Group 6: Ade Aiho, Heta Hartzell, Mika Laakkonen, Jonne Roponen

Qupids Speed Dating

Final Report

Second-year Software Engineering Project

School of ICT

Metropolia University of Applied Sciences

8 May 2025

Contents

AI was used to format and style parts of the document based on each writer's original text.

1 Project Development Process

1.1 Applied Methodology

The project Qupids Speed Dating was developed with agile methodology during the courses Software Engineering Project 1 and Software Engineering Project 2 at the Metropolia University of Applied Sciences. The project was split into eight sprints in total that each lasted two weeks. Agile methodology focuses on delivering features and development in repetitive phases. It also emphasizes improvement and adaptability to change. At the beginning of each sprint user stories were chosen from the product backlog according to their importance and sprint requirements. These were then planned and implemented into tasks for the sprint backlog.

Agile is a methodology that focuses on iterative approaches to software development. The agile approach that Qupids Speed Dating was executed with was Scrum. Scrum consists of set roles for each team member and their duties differ. Scrum is about teamwork and delivering features in increments. It consists of ceremonies that help the team work together towards the same goal. In Scrum, the team is open for constant feedback, self organizing, and flexibility.

Project task management was handled with a kanban style board called Trello. User stories were maintained and updated in a Trello board called product backlog and each sprint also had their separate boards called sprint backlogs, which consisted of planned tasks for the current sprint.

As part of continuous integration Jenkins pipelines were used to automate building, deploying, testing and reporting. With automation human error and workload reduced greatly. After configuration, CI/CD proved to be an amazing methodology for software project development.

Static analysis was done with several different tools that analyse the code without execution. They scan the code for vulnerabilities, bad practices, syntax

errors, and structure. Each tool has a slightly different focus when it comes to analysis. The tools used were PMD, Checkstyle, SpotBugs, and SonarQube.

For version control GitHub was used. Every team member worked on different parts of the code in their own branches and features were eventually merged back into the main branch. GitHub also serves as the main hub for any type of documentation relating to the project, be it about development process or user interaction.

1.2 Planning and Risk analysis

Project Planning

The project plan for Qupids Speed Dating was set around eight agile sprints, each with different specific objectives that were determined in sprint requirements. In sprint planning sessions at the start of each sprint, tasks were assigned to team members and their deadlines were estimated. The priority of each task was also assessed during sprint planning. Trello was used to manage and maintain the product and sprint backlogs. The following table 1. showcases each planned sprint and their respective themes and primary objectives.

Sprint	Main Focus
Sprint 1	Project planning and setup (requirements analysis, tech stack selection, GitHub and Trello setup.)
Sprint 2	CRUD and integration (profile/session creation, session joining, interest management, and JavaFX DB integration.)
Sprint 3	Frontend backend integration (UI improvements, matchmaking results, admin tools, help section, and CI/CD setup.)
Sprint 4	Final steps and deployment (finalize testing, set up containers, prepare presentation.)
Sprint 5	Localization (UI and database localization, multilingual support, and resource identification.)
Sprint 6	Code quality and acceptance testing (static code analysis, code cleanup, acceptance test planning, documentation.)
Sprint 7	Alpha, performance, and reliability testing (UAT execution,

	bug fixing, documentation of test results, retrospective.)
Sprint 8	Final delivery (presentation creation, final documentation, final review, project submission.)

Table 1. Sprint overview and themes.

Risk Analysis

Risk assessment of each potential risk and their mitigation strategy can be seen in table 2.

Risk	Likelihood	Impact	Mitigation Strategy
Illness of a team member	Medium	Medium	The likelihood of the risk occurring is nigh impossible to reduce. To manage members' illness, the ill member needs to inform the group about their wellbeing, and if they are in working condition or not. Depending on the timetable and level of illness, another member might need to take up their work or project goals may need to be re-evaluated to reduce the chance of the project failing.
Data corruption or loss	Low	High	Files should be backed up in a cloud and version control (Git) used for programming.
Poor quality code	Medium	Medium	Code should be tested and bug-fixed frequently. Code should follow best practices and standards to mitigate bugs and errors, and improve readability.
Aggressive deadlines	Medium	Medium	Aggressive deadlines can lead to incomplete features and team member burnout. Sprints should be planned with realistic expectations and estimates in mind to reduce this risk. Missed

			features can also be implemented during a later sprint, should there be one.
Changing requirements or scope creep	Medium	Medium	Project scope should be reviewed during each sprint to identify unnecessary changes and keep it in line with customer expectations.
Poor communication between team	Medium	Low	Poor or lack of communication is bound to occur. This can lead to duplicate or mismatched work. To mitigate this risk scrums should be held often and to-do's should be updated frequently in Trello.trello board must be kept up to date to ensure all team members stay aligned and avoid duplicate work.
Huge client requirements	Medium	Low	The client might require more of the project than can be delivered. In this scenario, the project should be executed according to the group's expectations, estimates and capabilities, rather than the client's.
Configuration of tools slowing progress	Medium	Medium	Make more experienced members do the setup tasks and create instructions for others to use.

