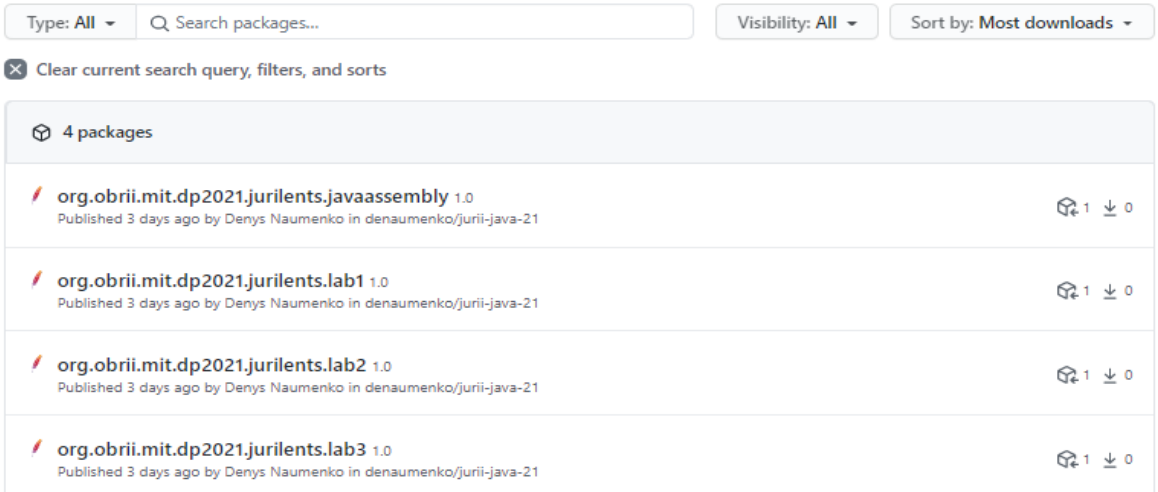


Рисунок 3.16 – Кінцевий результат роботи GitHub Actions Workflow у вигляді готових до розгортання артефактів

Висновки до розділу 3

Автоматизація оцінювання та розгортання лабораторних та практичних робіт була реалізована за допомогою створення, налаштування та подальшою інтеграцією наступних компонентів:

- СУБД PostgreSQL;
- GitHub API;
- GitHub VCS;
- GitHub Actions;
- GitHub Package Registry;
- Maven;
- Flask веб-застосунок;
- Go скріпти.

Було описано три сутності в моделі ORM для генерації схеми таблиць PostgreSQL для зберігання даних про студентів та репозиторії, а також для зберігання інформації про завдання. Також був розроблений застосунок, який дає можливість збирати статистику по репозиторіям. Це було реалізовано завдяки використанню бібліотеки та сервісів GitHub API. Також даний застосунок здатен робити порівняння репозиторії та виводити процент плагіату. Для цілей порівняння даним застосунком використовуються скріпти написанні на мові програмування Go. Вони ітеративно проходяться по всім директоріям репозиторію та порівнюють кожний файл даного проекту з фалом іншого репозиторію, а потім передають ці данні в представлення, а також генерують файл звіт в csv форматі.

Для беспосередньо налаштування CI/CD процесів були використані компоненти дистрибутива контролю версій GitHub VCS, GitHub Actions, GitHub Package Registry та інструмент автоматизації Maven. Всі вони були інтегровані та налаштовані для створення прикладу, як студенти мають реалізовувати в себе CI/CD процеси.

ВИСНОВКИ

Темпи розвитку сучасних інформаційних технологій призводять до автоматизації певних сфер людської діяльності. На основі цього з’являється все більше різних перспектив для автоматизації різних процесів за допомогою інформаційних технологій. Однією з таких сфер є оцінювання та виконанання лабораторних та практичних робіт. Метою цієї роботи був вибір та дослідження підходів автоматизації процесу перевірки лабораторних та практичних робіт.

