4 РОЗГОРТАННЯ РОЗРОБЛЕНОЇ РОЗПОДІЛЕННОЇ СИСТЕМИ НАВЧАННЯ UCODE У ХМАРІ

4.1 Система навчання Ucode та її найголовніші компоненти

На прикладі системи LMS школи UNIT.Factory у місті Київ розробляється нова система навчання Ucode, яка буде представлена в інноваційному кампусі НТУ «ХПІ». Одним із головних переваг кампусу будете те, що в ньому буде можливість отримати IT-освіту.

В системі навчання немає вчителів, студенти навчаються за допомогою методологічного підходу Challenge based learning. Велику роль відіграє Peer-to-peer, тобто студенти вчать одне одного. Якщо навчився технології самостійно, то навчи іншого. Система навчання побудована на здачі проектів, які тісно зв’язані з реальним світом. В Ucode немає домашніх завдань, ніхто не пише конспектів. Головне мати мотивацію, спрагу до знань та вміти думати.

Для успішного функціонування платформи Ucode необхідно кілька умов: LMS повинна бути розгорнута в хмарі, щоб мати можливість бути доступною з різних кластерів, а також повинна бути налаштована робота окремих мікросервісів що знаходяться на локальному сервері, до яких звертається LMS на проміжних етапах свого функціонування (рис. 4.1).

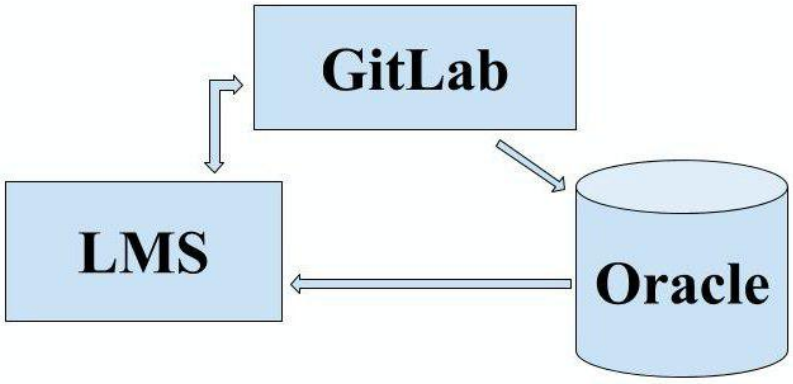

Рисунок 4.1 – Концептуальна модель платформи Ucode

4.1.1 LMS (Learning Management System)

LMS грає ключову роль у взаємодії кінцевого користувача і платформи Ucode. Це основний компонент для коммунікації студента та платформи, тобто студент через веб-додаток LMS може здійснювати майже всі операції необхідні для навчання, а саме: реєструватися на челленджі, брати завдання для них, отримувати підказки щодо виконання завдань, відстежувати свій робочий час, автоматично створювати та отримувати доступ до репозиторія, створювати та виконувати перевірки інших студентів, відстежувати свій прогресс та дізнаватися про майбутні події.

Також у свою чергу адміністратор має можливість сконфігурувати усі вищевказані пункти.

Наведені вище основні можливі дії для студента систематизовані у вигляді діаграми прецедентів на рисунку 4.2

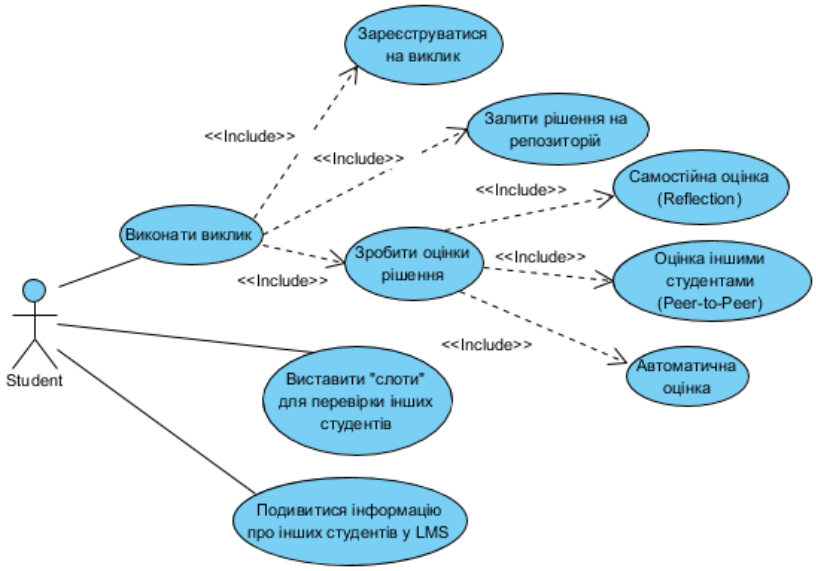

Рисунок 4.2 – Діаграмма прецедентів студента

Інший рівень доступу - адміністратор. Він повністю керує створенням та корегуванням виклику, додаванням нових користувачів та інше. Діаграмма прецедентів для адміністратора показана на рисунку 4.3.

