Software Architecture Report

Date: 30.01.2024

Version (Commit ID): 0e38c6094482f2899574b185ca415c0e5201b11e

Contents:

1. What the purpose of the software project is

    A. Summary of the planning documentation:

    We implemented a web application (made with Java + SpringBoot on the backend with data stored in a PostgreSQL database, containerization done with Docker and the frontend built in React.ts) that fulfills the general need of an easier and more accessible way of managing the school enrollment process.

    B. Fulfilled capabilities:

    After they have decided to use our application, guests can register as users with the default parent role. Parents can add and manage their children and apply for them to various schools. Every school is registered and managed by a principal whose role is granted by an admin after a prior offline discussion. A principal can add, delete and edit classes and add/delete teachers to his school. Every class must have a class master chosen from the teachers of the school.

    A parent can send applications (requests) to schools, where they can be either accepted or rejected by its principal. If the parent wants to cancel any request, they can do so, but only before it is accepted/rejected by the principal. After being accepted, the request is then confirmed/declined by the parent. If the request is confirmed, the child is automatically enrolled in that school and all of the other requests are deleted. From it is the principal's responsibility to assign that child into a class according to what the parent requested.

    A user is granted the role of teacher whenever they are added to a school by a principal and they have access to a separate page where they can see info about the class they are responsible for (class name and students).

    The admin's only purpose is to grant principal roles and keep watch over the users.

    The role of the user can be dynamically changed from within the user profile page. From this page they can also edit their personal profile data and permanently delete their account.

2. Guides on how to:
    a. How to run the project locally:

        In order to run the project locally, you have to install docker, open up a terminal and run the following command:

        docker-compose up -d

        This will set up containers for everything.

        After running the docker command above, just hop into any browser available and type this in the address bar:

        http://localhost/

    b. Build the project.

        The build task is automatically completed by docker when running the specified command.

    c. Deploy the project (either locally, or how is it hosted, where it is the case)

        For the moment (not for a long time), the web app can be accessed at this address:

        http://catag.go.ro

        The local deployment steps are described in section a. of this point.

    d. Contribution guide

        We adopted a layered software architecture in our application, having separated the Presentation, Business Logic, Persistence and DataBase Access Layer.

        For backend we used the Controller-Service-Repository pattern because it's well suited for web applications and facilitates testing and code modularity. For each entity in the persistence, we have a corresponding table in the database, a corresponding repository and a service that manages the business logic and that are used in the controllers where the endpoints are defined.

        Contributors must respect the basic Java coding style.

        As for the frontend side of the application, we used React with TypeScript and Redux for better syntax checking and state management.

        Contributors must respect the basic TypeScript Style Guide.

3. Application entry points
    a. Data sources

        The only data source that we used in building and running the application is our own made PostgreSQL database that is hosted on our own server. This database stores all the information needed for the application to properly function, starting with children's information and ending with school requests.

b. Data inputs

Data input is handled by the Spring MVC Controllers that manage http requests according to the application's business logic. Controllers are responsible for initiating the update of the model(the data displayed to the user in the frontend application) by calling services which use repositories to effectively mutate the state of the model and store the new state into the database.

c. Configuration files

In the backend application we use two configuration files, both called application.properties.

One of them is responsible for configuring the properties for the running case of the application, where we set the database connection details, such as the jdbc url, the username and the password (this one is stored in a system variable) as well as selecting the proper dialect of SQL that is used in the database. The second config file is the one being used under testing scenarios, in which it is specified to use an in-memory database for the tests, so the modifications performed by the tests do not affect the main database used when the application is executing.

4. High level diagrams of the architecture
   a. User/data journeys

      User journey map.

   b. Most valuable output

Parents are in search of a handy way of managing the school enrollment process of their children and principals are overwhelmed by the high amount of paperwork they have to go through when dealing with this same process. Our application's goal is to make life easier for both sides by using the technology that people use in their day to day life. Thus it provides an easy way of managing both the children and the school requests for the parents as well as school, classes, teachers and requests for the principals using some of the most popular technologies in the industry.

