INFO-AFP Project Report

G.A. Blanken (6438830), H. Fidder (6513395), M.N. Tjon (3240010)

April 2023

1 Introduction

Our project aims to develop a classroom-based submission system designed to improve the assignment management process for students, teachers, and teaching assistants (TAs) in an educational setting.

The systems allows teachers to set up classrooms, add students to these classrooms, and add assignments for the students to work on. The system then enables students to submit their responses to the assignments, providing a user-friendly interface for uploading their work. This simplifies the submission process, eliminates the need for physical copies, and ensures timely submission of assignments. The students are allowed to hand in multiple attempts, so that they may improve on their work over time.

The platform allows teachers and TAs to access submitted attempts for grading. With a streamlined interface, educators can efficiently review each submission, assign grades, and provide written feedback. This way, teachers have a singular entry point to gather and review work, which reduces administrative overhead.

A vital component of the classroom-based submission system is the ability to deliver feedback on assignments. While grading, teachers and TAs can share their feedback, including comments and suggestions, with students through the system. Students can then learn from this feedback to increase their understanding and aid the learning process.

In summary, the classroom-based submission system aims to optimize assignment submission, grading, and feedback processes for students, teachers, and TAs, ultimately enhancing the overall educational experience.

2 Important concepts and techniques

In this section, we will discuss the important concepts and techniques used in the implementation of our submission application written in Haskell and Elm.

2.1 Backend

The backend was written in Haskell, and mainly written using two libraries:

- Servant - A web application framework
- Beam - An Object-relational mapping libary

2.2 Servant

Servant is a type-level web framework for Haskell that allows us to define APIs in a concise and type-safe manner. Using Servant, we can specify the routes, requests and responses, all as a type. We can then define a function typed using this specification to define the implementation of the API. This enables us to define a clear spec for our web service and ensure consistency between the implementation and the spec.

In addition to defining the API for our application, Servant can also be used to serve static files needed for the frontend. This allows us to keep our frontend assets in the same repository as the backend and serve them from the same web server.

2.3 Beam

Beam is a type-safe, high-level, and extensible Haskell library for working with databases. It allows us to define our data models as "Beamable" types. The declared types should also be Generic, to allow the data to be written to multiple types of databases. These types represent the tables in the database, and by using Beam, we can interact with the database using type-safe Haskell functions. This approach reduces the chances of errors due to mismatched types or incorrect queries, making our application more robust.

2.4 Beam-migrate

In our project, we chose to use Beam-migrate for its powerful capabilities in managing database schemas and migrations. Beam-migrate allows us to automatically generate a schema from our data models, ensuring that our database structure accurately reflects the underlying Haskell types. This approach reduces the likelihood of inconsistencies and errors arising from manual schema creation. Additionally, Beam-migrate handles schema changes by generating migration steps when modifications to the data models occur. This feature streamlines the process of adapting our application to evolving requirements and ensures that our database remains consistent and up-to-date.

2.5 Frontend - Elm

For the frontend, we have chosen Elm, a purely functional, statically-typed language that compiles to JavaScript. Elm is designed to make frontend development more reliable and maintainable by providing a strong type system, helpful error messages, and a simple, yet rather rigid, architecture based on the Elm Architecture (TEA). The use of Elm ensures that our frontend is robust, easy to understand, and maintainable over time, complementing our type-safe Haskell backend. We used "elm make" to build the elm application to a single html file, containing the entire frontend.

3 Results

3.1 Backend

On the backend we implemented the types needed to facilitate our application, and leveraged Beam-migrate to generate a schema for this database.

We split the backend application into three layers:

- The database layer

- The application layer
- The api layer.

The database layer contains functions that simply interact with the database. The layer takes care of getting, filtering, writing, and deleting data. This is the layer that makes use of Beam the most.

The application layer takes input from the API layer, and uses the database layer to do validation, and eventually execute the required function. This layer can contain domain specific logic.

The API layer defines the API spec and how this spec is implemented using the application layer, this is done making use of Servant.

3.2 Frontend

The frontend of our application was written fully in Elm, the features provided by the frontend will be described in more detail below.

3.3 Login Page

A 'Login' page allows users to access the system by selecting their user account. This ensures that each user can access only the information and features relevant to their role. Given the scope of our project this does not involve any form of authentication. Rather, the users known in the database are simply provided in a drop-down list which you can select or delete. Users can also be registered simply by inputting a username and the desired role.

The system supports the following user types: students, teachers, and Teaching Assistants (TAs), each with distinct access levels and functionalities.

3.3.1 Students

Can view the classrooms they are participating in and provides the capability to create submissions, each having the potential for multiple attempts, for the associated assignments. Students can also view the grade and feedback received from a teacher or TA.

3.3.2 Teachers

Can create classrooms, add TAs and students to them as 'ClassroomParticipants', create assignments, and grade submissions.

3.3.3 TAs

When participating in a classroom, a TA can grade submissions of the participating students. A TA cannot, however, add participants, or create classrooms.

3.4 Classrooms

Shows the participants of a course, upon login a user is presented an overview of classrooms they participate in. As a teacher, the user can add new classrooms. By selecting a classroom, the user can view the participants and their respective roles, as well as a list of assignments associated with the classroom. Again, as a teacher, the user can add new assignments, as well as add new participants (TAs or students) to a classroom.

