DQGUI: Final Report (Semester 1)

DQGUI is a java based IDE (Integrated Developer Environment) for the IQM4HD (Intelligent Quality Monitoring for Heterogeneous Data) domain specific language. It provides users of the IQM4HD project a way to interact with the engine by utilizing a graphical user interface provided by a fat client. This allows users to develop components of the IQM4HD language without the need of setting up the entire IQM4HD infrastructure first. DQGUI features a tab based editor with syntax highlighting, management components that: create, edit and remove domain specific code, execute domain specific code, visualize and filter the results of such executed code, supervise an R process that is required by some features of the language and map database connections to be used by the IQM4HD engine. It contains a modular system for database and file operations that allows future developers to extend the application by their own functionality. The database abstraction layer allows developers to implement support for previously unsupported database engines while the domain specific language service abstraction creates a foundation to support different ways of accessing IQM4HD source code. It also features a framework that wraps the underlying JavaFX that is used for the graphical user interface.

Introduction

The IQM4HD project provides a java based evaluation engine for the projects' domain specific language. DQGUI implements an IQM4HD client, the client in this case is the graphical user interface which is also written in Java as that allows direct access to the IQM4HD application programming interface. IQM4HDs domain specific language is built on modularity that allows reuse of components. The current three main components are named action, source and check.

A source provides data for the check to work on. This data can be sourced from either a database or other data providers as well. The check now works on this data and analyzes it by user defined operations. Such operations could be a not null check that checks if the data contains null values, pattern checks that matches for patterns and various other checks. The action is comparable to the main method in a Java application. Sources and checks can not be executed directly as they are modular components (a source being a provider and a check a consumer), thus there is an action required to glue them together and provide a meaningful context.

As an Integrated Developer Environment (IDE) DQGUI lays a focus on being extensible for real world usage. The mentioned database abstraction layer currently supports PostgreSQL and MongoDB only, but allows developers to add support for other engines as well. These engines can then be used by IQM4HD and will also provide a graphical configuration wizard to set connection parameters.

Database connections are held in environments identified by an identifier. An environment is a set of database connections.

The contained connections also hold an identifier that has to be used within the domain specific code. The evaluation engine then knows which database connection to request from which environment.

This distinction allows for example to separate between production and test databases without changing the connection credentials and domain specific code connection identifiers manually.

The software offloads its features onto several windows at the moment. It features wizards that help the user setting up a connection or create a new source component. The three most important windows which are the main, report overview and R management window can be opened and worked on concurrently. Execution of domain specific code is done in a different thread to not block the UI and allow several actions running at the same time. It is also possible to display the IQM4HD log of a currently running worker thread.