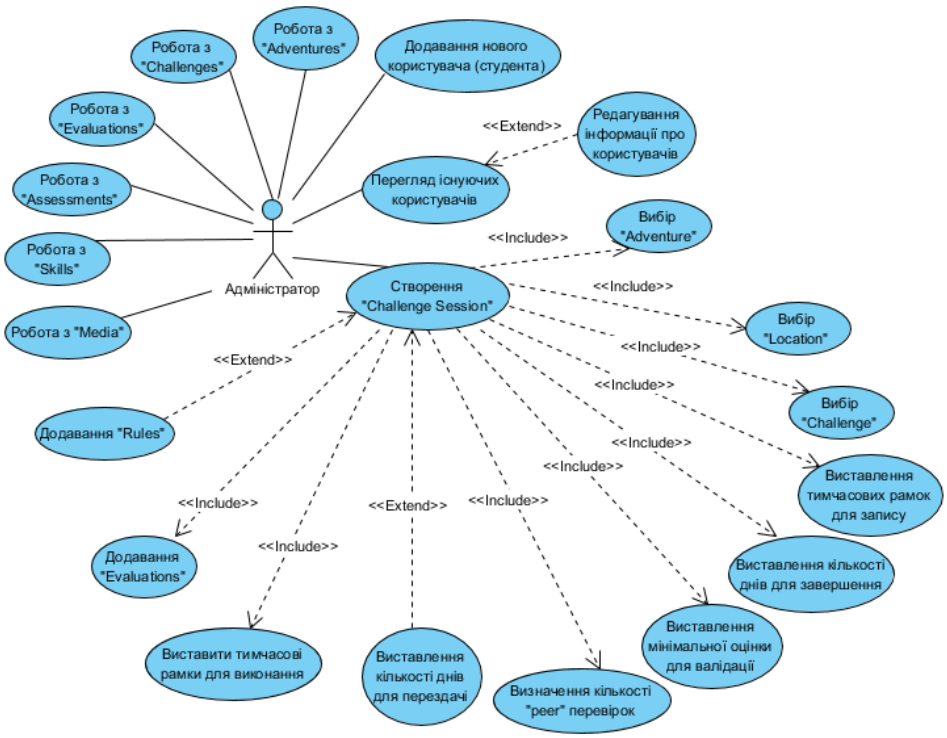

Рисунок 4.3 – Діаграмма прецедентів адміністратора

4.1.2 Oracle (Автоматична перевірка завдань)

Так як Ucode це все ж таки розподілена система, яка не тільки дає змогу на виконання, реєстрування і перевірки іншими пірами завдань а ще й надає свою авторитетну автоматичну оцінку, то неможна не розглянути сервіс Oracle (Автоматична перевірка завдань. Немає відношення до американскої компанії Oracle), який здійснює перевірку студентського коду на працездатність, правильність виконання і плагіат. Так як це ресурсомістке завдання, цей мікросервіс, точно так само як і механізм зберігання облікових записів студентів, знаходиться на локальному фізичному сервері. Особливість мікросервісної архітектури полягає в тому, що цим сервісам не потрібно обмінюватися між собою проміжними даними, тобто кожен компонент системи - це окремий самостійний блок, готовий приймати і відправляти вже оброблені дані.

4.2 Огляд роботи LMS, оцінювання та тестування виклику

4.2.1 Огляд роботи з LMS

Вище було описно, що робота з LMS призначена для студентів, які навчаються та спеціальних користувачів - адміністраторів, які керують системою та коригують її. На рисунку 4.4 зображен профіль студента.

Логін студента формується таким чином: береться перша літера імені та повне прізвище. У профілі студента відображається інформація про нього, його календар відвідуваності навчального закладу, поточний рівень, завершенні виклики та ті, які в прогресі. Також можно побачити діаграмму навичок та частину, де пишуться перевірки, які відбудуться. На рисунку 4.5 зображена панель адміністратора.