Table 2. Identified risks and mitigations.

1.3 Agile Implementation Approach

The project's agile implementation followed Scrum practices. Daily Scrums were conducted on most weekdays and occasionally on weekends. Every sprint started with a sprint planning session, in which the team discussed the scope, wrote new user stories, assigned tasks, and estimated the required effort. At the

end of every sprint, the team held a review session to review and discuss completed features and postponed tasks.

The project was divided into eight sprints, each lasting two weeks, with clearly defined goals drawn from the product backlog. The goals were updated regularly in Trello to reflect changes and to track progress. Each sprint had a Scrum master, who supervised the project's progress, and made sure that the deliverables of that sprint were met.

Throughout the process, the team practiced backlog refinement by continuously updating and reprioritizing user stories. This helped maintain a clear focus on project goals and made sure that the key features were implemented first. The agile approach enabled the team to remain flexible to changes, respond effectively to new findings, and to deliver the application within the scheduled time frame.

1.4 Scrum team roles and Responsibilities

During the project, each team member has taken the role of scrum master twice. The purpose of this fair distribution was to evenly provide team members with experience managing a diverse range of responsibilities and tasks. In addition to sprint-specific tasks, scrum masters needed to update the product backlog with new user stories based on updated requirements and manage that sprint’s Trello board. Additionally, everyone participated in the creation of sprint review and planning reports. The responsibilities and periods, during which members were scrum masters, are summarized in the below table 3:

Sprint	Scrum Master	Responsibilities	Dates
Sprint 1	Heta Hartzell	Product Owner, Project Requirements and Specs, Version Control, Project Structure, Frameworks	14.01.2025 -28.01.2025
Sprint 2	Mika Laakkonen	User Interface, Unit Testing,	28.01.2025 -11.02.2025

		Maven Dependencies, Code Coverage, Version Control	
Sprint 3	Jonne Roponen	Business logic implementation with UI and Server, Jenkins, Docker, Version Control	11.02.2025 -04.03.2025
Sprint 4	Ade Aiho	App Functionality Extension, Data Access Layer Implementation, Docker, High Level Documentation, ReadMe, Presentation	04.03.2025 -11.03.2025
Sprint 5	Jonne Roponen	UI Localization, DB Localization, Resource Identification	18.03.2025 -01.04.2025
Sprint 6	Mika Laakkonen	Code Review and Quality Analysis, Cleanup and refactoring, User Acceptance Testing Plan, Statistical Code Review Report	01.04.2025 -15.04.2025
Sprint 7	Ade Aiho	Finalized UAT Plan, User Acceptance Testing, Heuristic Usability Evaluation, Testing Report, Code Fixing	15.04.2025 -29.04.2025
Sprint 8	Heta Hartzell	Software Documentation, Project Report, Project	29.04.2025 -08.05.2025

		Presentation	

Table 3. An overview of sprints and their specifics.

Here are the links to the project Trello and Github:

- Github: https://github.com/Goliathuzzzz/OTP_Group6/tree/main
- Trello: https://trello.com/b/yD4DRsqo/product-backlog

2 Product Documentation

2.1 Introduction

Qupids Speed Dating is a locally hosted mock implementation of an automated speed dating system. It lets participants register, login, choose interested, and be matched with another participant from the database. The application includes both guest and registered users, with additional administrator privileges if needed. The application is also available in four different languages with a multilingual user interface that ensures accessibility for each available language.

Product documentation is meant for users, developers, testers, and anyone interested in software development. The documentation gives an overview of the product's design and functionality. It contains descriptions of system architecture, entity-relationship and UML diagrams, user interface mockups, and how core features are implemented. It also defines strategies used in tests and results, installation and usage instructions, and a troubleshooting guide.

2.2 Design

The user interface for Qupids Speed Dating was designed with a strong emphasis on aesthetic consistency and brand alignment. All mockups were created using Figma with a mobile layout, as the application is intended for use on mobile devices.

