

4 РОЗГОРТАННЯ РОЗРОБЛЕНОЇ РОЗПОДІЛЕНОЇ 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 ІТ-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 задачі 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).

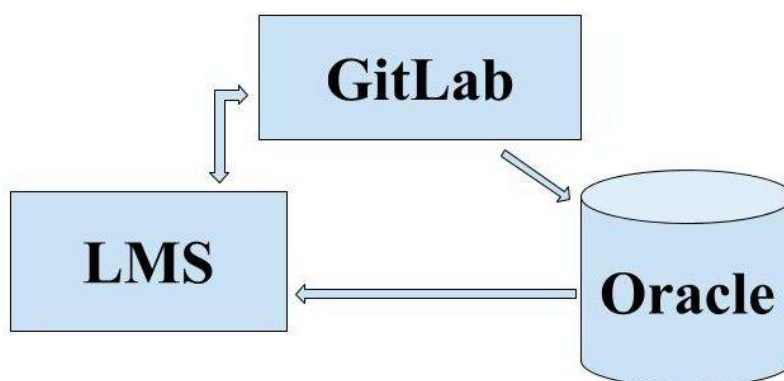


Рисунок 4.1 – Концептуальна модель платформи Ucode

4.1.1 LMS (Learning Management System)

LMS грає ключову роль у взаємодії кінцевого користувача і платформи Ucode. Це основний компонент для комунікації 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 прогрес 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

