НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ І ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Розрахунково-графічна робота

з дисципліни «Інтеграційні програмні системи»

Виконали:
студенти гр. ІО-53
Попов Станіслав
Фірсов Дмитро
Студент гр. ІО-52
Семчишин Віталій

Київ 2018 р.

Короткий опис проекту

Протягом семестру було розроблено веб-додаток WeatherObserver. Основна задача – вимірювання фізичних показників навколишнього середовища (на даний момент реалізовано для температури і вологості) для подальшого їхнього збереження і обробки. Зараз користувач може спостерігати зняті дані, час їхнього вимірювання, у вигляді таблиці та, для більш повного уявлення динаміки змін показників, у вигляді графіків. При цьому можна обирати різні часові інтервали (3, 6, 12 годин, 1 доба, 1 тиждень). Дані оновлюються кожні 10 хвилин, тому за допомогою веб-додатку можна отримати найактуальнішу інформацію. Також можна отримати середнє значення температури та вологості протягом останніх 24 годин.

Для написання проекту були використані такі технології:

- Клієнт: Angular 7.
- Сервер: Spring Boot 2.
- Збереження даних: СУБД Postgresql.
- Вимірювання даних: плата ESP8266, до якої був припаяний датчик температури та вологості DHT22.

Веб-додаток було розгорнуто на домені https://weatherobserver.herokuapp.com/.

Системи збірки у проекті

Для збірки серверної частини було використано Maven 3.6. Готовий jar-файл можна отримати у локальному репозиторії Maven за допомогою команди `mvn install`. До цієї команди автоматично виконаються компіляція (`mvn compile`), тестування за допомогою написаних JUnit-тестів і моків (`mvn test`) та, власне, збірка jar-файлу (`mvn package`). Запустити сервер локально можна дуже легко за допомогою плагіну `spring-boot-maven-plugin`. Він дозволяє отримати jar-файл та виконати його однією командою `mvn spring-boot:run`.

Для збірки клієнтської частини було використано Angular-CLI, який використовує для збірки WebPack. Для генерації статичних файлів було використано команду `ng build`. Додавання флагу `--prod` давало змогу отримати статичні файли, що були стиснуті для використання в production. Для того, щоб запустити сервер, є команда `ng serve`, яка також дозволяє запускати сервер у двох режимах. Дані команди були винесені до package.json файлу як скрипти, тому їх також можна запускати командами `npm run build` та `npm run start/dev/prod` відповідно.

Перелік та опис задач, які виконуються на сервері безперервної інтеграції

Опис для задач, що використовуються на сервері безперервної інтеграції, а саме Travis CI, знаходиться у файлі .travis.yml, що лежить у корені проекту. Розглянемо його структуру.

По-перше, вказано те, що збірка виконується тільки для гілки master. Це є виправданим, оскільки саме master вважається основною prod-гілкою.

По-друге, далі іде матриця, що розділяє збірку на Travis CI на 2 частини (jobs): для серверної та клієнтської частин відповідно. Розглянемо їх більш детально. Основна структура в них дуже схожа, оскільки спочатку йде перелік основних інструментів та сервісів, що уже встановлені на Travis CI з коробки, далі можна виконати власний install потрібних модулів і наприкінці вже іде script для збірки власне додатків. Підтримуються pre- і after- workarounds.

a) Для серверної частини спочатку вказуються основні інструменти (language: java, jdk: oraclejdk8), що вже встановлені. У проекті використовується postgresql 10, якого немає. Тому у файлі install.sh у корені проекту було описано установку та запуск даної СУБД. У воркераунді before_script як раз було встановлено і запущено postgresql 10, налаштовано певні конфігурації для того, щоб сервер міг успішно з'єднатися із СУБД.

До цього часу ми знаходились у корені проекту, але весь вихідний код серверу лежить у папці server. Щоб перейти до папки server виконуємо `cd server`. І наприкінці виконуємо `mvn clean package`, яка, як було описано вище, компілює, виконує тести і збирає серверний jar-файл.

b) Для клієнтської частини все простіше. Мовою із коробки вказуємо node_js, заходимо до папки, де лежить код клієнтської програми, `cd front`, встановлюємо залежні бібліотеки через `npm install` і, нарешті, збираємо проект для отримання статичних ресурсів через `npm run build`, що є рівносильним `ng build`.

Тільки, якщо ці 2 частини (jobs) завершились успішно, збірка вважається вдалою.

Графік, який ілюструє вибрані інтервали для повтору спроб при експоненціальній витримці

Результати зображені на рис.1. Експоненційну витримку було використано для плати. У разі невдалого виконання HTTP POST запиту плата очікує певний період (вказані на графіку) і повторює спробу. Це є дуже корисним у разі наявності великої мережі датчиків. На серверній частині для спілкування з базою було використано JdbcTemplate, який одразу обриває весь пул з'єднань, якщо не може доступитися до СУБД і використовує власну політику отримати нове з'єднання.

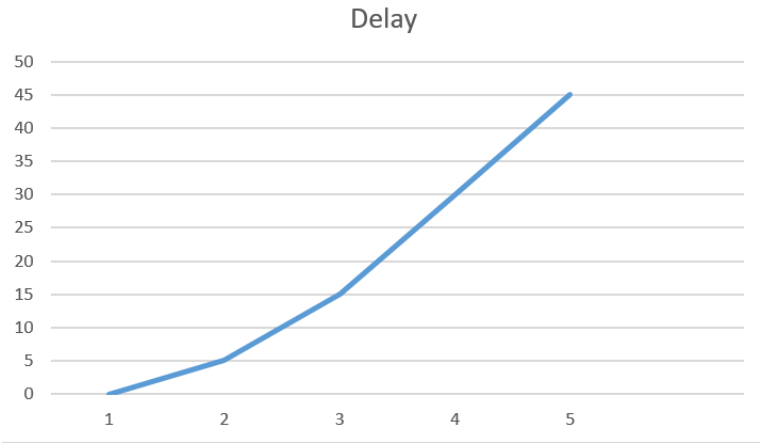

Рис. 1 – графік затримок між спробами при експоненційній витримці

```
♦♦♦Il ♦♦g♦Waiting for connection
[2018-12-28 12:41:40] temperature=3&humidity=99&header=c9bc6960efe4f
[2018-12-28 12:41:45] Retry to send HTTP POST for the 2 time...
[2018-12-28 12:42:00] Retry to send HTTP POST for the 3 time...
[2018-12-28 12:42:25] Retry to send HTTP POST for the 4 time...
[2018-12-28 12:43:10] Retry to send HTTP POST for the 5 time...
[2018-12-28 12:44:35] Server not responding!
```

Рис. 2 – Лог спроб відправлення POST запиту на вимкнений сервер

```
[2018-12-28 12:54:14] temperature=3&humidity=99&header=c9bc6960efe4f
[2018-12-28 12:54:19] Retry to send HTTP POST for the 2 time...
[2018-12-28 12:54:34] Retry to send HTTP POST for the 3 time...
[2018-12-28 12:54:54] HTTP Payload - {"message":measurement was successfully added}
```

Рис. 3 – Лог спроб відправлення POST запиту на вимкнений сервер із його увімкненням після 2-ї невдачі