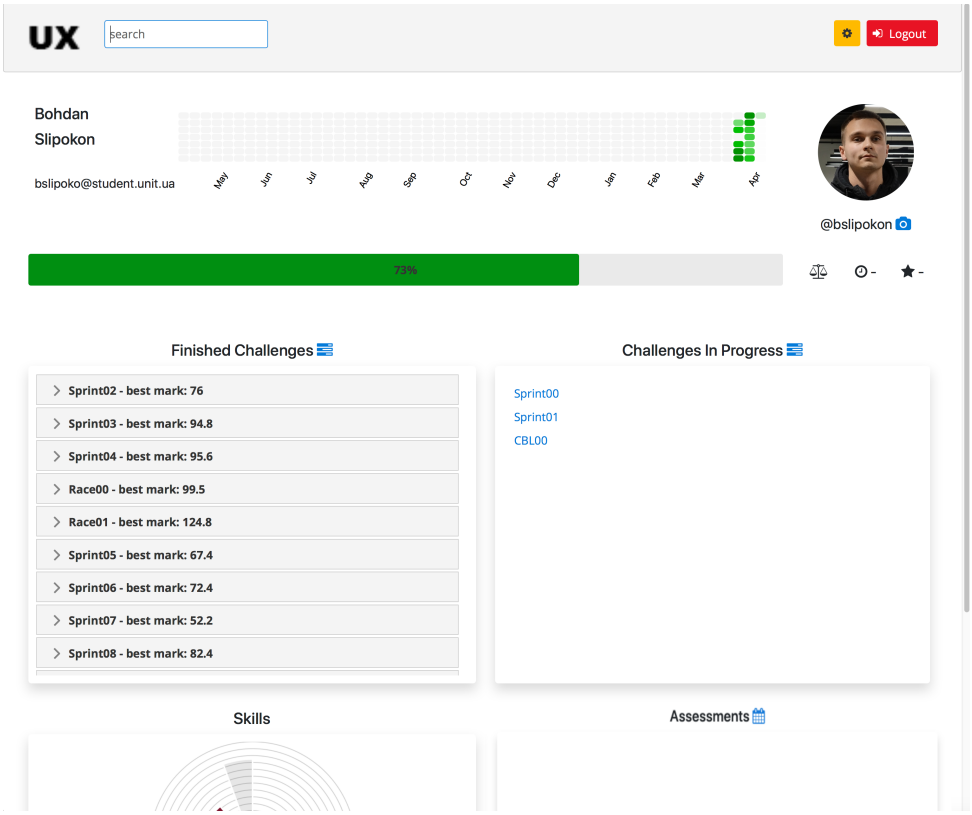

Рисунок 4.4 – Профіль студента у системі

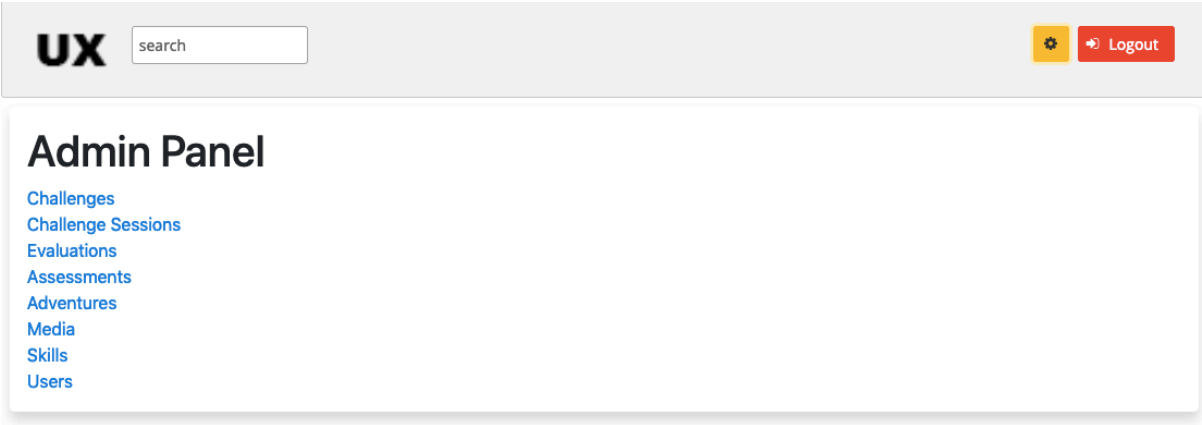

Рисунок 4.5 – Панель адміністратора

4.2.2 Тестування та оцінювання виклику

Як вже згадувалося в системі існує три типи тестування. Перша перевірка виклику - це Reflection. Студент відповідає на питання відносно виконаної роботи. Приклад форми зображен на рисунку 4.6.