3.5 Assignments, submissions, and attempts

From the classroom view, a user can view an individual assignment, including a description. Teachers and TAs can view which students have entered a submission and continue on to grading them. If a submission has already been graded, a grade can be removed from here.

Students can add attempts, which are aggregated into a submission. This means an assignment has a single submission per student associated with it, but each submission can have an arbitrary number of attempts.

3.6 Grading

Teachers and TAs can grade submissions, entering a score and leaving feedback. This will then be available for students to view in their assignment overview.

3.7 Adding and Deleting Data

The frontend enables users to add various elements, including:

- Users: Creation of new student, teacher, or TA accounts.
- Classrooms: Establishment of virtual classrooms for organizing courses.
- Classroom Participants: Addition of students, teachers, and TAs to the virtual classrooms.
- Assignments: Creation of assignments for students to complete and submit.
- Submissions: Students can submit their work for grading.
- Attempts: Keeping track of multiple attempts made by students for each assignment.
- Gradings: Teachers and TAs can assign grades and provide feedback on submissions.

3.8 Removing Data

The system also allows for the removal of specific elements, such as:

- Users: Deletion of user accounts, if necessary.
- Attempts: Removal of assignment attempts, if required.
- Gradings: Deletion of grading information, in case of errors or modifications.

For some impressions of the frontend, please see the attached images in the apppendix.

4 Reflection

In this section, we will highlight some of the struggles we had to overcome during the project.

4.1 OpenAPI

We explored the use of an OpenAPI specification for setting up our API server and clients. The promise is that we would not have to hand-write any calls ourselves and instead be able to generate both a server and a client from a single specification. However, we had already written and implemented a significant part of our database schema and interaction (see beam and beam-migrate). While there are generators for database schemas from OpenAPI specifications, it was too late to integrate with our existing work. Additionally, it was unclear whether we would have been able to make it work with beam at all. We therefore chose to forego the use of an OpenAPI specification. If, in the future, we would want to use an OpenAPI spec, we need to make sure it is the first thing we start with.

4.2 Beam-migrate

Beam-migrate has certain limitations that can pose challenges during implementation. Notably, the library suffers from missing documentation [1] which can make learning and using the tool effectively more difficult. Additionally, some online tutorials contain misleading or incorrect information [2], further complicating the learning process. Beam-migrate occasionally struggles with simple schema changes, such as renaming a column, resulting in failed automatic migrations that require manual intervention. Another significant limitation is the lack of support for Foreign Key constraints, as discussed in issue #502 on the Beam GitHub repository [3]. This absence means that our deletes do not cascade properly, and we should handle the cascading behavior manually to maintain referential integrity. We decided that manual cascading would be out of scope as it is a clear bug in the library chosen. These limitations presented challenges that needed to be carefully considered and addressed when using beam-migrate in our project.

4.3 Elm

While very educating and interesting to learn, using Elm proved to be a much more time consuming task than initially estimated. This is mostly due to the completely different style of programming that needs to be incorporated compared to using more standard languages likes JavaScript and related frameworks. As Elm also still has a rather limited ecosystem, it was hard finding easy to implement libraries and tutorials, and we stuck to mostly using the standard libraries.

Due to time constraints we also had to cut some corners, such as not handling error results in any way, leaving page styling to a minimum, and avoiding using sessions to keep users logged in.

It was also non-trivial to find a ui library that could do basic component creation for us, as most frameworks still required JavaScript and bundling which we decided against, favouring a purely elm based application.

4.4 Serving static files

While serving static files with Servant is possible, the support is limited. Due to this, we were unable to find a way to route a route such as a "/frontend" path to a single file, which means that reloading pages on the frontend causes a 404, as servant will try to load files that simply don't exist if Elm is not handling page navigation itself.

4.5 More future work

Due to time constraints, we were also not able to implement the following features:

- Defining quickcheck properties to test submissions automatically.
- Doing complete data validation, such as checking for duplicate usernames, or the values that grades may take. Submissions are also not checked for being handed in within the predetermined timeframe.
- Implementing proper error handling on the back- and frontend, currently we return Nothing on the backend if an error occured, and don't provide feedback on the frontend for this.

References

[1] https://haskell-beam.github.io/beam/schema-guide/library/

[2] https://www.williamyaoh.com/posts/2019-09-27-figuring-out-beam-migrations.html

[3] https://github.com/haskell-beam/beam/issues/502

5 Appendix

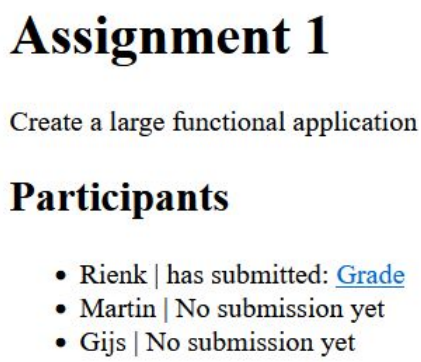

Figure 1: Example of an ungraded assignment as viewed by a teacher

Assignment 1

Create a large functional application

My submission:

Grade:

No grade yet

Attempts:

- 2023-04-21T17:59:41 | attempt id: 1

New attempt

Figure 2: Example of an ungraded assignment as viewed by a student

Assignment 2

Write an increment function in Haskell

Participants

- Rienk | No submission yet
- Martin | has submitted:
  already graded Delete grade
- Gijs | No submission yet

Figure 3: Example of a graded assignment as viewed by a teacher