The visual design follows the Tatskatytöt brand, incorporating its signature "cute and girly" aesthetic. This is reflected in the chosen color palette, logo integration, and typographic style. The font choice was based on the Tatskatytöt brand, using Hoefler Text with only lowercase letters to maintain visual consistency with other brand marketing materials.

Each view in the application was prototyped and tested using Figma's interactive features to evaluate navigation flow. The mockups showcase key application screens, including language selection, registration options, matchmaking and session view. Every design decision was made with the goal of creating a friendly, intuitive, and visually consistent user experience across all supported languages and user roles.

2.3 ER Diagrams

2.3.1 Entity Relationship Diagram

The below image showcases the application's entity relationships. All entities are ultimately linked to the participant entity, which is an abstraction of all different types of users for the application (User, Guest, Admin). Participants can be matched together based on their selected interests from the five interest entities with many-to-many relationships. The match-to-participant is technically a many-to-many relationship, as each match has to have two different participants, and a participant can be in many matches. The User-entity represents both admins and typical users, and has a one-to-one relationship with the Participant-entity. Localized user-data has its own LocalizedUser-entity with a many-to-one relationship with the User-entity.

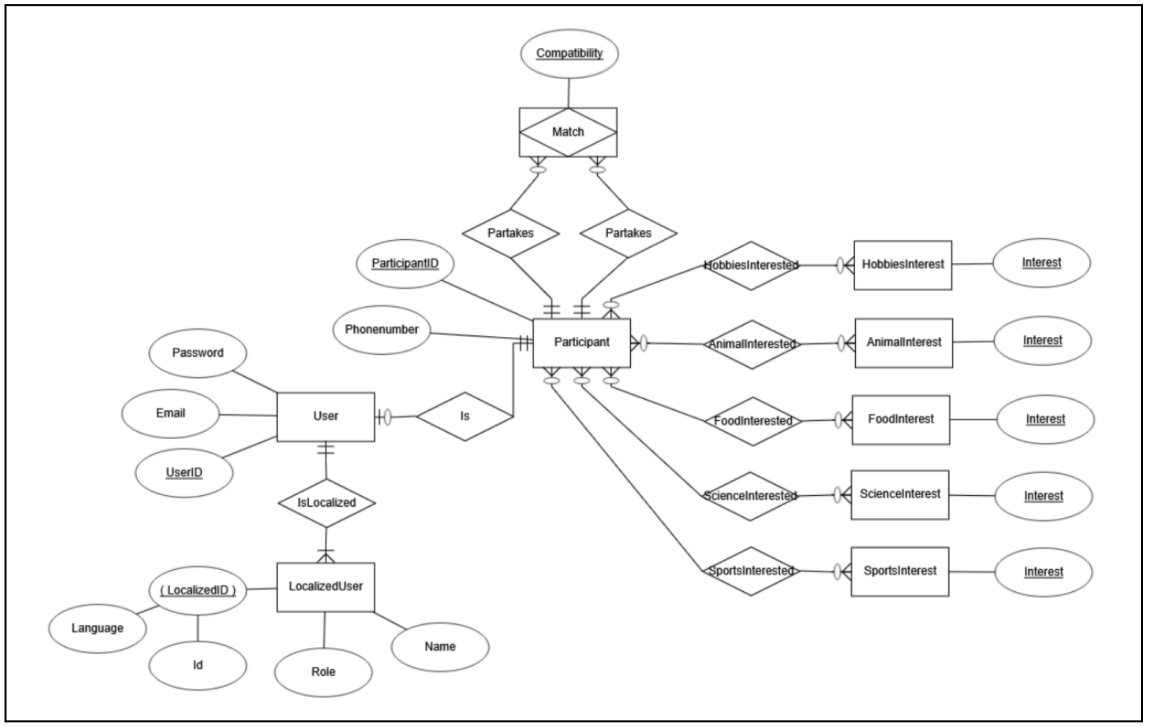

Image 1. Entity Relationship Diagram

2.3.2 Relational Tables Diagram

The below diagram shows how the entities are structured into tables. Many-to-many relationships between Participant and interest-tables each form their own middle-table.