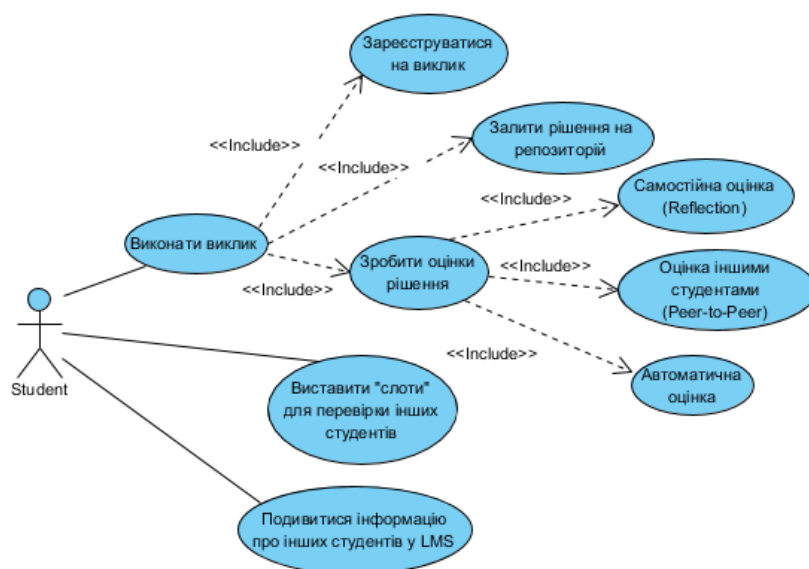


Рисунок 4.2 – Діаграма прецедентів студента

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

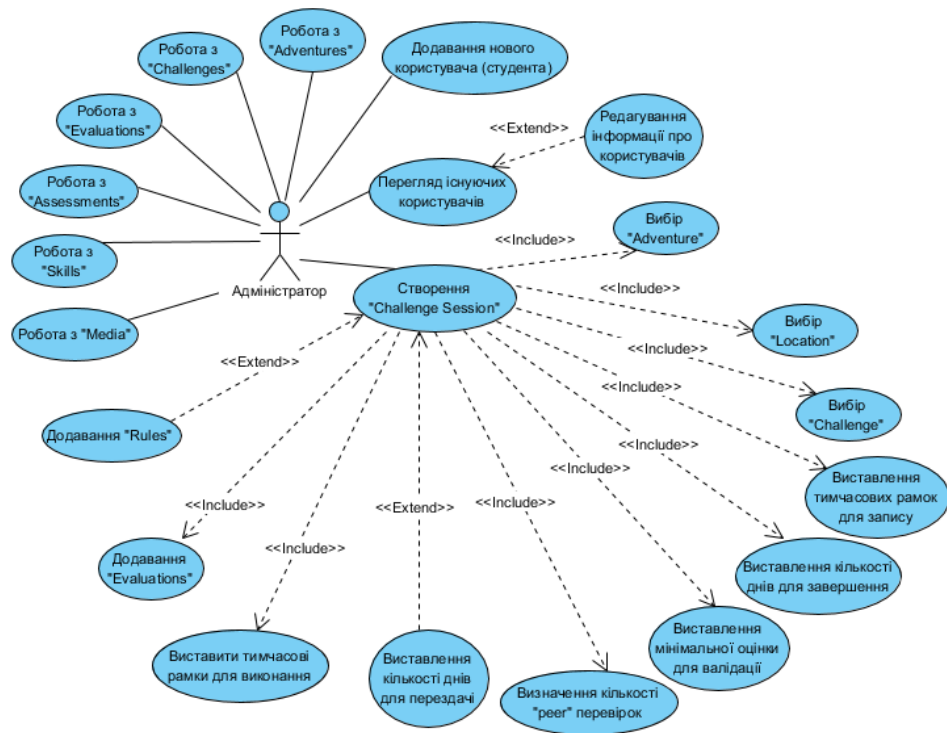


Рисунок 4.3 – Діаграма прецедентів адміністратора

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 американської 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 описано, 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 можна побачити діаграму навичок та частину, де пишуться перевірки, які відбудуться. На рисунку 4.5 зображена панель адміністратора.