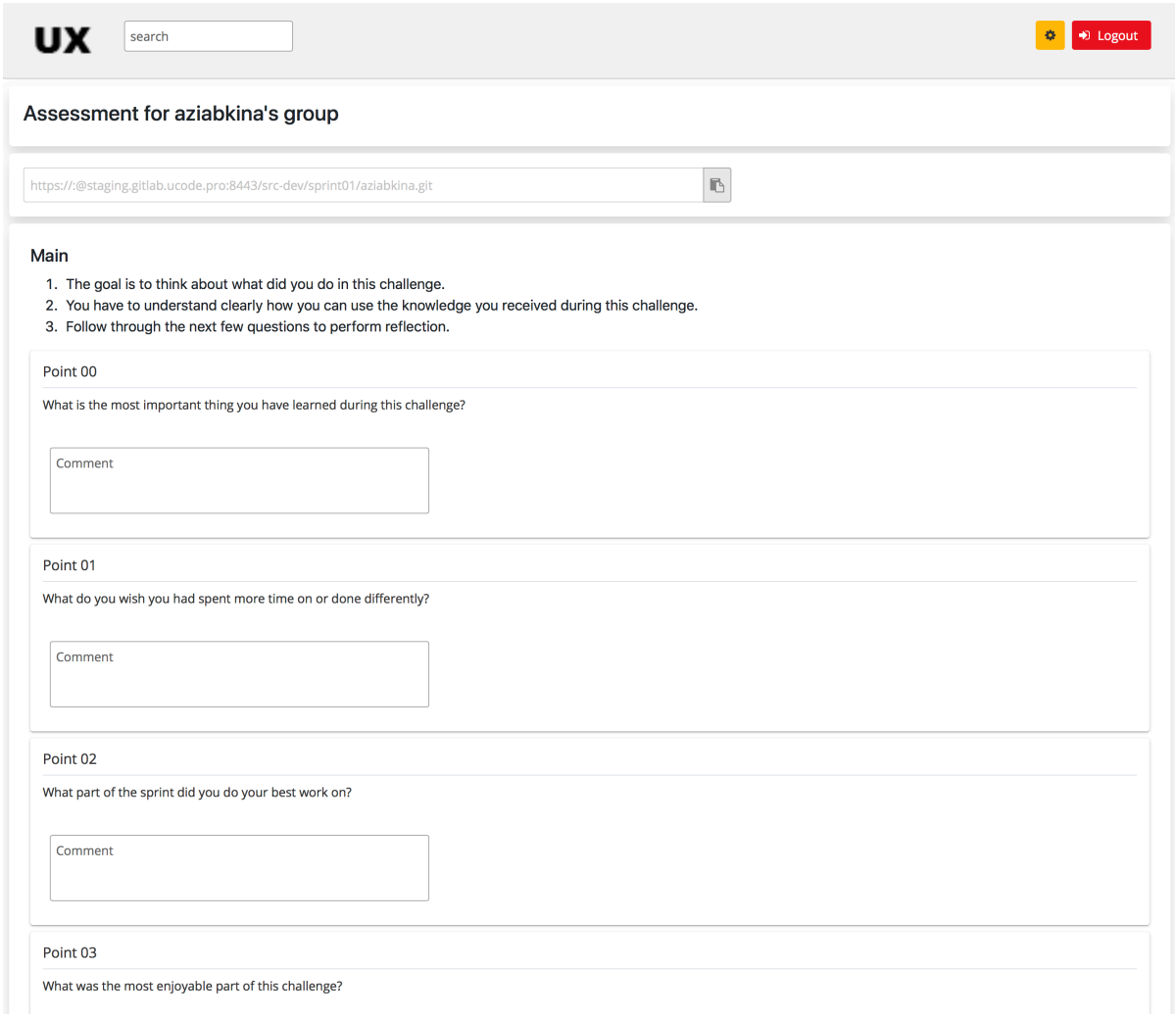

Рисунок 4.6 – Форма перевірки «Reflection»

Наступний етап - Peer evaluation. Перевірка, яка відбувається між студентами. Зазвичай потрібно зробити від двух до п’яти перевірок, це залежить від складності виклику. Щоб студент зміг записатися на перевірку він потрібен мати «tokens». З кожною пройденою перевіркою вони віднімаються у студента, який здає виклик, а той у студента-перевіряючого «tokens» додаються. Коли починається перевірка студентам пропонується форма, за якою потрібно перевіряти проект (рис. 4.7).