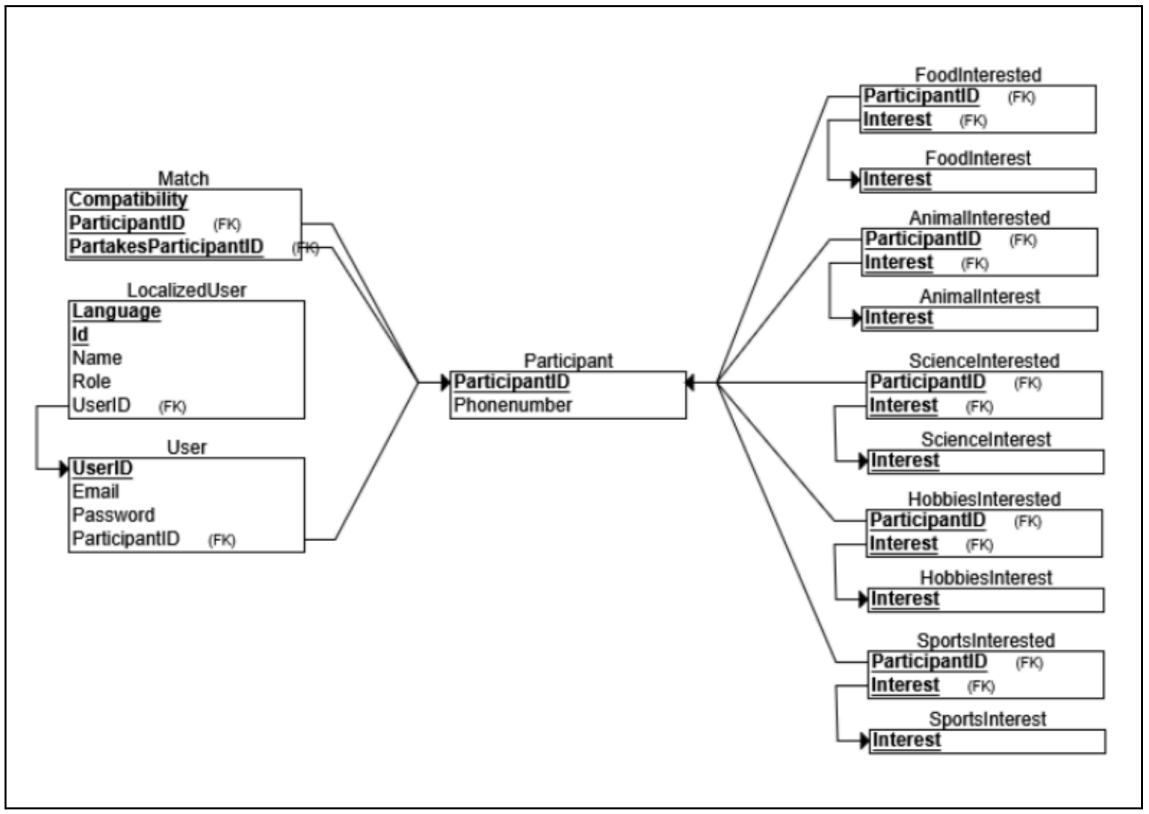

Image 2. Relational Tables Diagram

2.4 UML Diagrams

2.4.1 Use-Case Diagram

The below diagram shows the application's use-cases for all three types of user: User, Guest, and Admin (inherits user's use-cases). They can choose the language of the application, both before they use login features and from their profile. Login use-cases include logging into the application, registering into the application, and continuing as a guest by inserting your phone number. Profile use-cases include viewing your profile and editing it, while admins can also manage pairs and users. Session use-cases include completing the session and viewing details about who you were matched with. Finally, users can view the help section of the application.

Image 3. Use-Case Diagram

2.4.2 Class Diagram

The below diagram shows the application's class-structure. Like in the ER-diagram, the Participant-class is a central component of the class-structure with connections to multiple other classes. It is an abstraction of any user type. The User and Guest classes both inherit the Participant-class. The LocalizedUser-class contains localized user-data, and each User-class stores all LocalizedUsers related to it. The User-class stores all relevant non-localized user-data and has methods to manage localizations.

Both the User and Guest classes have their own controller classes to connect them to the database. Both controllers implement the IController-interface, which defines methods for database CRUD-operations. Notably, the UserController class has additional methods to manage user login and check for their existence in the database by email.

The Category-interface is a common interface for all interest-enums. It is used mainly to store different enums in a single list-structure.

The Session-classes' main function is to store selected participant interests and pass them to the Matcher-class. It has variables for all relevant data, such as selected interests, and methods to get and set them.

The Matcher-class is responsible for matching the participant with another one from the database with the matchParticipant()-method.

The Match-class is a representation of a pairing between two participants. It has a variable for storing the compatibility between them, along with getters and setters for other data. It also has its own controller class, MatchController, that implements the IController-interface to manage database CRUD-operations.