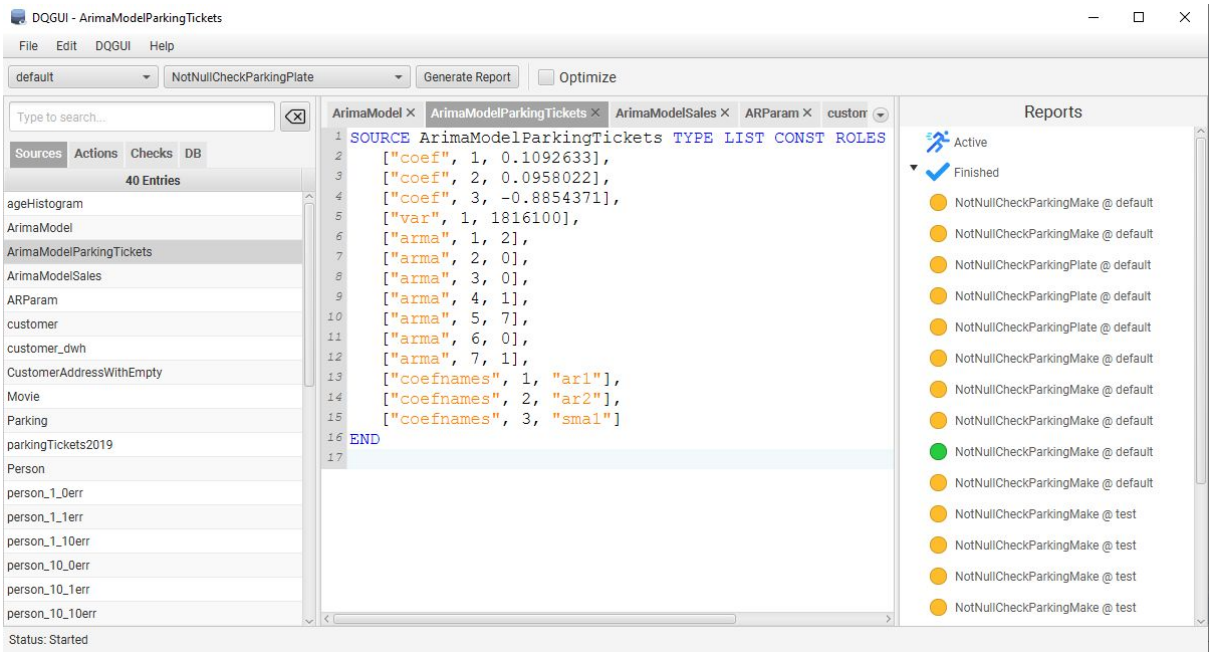

Figure 1: DQGUI's main window

The following parts of this report will provide an in-depth analysis of the projects architecture and important components. It will attempt to explain the inner workings of the software in its current stage and possible advantages / disadvantages of certain implementations. It will then allow the team to reflect on itself and discuss missing features that are planned to be completed in the second semester. At the end this report shall draw a conclusion on how this project benefited our skills and if we were able to grow as a developer.

Architecture

We have gravitated towards a fat client architecture as this allows easy deployment towards the end user. The resulting jar file contains all dependencies including IQM4HD and can then be run directly as long as the correct Java runtime environment is provided.

There are plans to remove the IQM4HD dependency from the jar file and providing the user a way to register his own IQM4HD jar that will then be loaded by the class loader externally. This would allow us to decouple DQGUI from IQM4HD and all further changes towards the IQM4HD project would not force the maintainer to run the build process for DQGUI again just to include the updated library (unless the API changes and the DQGUI calls have to be updated).