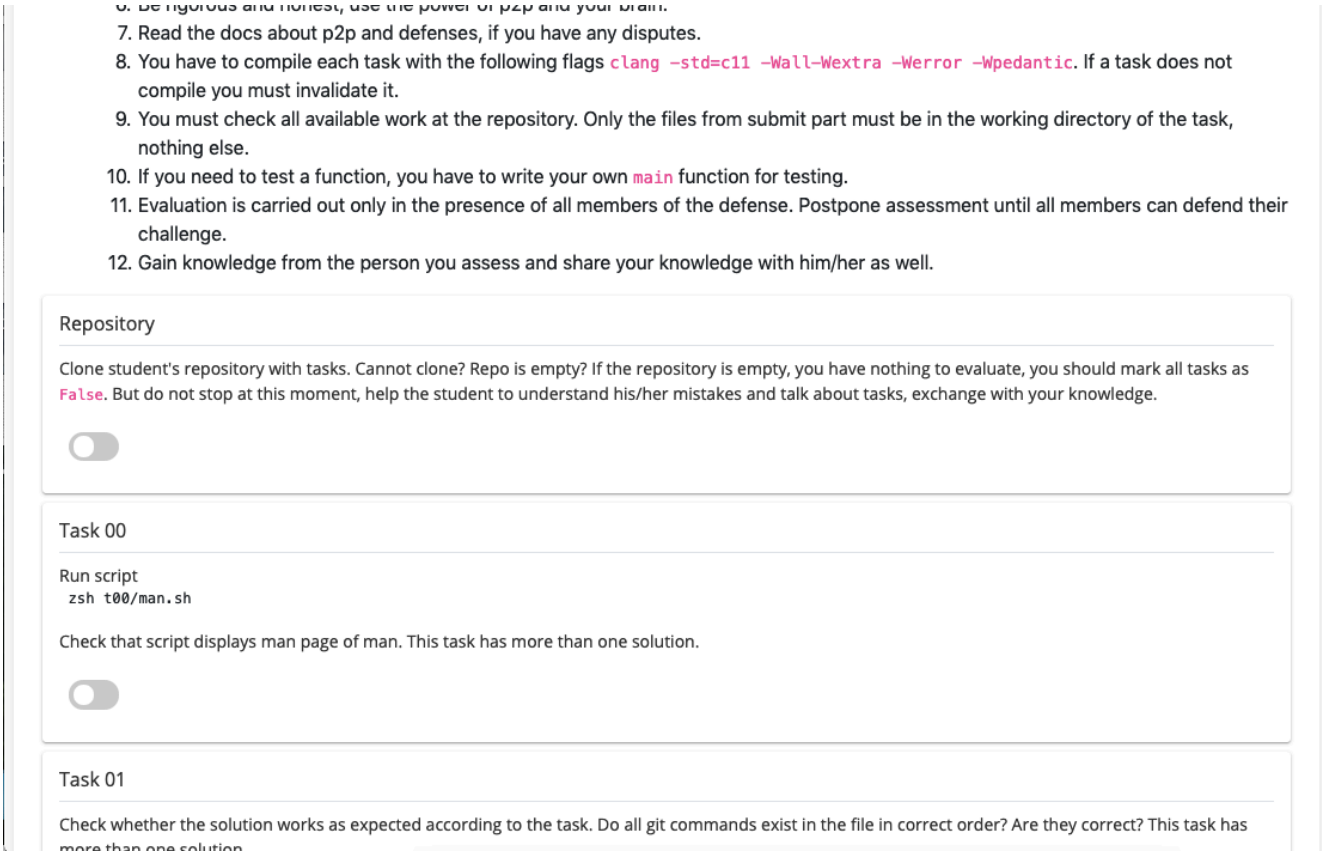

Рисунок 4.7 – Форма перевірки «Peer evaluation»

Наступний й останній етап - Oracle. Автоматичні тести, які запускаються на окремих комп’ютерах. Коли настає час перевірки система відправляє репозиторій студента, на якому зберігається його рішення, до Oracle. Система автоматичних перевірок у свою чергу клонує вміст репозиторію та запускає тести.

Після завершення автоматичної перевірки оцінка відправляється до LMS.

Останній етап формування оцінки заснований на підрахунку усіх трьох перевірок. Відповідно до ваги кожної з перевірок підраховується та виставляється бал. Таким чином, формується оцінка і відображається у профілі студента (рис. 4.8).