Отже, у цій роботі було зроблено:

1. Аналіз предметної області та опис процесу автоматизації на основі технологій безперевної інтеграції та безпервного розгортання. Був вибраний стек технологій для реалізації системи після проектування. Було визначено основні етапи проектування та подальшого створення системи автоматизації.
2. Спроектовано застосунок для перегляду статистики та порівнянння репозиторіїв студентів.
3. Спроектовано БД для зберігання даних про студентів та репозиторії.
4. Спроектовано стадії конвейеру CI/CD.
5. Спроектовано взаємодію різних компонентів системи автоматизації.
6. Реалізовано застосунок на мові програмування Python з використанням стороніх бібліотек та фреймворків. В цьому застосунку був реалізований зручний інтерфейс з можливістю перегляду статистики та порівнянням
7. Реалізовано СУБД для зберігання інформації про студентів та їх репозиторії. Дана БД забезпечує належне та швидке занесення даних за допомоги застосунку.
8. Було налаштовано конвейер CI/CD для прикладу, як його повинні налаштовувати студенти.
9. Було проведено дослідження продуктивності конвейерів CI/CD для різних проектів на різних мовах програмування.

ПЕРЕЛІК ПОСИЛАНЬ

1. Martin Fowler, Continuous Integration [Електронний ресурс]. – Режим доступу: https://www.martinfowler.com/articles/continuousIntegration.html

2. Rafał Leszko Continuous Delivery with Docker and Jenkins: Create Secure Applications by Building Complete CI/CD Pipelines, 2nd Edition. [Електронний ресурс]. – Режим доступу: https://www.perlego.com/book/526974/continuous-delivery-with-docker-and-jenkins-pdf

3. David Farley, Jez Humble, Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation. [Електронний ресурс]. – Режим доступу:http://www.dccia.ua.es/dccia/inf/asignaturas/MADS/lecturas/10_Continuous_Delivery_Dzone_Refcardz.pdf

4. Sander Rossel, Continuous Integration, Delivery, and Deployment: Reliable and Faster Software Releases with Automating Builds, Tests, and Deployment [Електронний ресурс]. – Режим доступу: http://tisten.ir/wp-content/uploads/2019/02/Sander-Rossel-Continuous-Integration-Delivery-and-Deployment-Packt-2017.pdf

5. Nikhil Pathania, Pro Continuous Delivery: With Jenkins 2.0 [Електронний ресурс]. – Режим доступу: https://www.oreilly.com/library/view/pro-continuous-delivery/9781484229132/

6. Mikael Krief, Building a good CI/CD pipeline [Електронний ресурс]. – Режим доступу:https://subscription.packtpub.com/book/cloud_and_networking/9781838642730/15/ch15lvl1sec128/building-a-good-ci-cd-pipeline

7. Alan Hohn, Hands-On Continuous Integration and Delivery with Jenkins X and Kubernetes [Електронний ресурс]. – Режим доступу: https://subscription.packtpub.com/video/cloud_and_networking/9781838982065

8. Shalabh Aggarwal, Flask Framework Cookbook - Second Edition [Електронний ресурс]. – Режим доступу: https://subscription.packtpub.com/book/web_development/9781789951295

9. Jack Chan, Ray Chung, Jack Huang, Python API Development Fundamentals [Електронний ресурс]. – Режим доступу: https://subscription.packtpub.com/book/web_development/9781838983994

10. Gaston C. Hillar, Hands-On RESTful Python Web Services - Second Edition [Електронний ресурс]. – Режим доступу: https://subscription.packtpub.com/book/application-development/9781789532227

11. Mihalis Tsoukalos, Mastering Go - Second Edition [Електронний ресурс]. – Режим доступу: https//subscription.packtpub.com/book/programming/9781838559335

12. Naren Yellavula, Hands-On RESTful Web Services with Go - Second Edition [Електронний ресурс]. – Режим доступу: https://subscription.packtpub.com/book/web_development/9781838643577

13. Luca Ferrari, Enrico Pirozzi, Learn PostgreSQL [Електронний ресурс]. Режим доступу: https://subscription.packtpub.com/book/data/9781838985288

14. Vallarapu Naga, Avinash Kumar, PostgreSQL 13 Cookbook [Електронний ресурс]. Режим доступу: https://subscription.packtpub.com/book/data/9781838648138

15. Kenneth Geisshirt, Emanuele Zattin, Aske Olsson, Rasmus Voss, Git Version Control Cookbook - Second Edition [Електронний ресурс]. – Режим доступу: https://subscription.packtpub.com/book/application-development/9781789137545