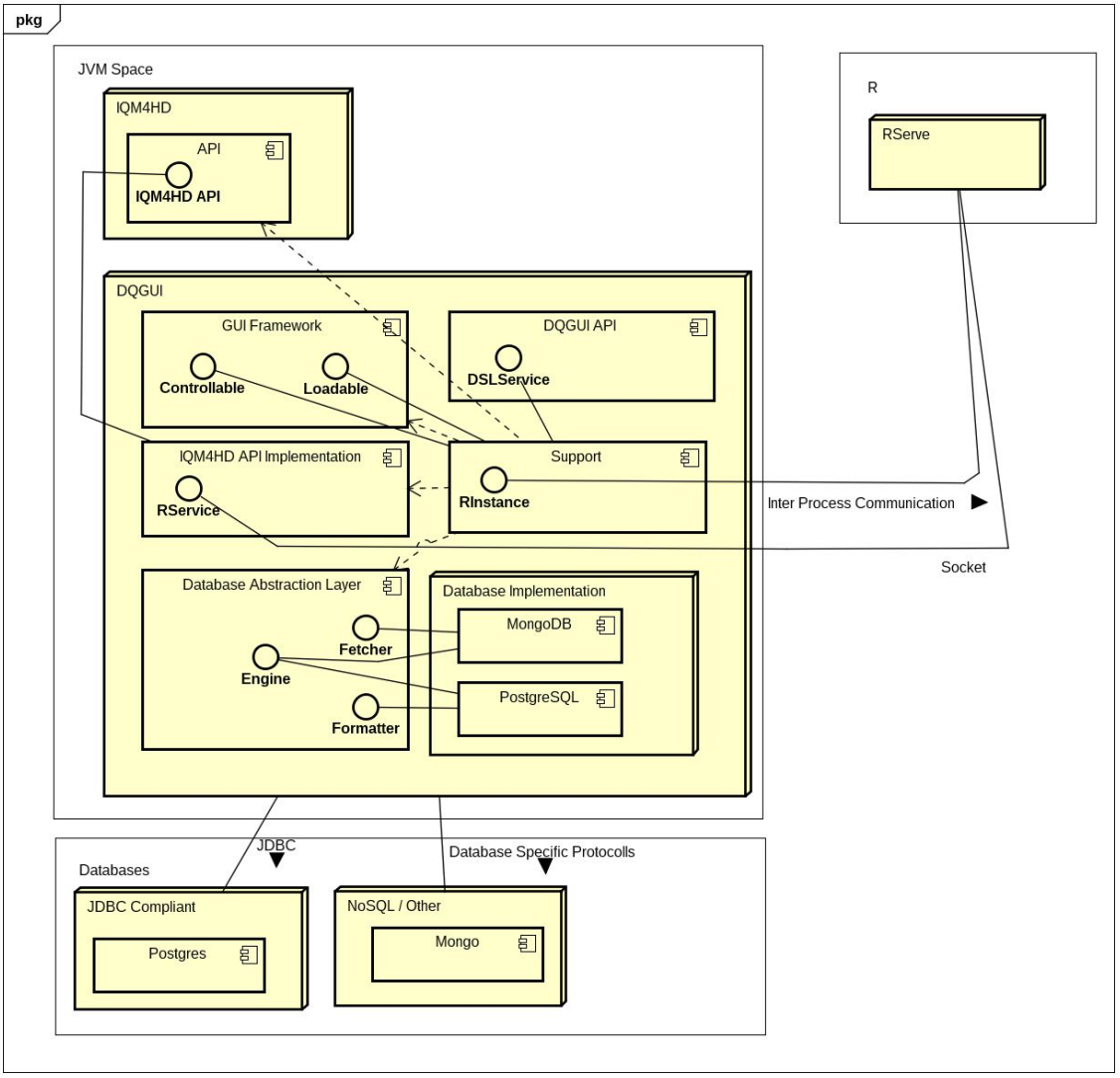

Figure 2: Overview of internal and external components and communication

GUI Framework

The internal GUI Framework wraps itself around JavaFX and provides features that JavaFX does not implement by default and aims to abstract as much repeating code away as possible to allow development without worrying too much about JavaFX specific recurring procedures.

Current features are:

- Primitive dependency injection configurable over a YAML file or by finding a matching constructor for a varargs object input
- Lifetime management of windows, limiting of max. parallel opened window instances

- Subcomponents which can be lazily loaded and attached to the scene graph by a given controller
- Communication over multiple windows using a signal sender/receiver/handler approach
- Automatic signal registration/unregistration configurable over a YAML file
- Dialog API for a set of pregiven dialogs, option to create and load custom dialogs
- Wizard API for retrieving and validating user input
- Generic Data Container for JavaFXs cell based controls (TreeView, TableView)

DQGUI's graphical user interface is basically an implementation / extension of this internal framework. Self development of such a framework is justified as the JavaFX eco system is rather small and a Spring equivalent for JavaFX does not exist yet. The other existing frameworks for JavaFX did not suit our needs, so tailoring a framework specifically for the application seems the best way to go without introducing too many rarely used external.

Database Abstraction Layer

The Database Abstraction Layer aims to allow programmers to support previously unsupported database engines for usage within IQM4HD. It also connects to the database environment / connection management system and allows the maintainer of the application to implement a solution that works with the database wizard so users can organize / test their database connections utilizing the GUI.

