DESIGN DECISIONS

Project: Public transport lookup application

FONTYS UNIVERSITY OF APPLIED SCIENCES
Mertan Rasim, S3-CB03

Contents

Document versioning

Version	Description	Date
1.0	• Initial version	16-Sep-2022
1.1	• Extended API Endpoints	07-Oct-2022
2.0	• Migrated API documentation to a separate document • Added documentation on separation of concerns	30-Oct-2022
2.1	• Clarified the content, worded differently • Added that dependency injection is being used	04-Nov-2022

3.0	• Expanded documentation to the existing material • Added Introduction	12-Nov-2022
4.0	• Added CI/CD documentation	15-Nov-2022
4.1	• Added reference to the research documentation	19-Nov-2022
5.0	• Added Known vulnerabilities paragraph	09-Dec-2022
5.1	• Updated pipeline • Added Links heading	14-Dec-2022
5.2	• Merged ‘How concerns are separated’ with ‘Use of SOLID principles’	15-Dec-2022
5.3	• Extended ‘Use of SOLID principles’ • Added ‘Project overview’	16-Dec-2022
5.4	• Added ‘Test’ heading • Added ‘Docker’ heading	19-Dec-2022
6.0	• Updated Docker heading • Updated Testing heading to include the frontend tests • Updated C4	13-Jan-2022SS

Introduction

The following document is concerned with explaining in depth the project's architecture, reasoning involved, goals, and scope. All available documentation can be found under the "doc" directory in PDF format.

If you are more interested in the project's requirements, organisation & execution, you can find useful information in the Project plan. An overview of the system's architecture is available in the C4 diagram (see https://c4model.com regarding C4 diagrams), along with the UML diagram for a close overview of the code. API documentation is provided in the form of PDF and Swagger UI. Furthermore, a research paper is also available, regarding the usage of relational over non-relational database system. Lastly, there's a Links heading if you want see references to other documentation.

Technologies in use

To justify the decisions for using the specified technology stack, one of the methods from the DOT framework is used – Available product analysis. In short, available technologies are presented based on their features, resources, and availability. You can find more information in the research paper dedicated for this use case.

1. React.js – JavaScript framework, concerned with the user interface of the web application.
    a. Available resources to use by the teaching institute
    b. Large ecosystem of software libraries to choose from
2. Material UI – React-based UI/UX framework for creating aesthetic interfaces.
    a. Personal choice
    b. Reasonably large amount of documentation
3. Java – Object-oriented language, recommended by the teaching institute to learn and develop with.
    a. Required by the teaching institute
    b. Large ecosystem of software packages to choose from
4. Spring Boot – Java web framework.
    a. Required by the teaching institute
5. Spring Security – Spring framework concerned with security.
    a. Required by the teaching institute
6. Hibernate – Standard object relational mapper (ORM) for Java, used for managing database persistence.
    a. Ease of use
7. MySQL – Relational database management system used for storing data.
    a. Main research can be found in the Research paper of this project
    b. Familiarity with the technology
8. SonarQube – Quality assurance software, integrated to the pipeline to ensure code quality.

    a. Required by the teaching institute
9. Swagger – Software for documenting API interfaces.
    a. Ease of use
10. GitLab – Git management system.
    a. Required by the teaching institute
    b. Familiarity

Project overview

Before reading the overview, make sure you have read the project plan to have an idea of what the project is.

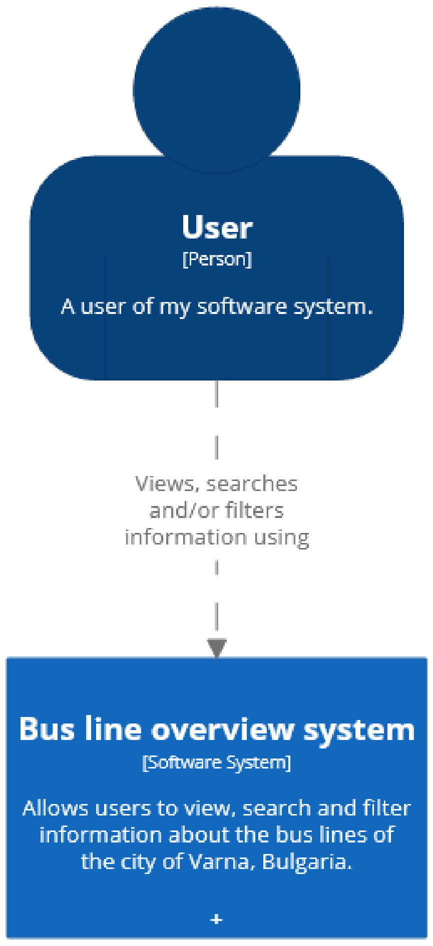
Let's start with the context of the system.