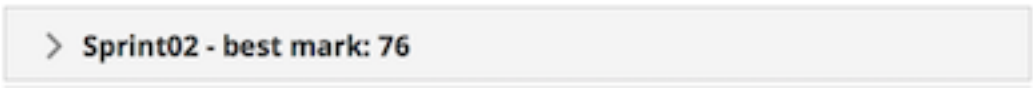

Рисунок 4.8 – Відображення фінальної оцінки

4.3 Провайдер

Найбільш за все для цього проекту підішов хмарний провайдер Digital Ocean. Ця платформа хмарних обчислень дозволяє мати:

1) виділений сервер баз даних. Система управління базами даних (СУБД) може бути відокремлена від решти оточення, щоб виключити конкуренцію за ресурси сервера між додатком і базою даних і посилити безпеку, прибравши базу даних з DMZ, загальнодоступного інтернета. Приклад використання: добре підходить для швидкого розгортання програми;

2) балансувальник навантаження (зворотний проксі). Балансувальник навантаження можуть бути додані в оточення сервера для збільшення продуктивності і надійності шляхом розподілу навантаження між декількома серверами. Якщо один з серверів впаде, то інші сервери будуть обробляти вхідний трафік, поки впавший сервер знову не запрацює. Балансувальник навантаження так само може бути використаний для обслуговування декількох додаток через один і той же домен і порт за допомогою зворотного проксі.

Приклади програмного забезпечення, що підтримує зворотний проксі: HAProxy, Nginx, і Varnish;

3) HTTP Accelerator (кешуючий зворотний проксі). HTTP accelerator, або кешуючий HTTP-запити зворотний проксі-сервер, може бути використаний для зменшення часу, необхідного для надання контенту користувачеві, за допомогою різних методів. Основою метод, який

використовується з HTTP accelerator, це кешування відповідей від веб-сервера або сервера додатків в пам'яті, при цьому наступні запити на той же самий контент можуть бути оброблені швидко і меншою кількістю зайвих взаємодій з веб-сервером або сервером додатків;

4) реплікацію бази даних по схемі провідний-ведений (Master-Slave). Одним із способів поліпшення продуктивності системи бази даних, до якої запросив на читання відбувається набагато більше, ніж на запис, як, наприклад, в системах управління контентом (CMS), є використання реплікації бази даних по схемі провідний-ведений. Така схема припускає наявність одного ведучого і одного і більше відомих вузлів. В такому випадку, всі записи направляються на провідний вузол, а запити на читання можуть бути розподілені між усіма вузлами.

Так само важливо відзначити що є можливість створювати VM і розпоряджатися їх ресурсами.

4.4 Контейтерізація

Для реалізації підходу CI/CD в хмарі було запущено програмне забезпечення Docker (рис. 4.9), що дозволяє відокремити додаток від інфраструктури і звертатися з інфраструктурою як керованим додатком. Docker допомагає викладати код швидше, швидше тестувати, швидше викладати додатки і зменшити час між написанням коду і запуску коду. Docker робить це за допомогою легковагій платформи контейнерної віртуалізації, використовуючи процеси і утиліти, які допомагають керувати і викладати ваші програми.

У своєму ядрі Docker дозволяє запускати практично будь-який додаток, безпечно ізольоване в контейнері. Безпечна ізоляція дозволяє запускати на одному хості багато контейнерів одночасно. Легка природа контейнера, який запускається без додаткового навантаження гипервизора, дозволяє домогтися більше від заліза.

LMS - і всі її компоненти, включаючи базу даних під керуванням СУБД PostgreSql знаходяться в окремому контейнері. Так само для реалізації

підходу CI / CD, який відмінно підтримується GitLab, був розміщений другий контейнер, з entity GitLab.