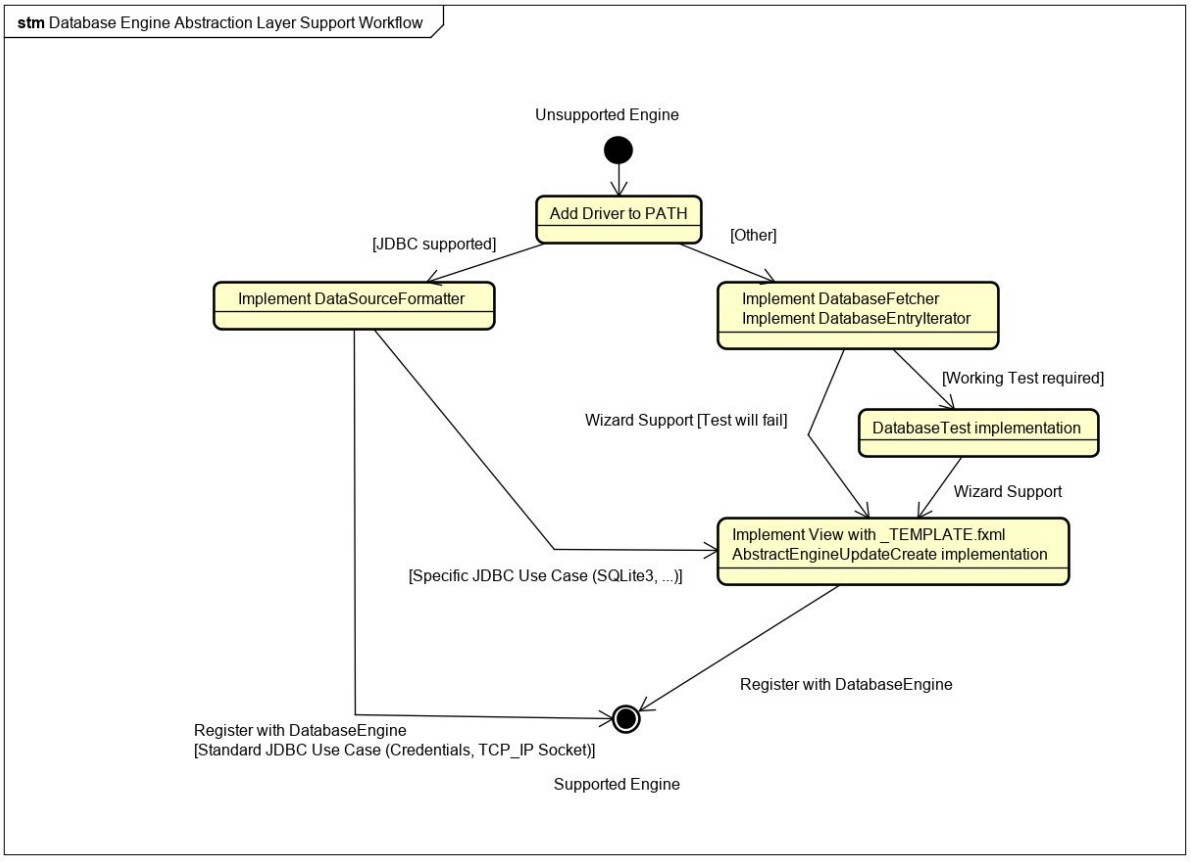

Figure 3: Necessary steps to support a previously unsupported database engine within DQGUI

Supporting a database engine is easier if a JDBC supported driver exists for the engine, the user is then only required to implement a DataSourceFormatter to convert the user specified settings to a JDBC data source URL that can be passed to the DriverManager. If no special configuration parameters are required a generic JDBC input window will be used so a programmer would not even have to support the wizard GUI component.