Users that want to view information about the city's traffic have to access the public web application. The user can either be an administrator, a user, or a guest.

Guests are users that use the website without an account. A guest can become a registered user by creating an account. Registered users can save the option to display a route of their choice on subsequent visits.

Administrators can be created only by other administrator accounts, which assumes that there is already a "hard-coded" account in the system. Their role allows CRUD operations of buses, tickets, stations, buslines, and alerts.

Proposed updates on the system will be: including an external system used by actual buses, which software would act as a GPS-locator and push updates to the main system.

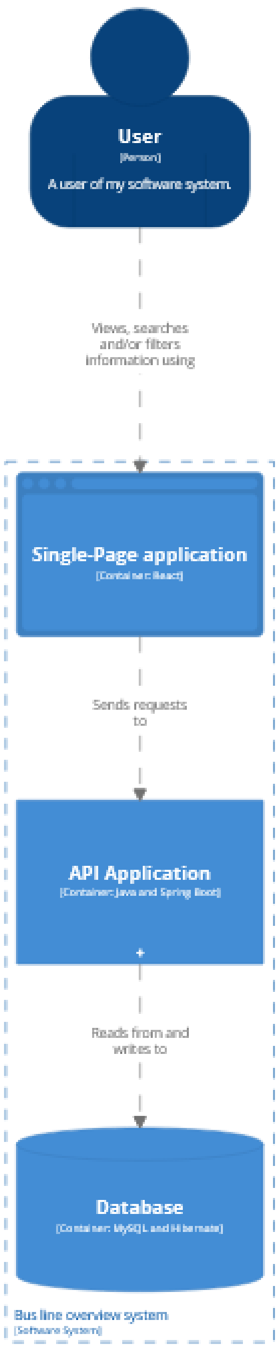

User
[Person]
A user of my software system.
Views, searches and/or filters information using
Bus line overview system
[Software System]
Allows users to view, search and filter information about the bus lines of the city of Varna, Bulgaria.
+

Moving on to the container context (C2), we can see under the system there are several applications, each responsible for a specific task.

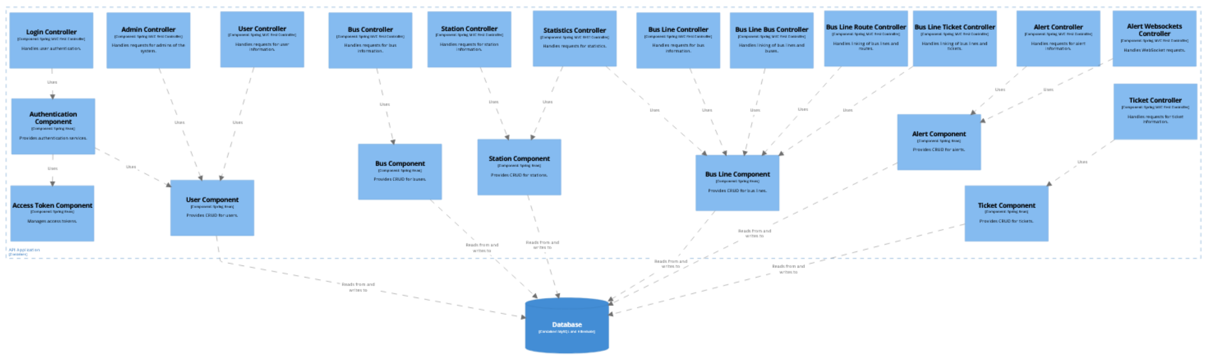
The interface that is presentend to the end user is the Single-Page application. The main library is React. React-leaflet is used to display the map from OpenStreetMap. Regarding UI/UX, Material-UI is utilized. Every user interaction uses Axios to fetch data from the backend. To ensure security, JSON web tokens (JWTs) are used. WebSockets are handled by the SockJS client, which connects to the backend endpoint on entering the website (or on refresh).