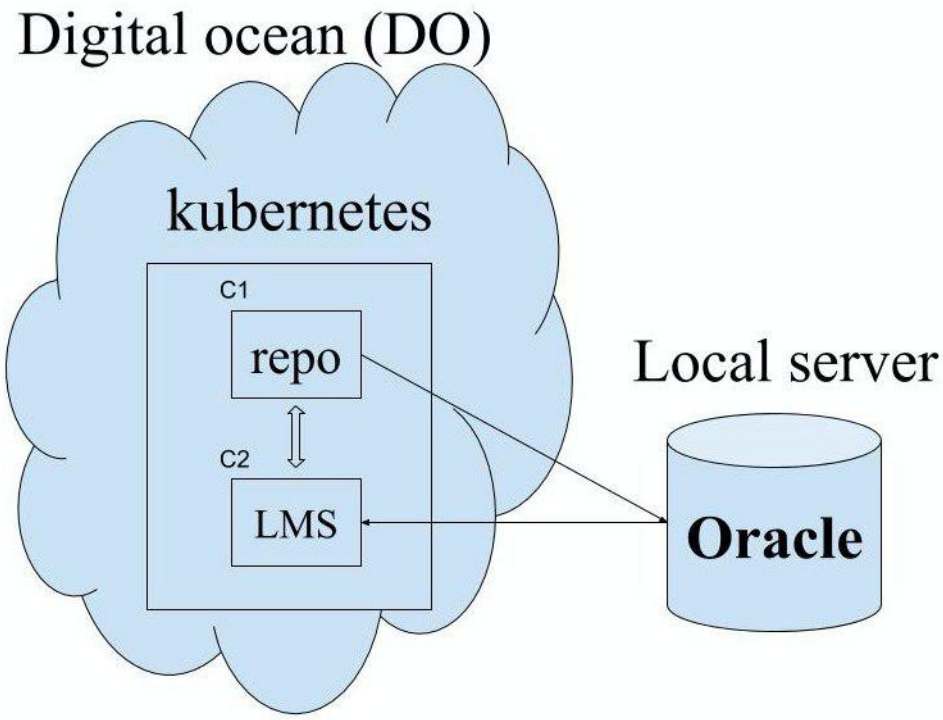

Рисунок 4.9 – Розміщення компонентів Ucode в продакшн

4.5 Конфігурування автоматичного запуску тестів та деплою змін в контейнер

GitLab CI також дозволяє вам налаштовувати безперервну інтеграцію з використанням будь-якого з образів Docker, доступного на Docker Hub.

4.5.1 GitLab CI YML

GitLab CI використовує YAML файл .gitlab-ci.yml для визначення конфігурацій проекту, що включають в себе визначення всіх етапів, які будуть виконуватися після того, як конвеєр CI/CD запускається у відповідь на git push/merge. У цьому прикладі буде проводитися unit-тест над простим Node.js модулем системи LMS, щоб переконатися, що в коді немає помилок. Приклад зображено на рисунку 4.10.

```yaml
stages:
  - lint-css
  - lint-js
  - unit-test

image: node:6.11.2


lint css:
  stage: lint-css
  before_script:
```

Рисунок 4.10 – YAML-файл конфігурації

У наведеному вище файлі конфігурації YAML весь процес розбитий на 3 етапи. Кожен з етапів це просто gulp.task, заданий в gulpfile.js. Так як у контейнері встановлений Node.js, користувач може окремо запускати будь-який з етапів. Але в GitLab CI потрібно вказати, який з образів Docker вам потрібен. У нашому випадку, це вузол: 6.11.2. Крім того, даний Docker-атрибут можна задати всередині певного етапу, тому можна використовувати різні інструменти для будь-якого з етапів (рис. 4.11).

```yaml
lint css:
  stage: lint-css     # <- Should map to the value defined in the 'stages'
  before_script:      # <- Pre-script
    - npm install
  cache:              # <- Enable caching on files so they are available in the next stage
    untracked: true   # <- Cache the git untracked files (ex. node_modules)
  only:
    - master          # <- This stage run only on master branch update
  script:             # <- The actually script of this stage
    - ./node_modules/gulp/bin/gulp.js lint-css
```

Рисунок 3.11 – YAML-файл конфігурації

Атрибути before_script і script можуть мати кілька значень (array в .yml) .Якщо виконання скрипта завершиться невдачею, весь етап буде класифікований як неправильний.

4.5.2 Запуск Pipeline (процес розробки)

В налаштуваннях, у вкладці Pipeline в меню CI/CD можна побачити всю історію процесу розробки.

Натиснувши на певний Pipeline, є можливість прочитати докладний консольний висновок будь-якого з етапів. Це корисно, коли з'являються збої в роботі (рис. 4.12-13).