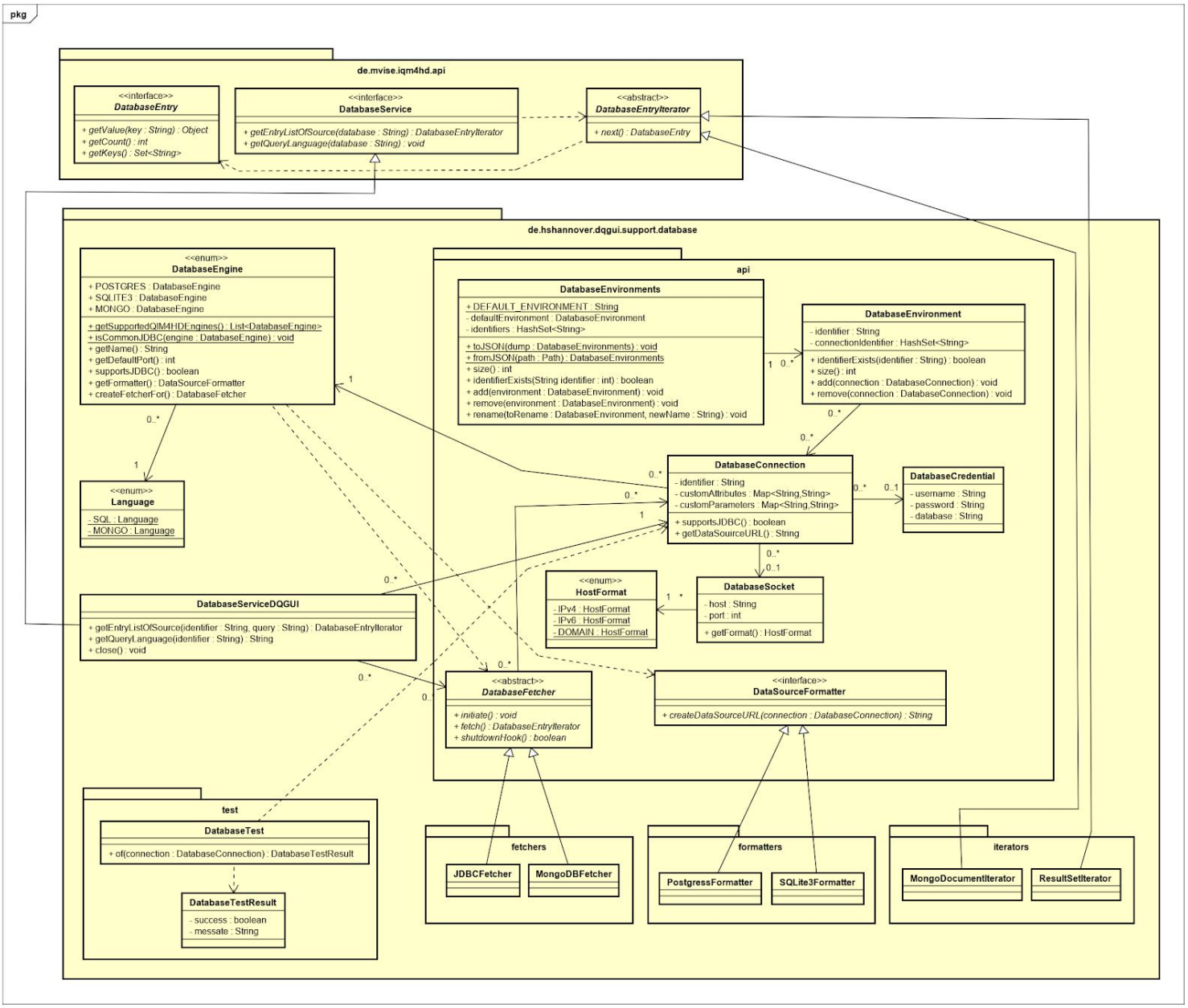

Figure 4: Overview of the Database Abstraction Layer

More in-depth documentation can be found within this wiki page or the javadocs of the database package as a full description would go well beyond the scope of this final report.

Support

The support package houses the database abstraction layer, concurrency, serialization and the DSLService interface that is used by DQGUI in order to execute CRUD operations on the currently selected repository.

Implementing the DSLService interface allows DQGUI to access repositories via different protocols. Currently there is no way to change implementation through the DQGUI properties window as only a file system based service is implemented and configurable. If another protocol such as a database should be supported a GUI support within the properties can be implemented to allow switching through different service implementations. A service can also validate itself and the state of the repository to notify the user about malformed or otherwise misconfigured repositories (i.e. a connection failing, a selected repository not existing on the file system). Since the service is tightly coupled with IQM4HD's RuleService the DSLService also acts as a factory for RuleServices used within the IQM4HD evaluation process.

Specifications regarding the DSLService can also be found within this wiki page to not go beyond scope of this final report.