The API application on the diagram is the backend. It acts as a delegate between the frontend and the database, with one key difference – it is responsible for validating and providing as minimal as possible data (information hiding). REST controllers are used to access the data from the frontend, provided from Spring boot.

The database is responsible for data persistence and accessibility. The backend uses Hibernate for connections.



For a more in-depth look into the backend application, let's look at the component view (C3).


Login Controller
[Component: Spring MVC Rest Controller]
Handles user authentication.
Admin Controller
[Component: Spring MVC Rest Controller]
Handles requests for admins of the system.
User Controller
[Component: Spring MVC Rest Controller]
Handles requests for user information.
Bus Controller
[Component: Spring MVC Rest Controller]
Handles requests for bus information.
Station Controller
[Component: Spring MVC Rest Controller]
Handles requests for station information.
Statistics Controller
[Component: Spring MVC REST Controller]
Handles requests for statistics.
Bus Line Controller
[Component: Spring MVC Rest Controller]
Handles requests for bus information.
Bus Line Bus Controller
[Component: Spring MVC Rest Controller]
Handles linking of bus lines and buses.
Bus Line Route Controller
[Component: Spring MVC Rest Controller]
Handles linking of bus lines and routes.
Bus Line Ticket Controller
[Component: Spring MVC Rest Controller]
Handles linking of bus lines and tickets.
Alert Controller
[Component: Spring MVC Rest Controller]
Handles requests for alert information.
Alert Websockets Controller
[Component: Spring MVC Rest Controller]
Handles WebSocket requests.
Ticket Controller
[Component: Spring MVC Rest Controller]
Handles requests for ticket information.
Authentication Component
[Component: Spring Bean]
Provides authentication services.
Access Token Component
[Component: Spring Bean]
Manages access tokens.
User Component
[Component: Spring Bean]
Provides CRUD for users.
Bus Component
[Component: Spring Bean]
Provides CRUD for buses.
Station Component
[Component: Spring Bean]
Provides CRUD for stations.
Bus Line Component
[Component: Spring Bean]
Provides CRUD for bus lines.
Alert Component
[Component: Spring Bean]
Provides CRUD for alerts.
Ticket Component
[Component: Spring Bean]
Provides CRUD for tickets.
Uses
Reads from and writes to
API Application
[Container]
Database
[Container: MySQL and Hibernate]

The controllers can be seen on the top part of the diagram. Every controller corresponds to a service component, sometimes depending on multiple (LoginController), or a service being used by multiple (Bus line component). These services communicate with the database through the repository classes, which map objects into database-specific representation.

Use of the SOLID principles

Introduction

The SOLID principles are a set of five design principles for object-oriented programming. The acronym SOLID stands for:

- Single Responsibility Principle: A class should have only one reason to change.
- Open/Closed Principle: A class should be open for extension but closed for modification.
- Liskov Substitution Principle: Objects in a program should be replaceable with instances of their subtypes without altering the correctness of the program.
- Interface Segregation Principle: Many client-specific interfaces are better than one general-purpose interface.
- Dependency Inversion Principle: Depend on abstractions, not on concretions.

These principles are intended to make it easier to develop software that is easy to maintain and extend over time. By adhering to these principles, developers can create software that is more resilient to changes in requirements, and that is easier to understand and work with.

How are they applied?

These principles are primarily applied to the backend part of the application. The frontend part does not follow the OOP pattern, therefore shifting focus on the backend.

Single responsibility principle

The backend is segregated to three layers of concern:

- Controller layer – responsible for handling HTTP requests

- Logic layer – responsible for validating requests and operating business logic
- Database access layer – concerned with data persistence and object relational mapping

The project structure closely resembles the mentioned layering:

- src/main/java/com/varnabuslines/controllers – Controllers, separated by folders and their DTOs
- src/main/java/com/varnabuslines/services – Service classes, responsible for handling logic operations
- src/main/java/com/varnabuslines/domain – Domain objects, used by the services
- src/main/java/com/varnabuslines/configuration – Spring specific configuration classes
- src/main/java/com/varnabuslines/persistence – Repository classes/interfaces

Tests also follow the same folder structure.

Open/Closed principle

The structuring allows the project to be extended further. Due to the relative simplicity of the project, inheritance/generics/advanced design patterns have not been used, as it was not found applicable anywhere without causing unnecessary complexity.