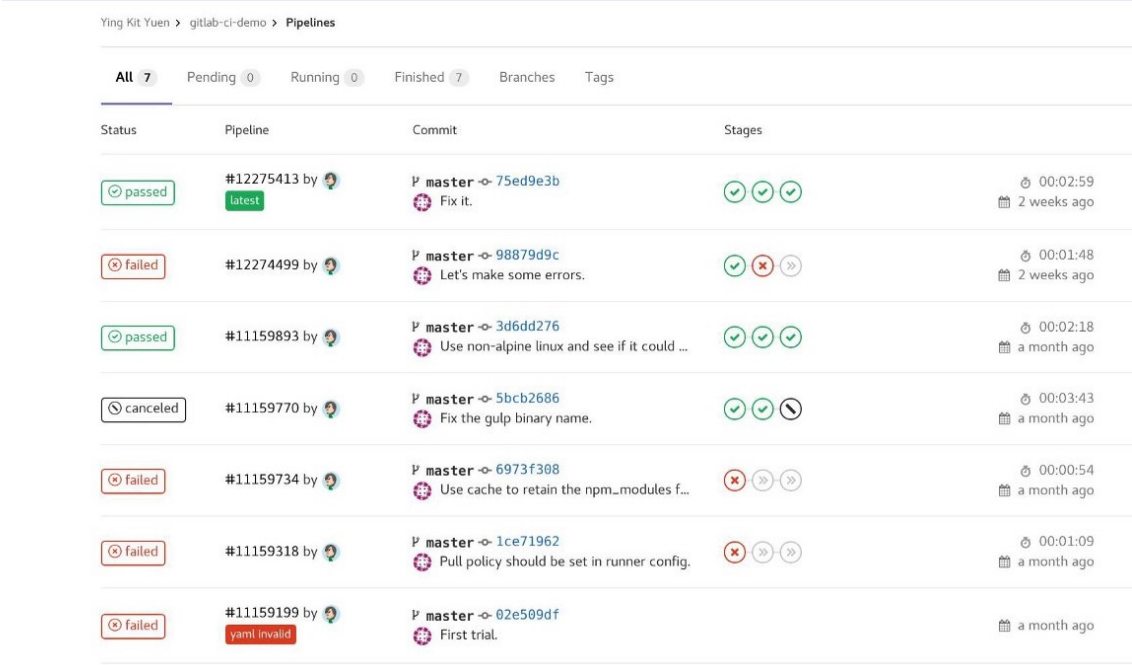

Рисунок 4.12 – Історія пайплайнів

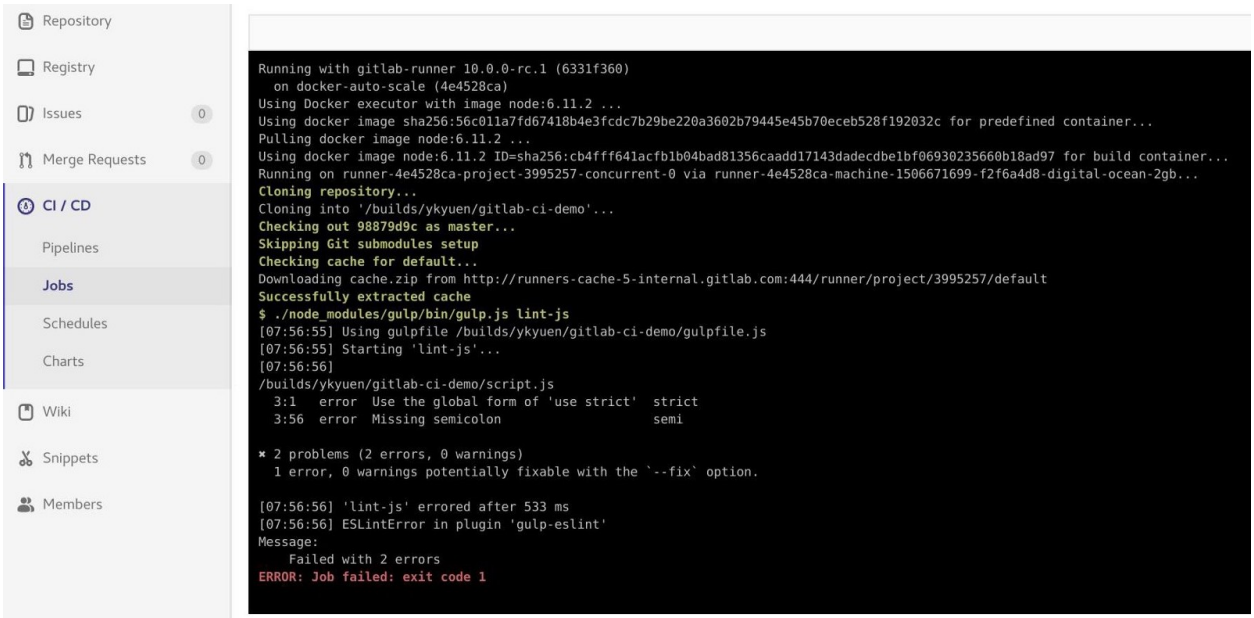

Рисунок 4.13 – Консольний висновок

Після успішної збірки, тестів і викочування в pre-production проводяться додаткові ручні тести нової версії, показ замовнику і інші «знущання» над додатком. Якщо все пройшло успішно, то реліз-інженер запускає завдання approve. Якщо щось пішло не так, то реліз-інженер запускає завдання not approve.

Викочування додатки на production можливий тільки після успішного викочування на pre-production і виконання завдання approve. Викочування на production може запустити реліз-інженер або DevOps-інженер.