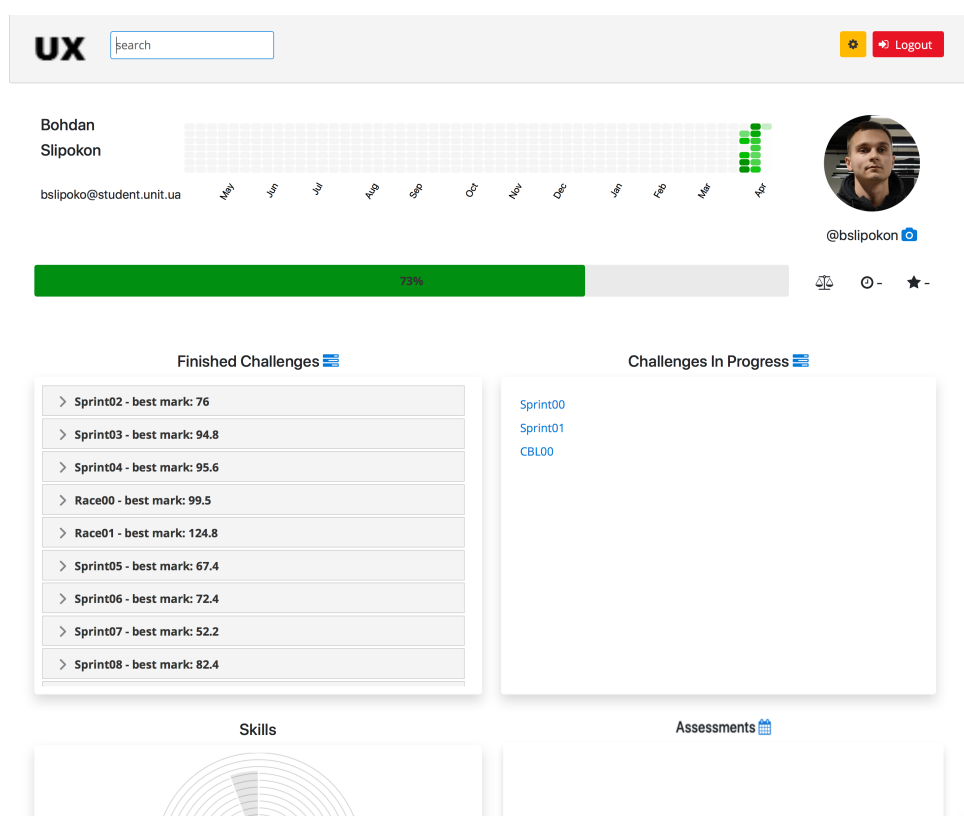


Рисунок 4.4 – Профіль студента у системі

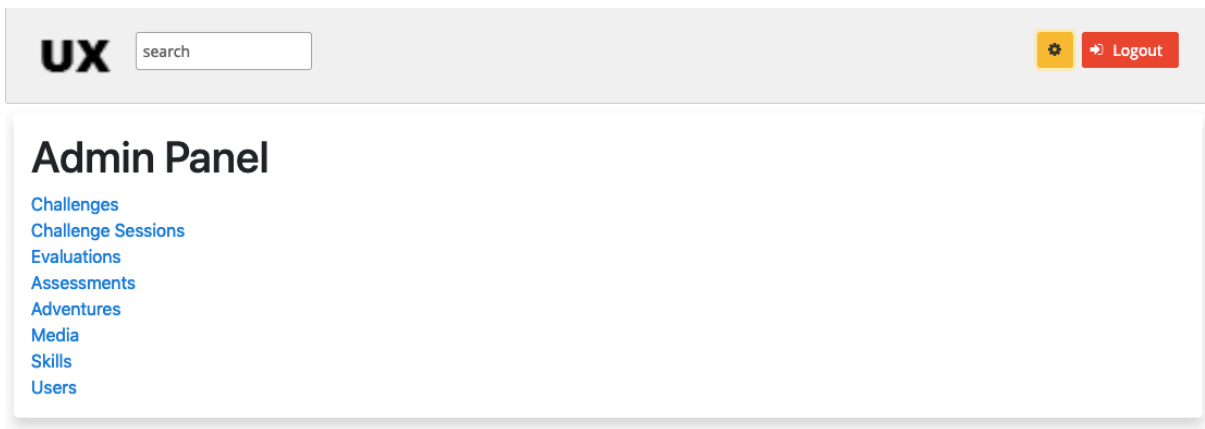


Рисунок 4.5 – Панель адміністратора

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.

The image displays a web form titled 'Assessment for aziabkina's group'. At the top of the form, there is a URL input field containing 'https://staging.gitlab.unicode.pro:8443/src-dev/sprint01/aziabkina.git' and a small icon to its right. Below the URL field, the form is divided into sections. The first section is labeled 'Main' and contains three numbered instructions: 1. 'The goal is to think about what did you do in this challenge.', 2. 'You have to understand clearly how you can use the knowledge you received during this challenge.', and 3. 'Follow through the next few questions to perform reflection.' Following these instructions, there are four distinct assessment points, each with a title and a question, followed by a 'Comment' text input field. 'Point 00' asks: 'What is the most important thing you have learned during this challenge?'. 'Point 01' asks: 'What do you wish you had spent more time on or done differently?'. 'Point 02' asks: 'What part of the sprint did you do your best work on?'. 'Point 03' asks: 'What was the most enjoyable part of this challenge?'. Each question is followed by a large, empty text box for the user's response.

Рисунок 4.6 – Форма перевірки «Reflection»

Наступний етап - Peer evaluation. Перевірка, яка відбувається між студентами. Зазвичай потрібно зробити від двох 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).

6. Be rigorous and honest, use the power of p2p and your brain.

7. Read the docs about p2p and defenses, if you have any disputes.

8. You have to compile each task with the following flags `clang -std=c11 -Wall-Wextra -Werror -Wpedantic`. If a task does not compile you must invalidate it.

9. You must check all available work at the repository. Only the files from submit part must be in the working directory of the task, nothing else.

10. If you need to test a function, you have to write your own `main` function for testing.

11. Evaluation is carried out only in the presence of all members of the defense. Postpone assessment until all members can defend their challenge.

12. Gain knowledge from the person you assess and share your knowledge with him/her as well.

Repository

Clone student's repository with tasks. Cannot clone? Repo is empty? If the repository is empty, you have nothing to evaluate, you should mark all tasks as `False`. But do not stop at this moment, help the student to understand his/her mistakes and talk about tasks, exchange with your knowledge.

☐

Task 00

Run script
`zsh t00/man.sh`

Check that script displays man page of man. This task has more than one solution.

☐

Task 01

Check whether the solution works as expected according to the task. Do all git commands exist in the file in correct order? Are they correct? This task has more than one solution.

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).

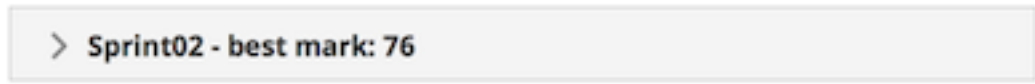


Рисунок 4.8 – Відображення фінальної оцінки

4.3 Провайдер

Найбільш за все для цього проекту підшов 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 інтернету. 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 запити на читання відбуваються 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 Контейнеризація

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 гіпервизора, 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.