Following the Closed for modification principle, domain models' API's have been kept private, unless needed.

Liskov substitution principle

Polymorphism is not used in this project currently, making this principle unapplicable.

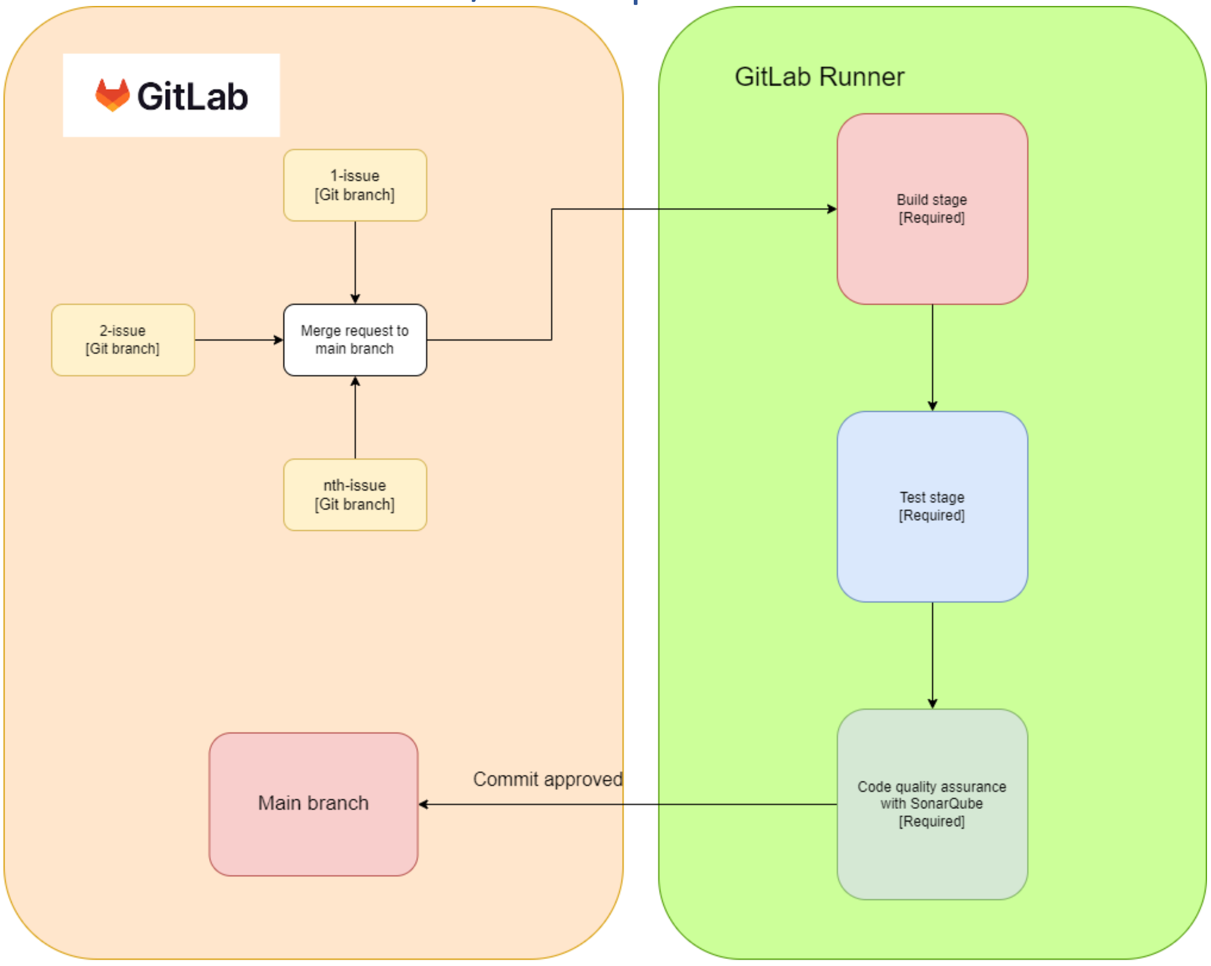
Interface segregation

Interfaces have not been used extensively in the project, due to the ability of Spring to automate the extraction of interfaces when needed. Principle is not applied at the current state of the project.

Dependency inversion

One of the main features of Spring is the ease of dependency inversion. Explicit code for dependency inversion is not used, except the provided annotations from Spring.