With this web application we aim to take away the burden that comes with this time-consuming process and with all the functionalities that it presents, we believe that we successfully achieved that.

5. Deployment plan
   a. Where is the application deployed.

      The application is deployed on a self-hosted server and can be accessed at this link (mentioned earlier as well):

      http://catag.go.ro

   b. How the CI/CD pipeline works.

We used github actions to build and test every push made to our repository. Every push to the main branch triggers a workflow that starts an ubuntu container that pulls our repository from github, sets up JDK 21 and uses gradle-build-action to build the project and run the unit tests. The secrets needed for database connection and JWT are stored in repository secrets and passed to the workflow via environment variables.

6. Description of the QA process

a. Test suites – what do they test.

We decided that the backend service layer is most suitable for unit testing because the controllers are complex operations that involve several service functions and the repository layer is already implemented by the framework. In total we have 41 unit tests that check and validate services functionality in various scenarios.

Additionally, we have a few end-to-end tests written with cypress that test the main functionalities of the application, such as adding a child, sending a request to a school for a child, confirming the enrollment in a school, accepting a request by a principal and adding a child in a class by a principal.

7. External dependencies included in the project

a. APIs used

We did not use any external APIs in our projects. The only used ones came out of the box with the frameworks that we decided to code in (eg. React comes with fetch out of the box and axios after installing axios and in Java we used SpringBoot which comes with Spring Data JPA, Hibernate, DevTools and many more)

b. Libraries

We used plenty of libraries (mostly on the frontend part). Some of the most important are the following:

- React
- Axios
- Material UI
- Bootstrap
- Redux
- Cypress (testing)
- Spring Web
- Spring Data JPA
- PostgreSQL driver

How vulnerable is the project to dependency attacks (ex. Dependency Confusion)

Overall, there are no high vulnerabilities regarding dependency attacks with the libraries that we use in our project. However, the React library (node actually) can throw some dependency warnings, errors, or even vulnerabilities, but those can be fixed running npm audit or by installing the latest versions of the libraries used.

8. Additional questions:

a. What are the technologies that you have used, and why?

- For the Front-End part we used react because it is a very popular library and if we were to make mistakes, we could easily find the error on the internet and try to fix it. We used redux as a state manager for the same reasons;
- For the Back-end part we used Java because it is a reliable, secure and easy to use programming language that has a lot of libraries and APIs to help develop a back end application; alongside Java we use SpringBoot because it is one of the most popular framework for Java that facilitates all the work needed to build a web application from scratch.

b. What are the architectural patterns you have implemented and were they appropriate, and why?

The architectural patterns that we used were presented earlier (point 2. d.) and we mainly used them because they are considered good practice, they are standard in layered applications like ours and the framework used offered simple ways of implementing them.

c. Were the coding principles established enforced successfully? (For example, if you included linting in the application, did you respect it? Did you try to implement coding standards and practices?

Some of us used linters for the React code and we tried to respect it as much as possible, but sometimes the errors were so weird we just gave up and left it how it is.

As far as the Java code is concerned, we did not use any specific tool for establishing code principles, but we tried to follow basic java coding style and good practices.

d. What are the faults that were discovered during development that haven't been addressed by the time of delivery?

There are no faults that we discovered during the development process that we haven't been able to correct prior to the delivery. Every bug that was discovered was addressed and fixed before the final presentation of the project.

e. Does any part of the project require refactoring?

- On the Front-End part, there are components that can be written in a more readable and understandable way and because we do not have experience working at a pretty-complex project, we sometimes had different view perspectives about the app and this led to some minor inconsistencies within the code that can be fixed in future versions.
- On the Back-End part some refactoring could be done both in the controllers and the services, because sometimes we used services poorly and we did a little too much business logic directly in the controller.