Projection for Second Semester

A large chunk of functionality is already implemented at this stage. Testing will now play an important role — minor bugs that have never been caught yet need to be ironed out as there are no UI tests for our software. There might be issues with the interface layout as well that could confuse new users, as all developers are out of touch when it comes to assess the initial usability for new users due to having built the software and thus knowing the entire set of functionality. Data profiling still has to be designed and implemented and is the missing link to completion of the project. During the Texas excursion Mr. Kleiner has also created a list of issues regarding the software that should be handled. Syntax errors in the DSL are currently still hard to trace under certain conditions. Semantic checks are also missing which might cause confusion under certain circumstances. Improvement in both areas might be possible after implementing data profiling and clearing the list of issues that arose during the Texas excursion.

Teamwork Reflection

Grant Singleton - *It was great to finally meet Marc and get to work on the project with him. This past sprint was not my most involved due to finals and the Christmas break between semesters. I was, however, able to learn a bit about ANTLR in order to parse the grammar for syntax error highlighting. I have learned a lot about Java development and specifically JavaFX throughout this process. I have also learned a great deal about Agile and the software development process. Overall this has been a great experience.*

Marc Herschel - *It was fun working together with students of another university. Although the distance naturally makes it harder to communicate efficiently it was still manageable during the first semester of the capstone project. I learned a lot about software architecture and SCRUM during the sprints and enjoyed the freedom of working on something new instead of maintaining legacy code. This project allowed me to practice working remotely which is an important skill in our globalised world. I had no issues speaking / writing and understanding English and additionally welcome this cooperation as an advanced English training session.*

Julian Sender - *I have made very good experiences here in this team so far. After we had some minor difficulties in understanding the project task at the beginning, we all showed great commitment and everyone made their contribution. As a result, we were able to create a great product, got through much faster than expected and better than planned. The team has grown together and we all regretted it very much when Daniel left. For me, the key to success was a great and frequent communication which was kind of easy since we are all communicative personalities. I can also say that my English has improved a lot due to the frequent practice and I am sure that we will stay in touch beyond the project.*

Kevin Duan - *This project has been an extremely enlightening experience for me. We had a rough start, as we did not understand the project parameters initially and struggled with working around the large time gap. However, as we got to know each other and interacted more, we began to mesh incredibly well and I would go as far as to say that our team dynamic has been an extremely large part of our success as a group. We fully intend to stay in touch after the completion of our project and we have already begun discussing future trips together.*

In addition, the project helped me become more proficient working in an agile scrum environment as well as working remotely. With how global companies are now, working with time and language differences as well as other barriers are a huge part full time work and I am extremely happy and proud of our abilities to overcome our initial hurdles.

Conclusions

Major difficulties encountered

There were no interpersonal problems that are worth mentioning here. Issues of technical nature rarely arose. Our biggest difficulty was to understand the requirements correctly in order to implement them properly.

Lessons learned

Good communication is crucial while working within a team. Issues can be discussed openly and introducement of bugs / wrong decisions within the software design are dealt with in a cooperative and non-accusatory way. A welcoming and open team atmosphere results as a product of the previously mentioned actions.

Things we wish we would have done differently

Each team member is very satisfied with the course of the project, with the other team members, as well as with the care and responsiveness of the supervisors. For this reason, we do not see any significant points that should have been done differently.

Specific knowledge and tools that helped a lot

Git - Especially with different workers on the same project, it was great that each team member was able to checkout a branch and work on it separately since we were working within different areas of the software. Merging back into the master went smoothly with minor conflicts to resolve.

Communication - Due to our team being spread across two continents good communication is crucial. Slack, Teamspeak, Whatsapp and Trello helped immensely to keep the exchange between the team members going and eased coordination / mitigation of issues and bugs.

Others - JavaFX, Java, database related knowledge

Which modules from the CS curriculum would we like to have completed before the project / What we would have liked to know before

Software Engineering 2 completed would have helped with our understanding of SCRUM. We taught ourselves SCRUM via the internet, but it would have been far easier in the beginning with solid SCRUM knowledge.

Lastely, we should have paid more attention to reading the documents submitted to us thoroughly down to the last detail. Then our start would certainly have been even faster.