CI/CD Pipeline



The repository in GitLab has a main branch and several sub-branches, which eventually merge to the main one. When a merge request is created, it is required that the sub-branch passes the pipeline.

The committed changes first need to pass the Build stage – successfully build the whole backend using Gradle.

Then comes the Test stage – it run all the tests in the backend in src/test/java.

The third stage is the Code quality assurance using SonarQube. The build is analyzed and metrics are extracted which are used to determined whether the build passes the quality gate of the Sonar server.

Lastly, if any of the required stages fail the check, the merge request is rejected.

Testing

Frontend

The current state of the project includes two E2E tests using Cypress. The tests are designated for the Login page and Registration page. You can find the tests under the *frontend/cypress/e2e* folder.

Backend

Backend testing includes integration tests and unit tests under *src/test/java.*

The integration tests are focused on the controller layer. Every controller test class is annotated with the custom meta-annotation *@ControllerTest* which you can find in the same directory. It enables Spring context with the *@SpringBootTest*, mock MVC with *@AutoConfigureMockMvc*, and specifies the test profile, which simply uses the *application-test.properties* configuration file. In this file, H2 in-memory database is set up. Although the database is nowhere used in the test classes, it is still required due to two reasons:

- Limitation of Hibernate to always require database connection when using *@SpringBootTest*
- Guaranteed database connection, independent whether the production one is down

Unit tests are focused on the logic layer. All classes under `*domain*` and `*business*` sub-directories are annotated with the *@BusinessLayerTest* meta-annotation, which simply includes the *application-test.properties* file.

Test coverage is measured using Jacoco test coverage tool, which is supplied to SonarQube during CI/CD process.

Docker

There are two *docker-compose* files – one for debugging (*docker-compose.yml*) and one for production (*docker-compose.prod.yml*).

The former configuration initializes the frontend and backend, with the backend set up to use a dockerized MySQL image as the database, with phpMyAdmin for developer convenience.

The latter configuration initializes the frontend and backend, the difference being that the frontend is ran with the production image.

Known vulnerabilities

WebSockets

User story #11 requires WebSockets to be used for sending notifications to all clients that are subscribed to the server socket. The problem is, anyone can send alert notifications to the server, which can lead to spam issues. The feature is intended for admins to push notifications automatically when they create an alert, and every other client with other roles to receive the message.

The issue comes from the inability for SockJS to include authorization headers. [1] [2] A possible workaround is to include the authorization token in the URL, although this brings up a new security issues – exposed tokens. [3]

The current state of the feature is not production ready, but will be included as a requirement by the teaching institution's assignment.

Links

Git repository - https://git.fhict.nl/I478158/s3-indiv

Project plan - https://git.fhict.nl/I478158/s3-indiv/-/blob/main/doc/Project%20plan.pdf

Design document - https://git.fhict.nl/I478158/s3-indiv/-/blob/main/doc/Design%20document.pdf

UML diagram - https://git.fhict.nl/I478158/s3-indiv/-/blob/main/doc/UML.pdf

C4 diagram - https://structurizr.com/share/77232/diagrams

Pipeline diagram - https://git.fhict.nl/I478158/s3-indiv/-/blob/main/doc/Pipeline%20diagram.png

Research paper on SQL vs NoSQL regarding the project - https://git.fhict.nl/I478158/s3-indiv/-/blob/main/doc/Applied%20research.pdf

Test plan - https://git.fhict.nl/I478158/s3-indiv/-/blob/main/doc/Test%20plan.pdf

UX Feedback & Report - https://git.fhict.nl/I478158/s3-indiv/-/blob/main/doc/UX%20Feedback%20&%20Report.pdf

Research paper for justifying technology stack used - https://git.fhict.nl/I478158/s3-indiv/-/blob/main/doc/Research%20paper%20for%20justifying%20technology%20stack%20used.pdf

OWASP Security report - https://git.fhict.nl/I478158/s3-indiv/-/blob/main/doc/OWASP%20Security%20report.pdf

API documentation with Swagger – Start the app and open the link localhost:8080/swagger-ui/

References

[1] Raman, "Stack Overflow," 10 September 2016. [Online]. Available: https://stackoverflow.com/questions/32393519/providing-auth-header-with-sockjs.

[2] SockJS developers, [Online]. Available: https://github.com/sockjs/sockjs-node#authorisation.