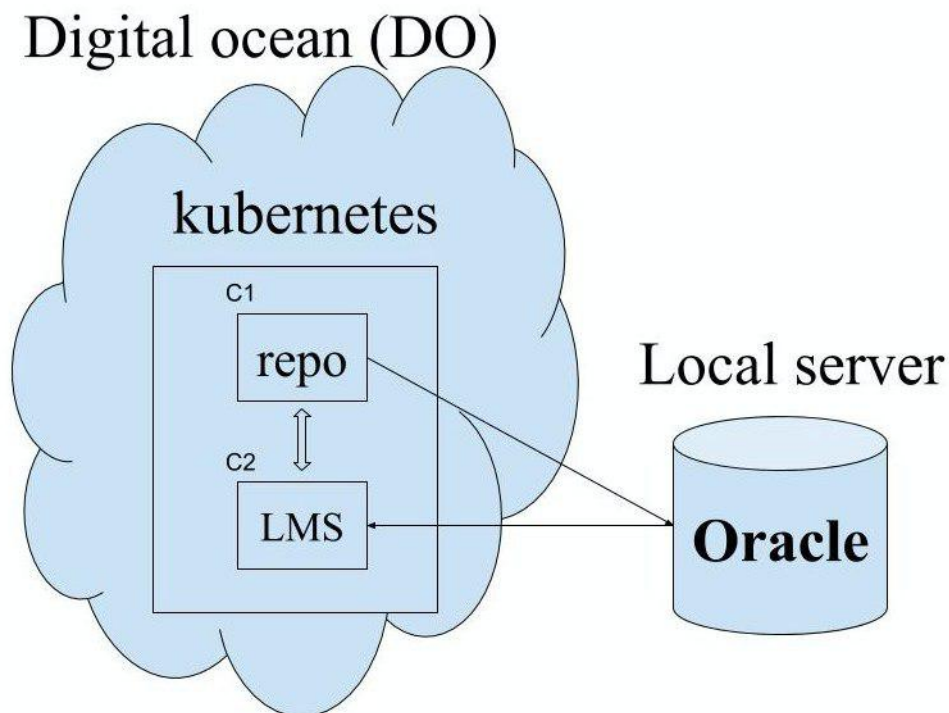


Рисунок 4.9 – Розміщення компонентів Ucode в продакшн

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 YAML

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.

```

1  stages:
2    - lint-css
3    - lint-js
4    - unit-test
5
6  image: node:6.11.2
7
8
9  lint css:
10   stage: lint-css
11   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).

```

1  lint css:
2    stage: lint-css      # <- Should map to the value defined in the 'stages'
3    before_script:      # <- Pre-script
4      - npm install
5    cache:               # <- Enable caching on files so they are available in the next stage
6      untracked: true   # <- Cache the git untracked files (ex. node_modules)
7    only:
8      - master          # <- This stage run only on master branch update
9    script:              # <- The actually script of this stage
10     - ./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 .yaml). 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).

Ying Kit Yuen > gitlab-ci-demo > Pipelines			
All 7 Pending 0 Running 0 Finished 7 Branches Tags			
Status	Pipeline	Commit	Stages
passed	#12275413 by latest	master -> 75ed9e3b Fix it.	00:02:59 2 weeks ago
failed	#12274499 by	master -> 98879d9c Let's make some errors.	00:01:48 2 weeks ago
passed	#11159893 by	master -> 3d6dd276 Use non-alpine linux and see if it could ...	00:02:18 a month ago
canceled	#11159770 by	master -> 5bcb2686 Fix the gulp binary name.	00:03:43 a month ago
failed	#11159734 by	master -> 6973f308 Use cache to retain the npm_modules f...	00:00:54 a month ago
failed	#11159318 by	master -> 1ce71962 Pull policy should be set in runner config.	00:01:09 a month ago
failed	#11159199 by yamlinvalid	master -> 02e509df First trial.	a month ago

Рисунок 4.12 – Історія пайплайнів

Repository

Registry

Issues0

Merge Requests0

CI / CD

Pipelines

Jobs

Schedules

Charts

Wiki

Snippets

Members

```
Running with gitlab-runner 10.0.0-rc.1 (6331f360)
on docker-auto-scale (4e4528ca)
Using Docker executor with image node:6.11.2 ...
Using docker image sha256:56c011a7fd67418b4e3fcdc7b29be220a3602b79445e45b70e0eb528f192032c for predefined container...
Pulling docker image node:6.11.2 ...
Using docker image node:6.11.2 ID=sha256:cb4fff641acfb1b04bad81356caadd17143dade0db1bf06930235660b18ad97 for build container...
Running on runner-4e4528ca-project-3995257-concurrent-0 via runner-4e4528ca-machine-1506671699-f2f6a4d8-digital-ocean-2gb...
Cloning repository...
Cloning into '/builds/ykyuen/gitlab-ci-demo'...
Checking out 98879d9c as master...
Skipping Git submodules setup
Checking cache for default...
Downloading cache.zip from http://runners-cache-5-internal.gitlab.com:444/runner/project/3995257/default
Successfully extracted cache
$ ./node_modules/gulp/bin/gulp.js lint-js
[07:56:55] Using gulpfile /builds/ykyuen/gitlab-ci-demo/gulpfile.js
[07:56:55] Starting 'lint-js'...
[07:56:56]
/builds/ykyuen/gitlab-ci-demo/script.js
 3:1  error  Use the global form of 'use strict'  strict
 3:56 error  Missing semicolon                    semi

* 2 problems (2 errors, 0 warnings)
1 error, 0 warnings potentially fixable with the '--fix' option.

[07:56:56] 'lint-js' errored after 533 ms
[07:56:56] ESLintError in plugin 'gulp-eslint'
Message:
  Failed with 2 errors
ERROR: Job failed: exit code 1
```

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.