All controllers have a service-class that acts as an intermediary between the controller and database-access-layer. They implement the IService-interface to ensure each class has all important CRUD-operations. Like the UserController-class, the UserService-class has additional methods for login and checking existence by email.

The User, Guest, and Match classes each have a DAO-class (Database Access Object) for CRUD-operations. They implement the IDao-interface to ensure each CRUD-operation exists. They are accessed through their respective service-classes.

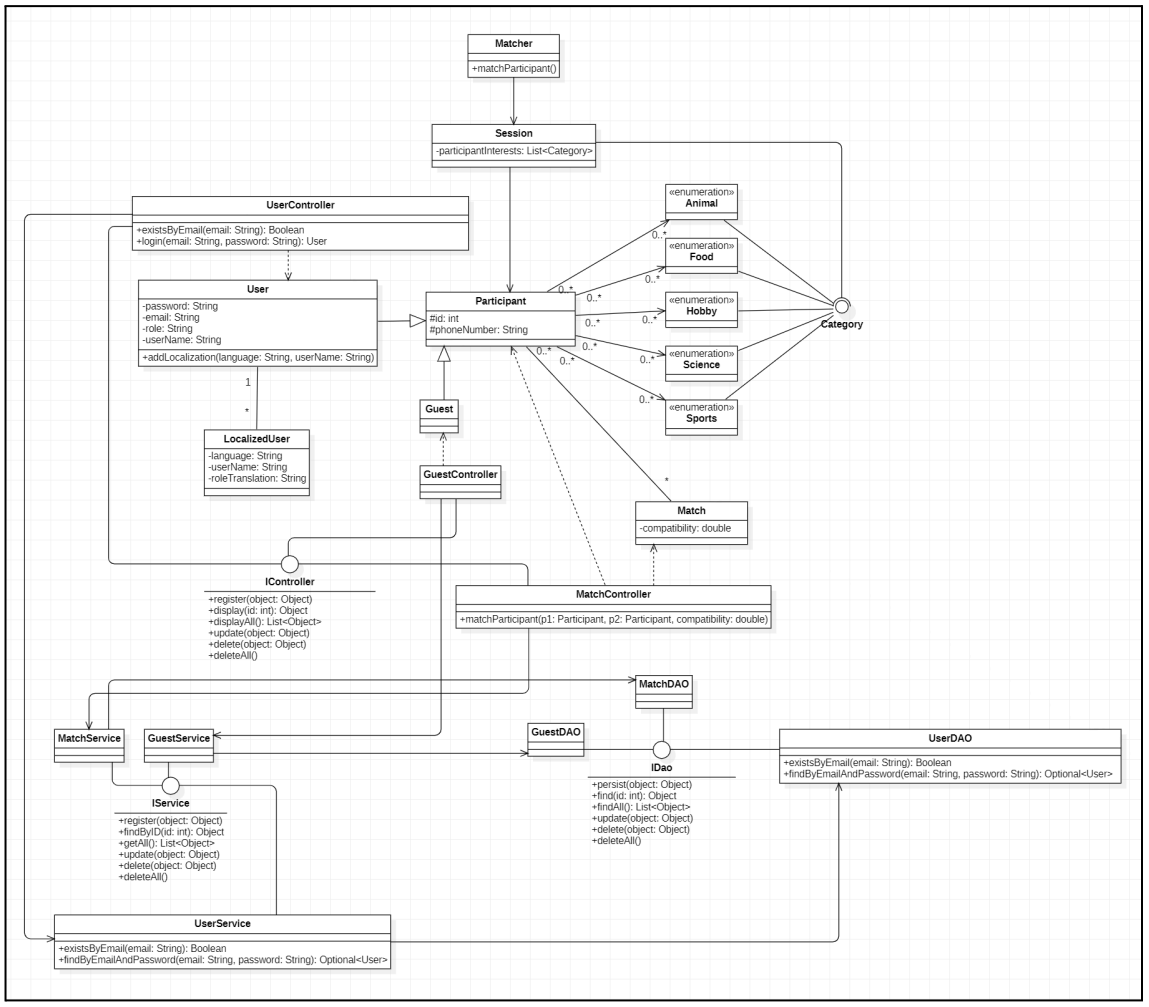

Image 4. Class Diagram

2.4.3 Packaging Diagram

The below diagram showcases how the application's source code is structured into packages. It was initially meant to follow the MVC-model. However, as the application developed further, a need for new packages that broke the model arose. These included a database-access-layer (dao-package), a services package, a configuration package to store the application's local data, and a package for utility classes.

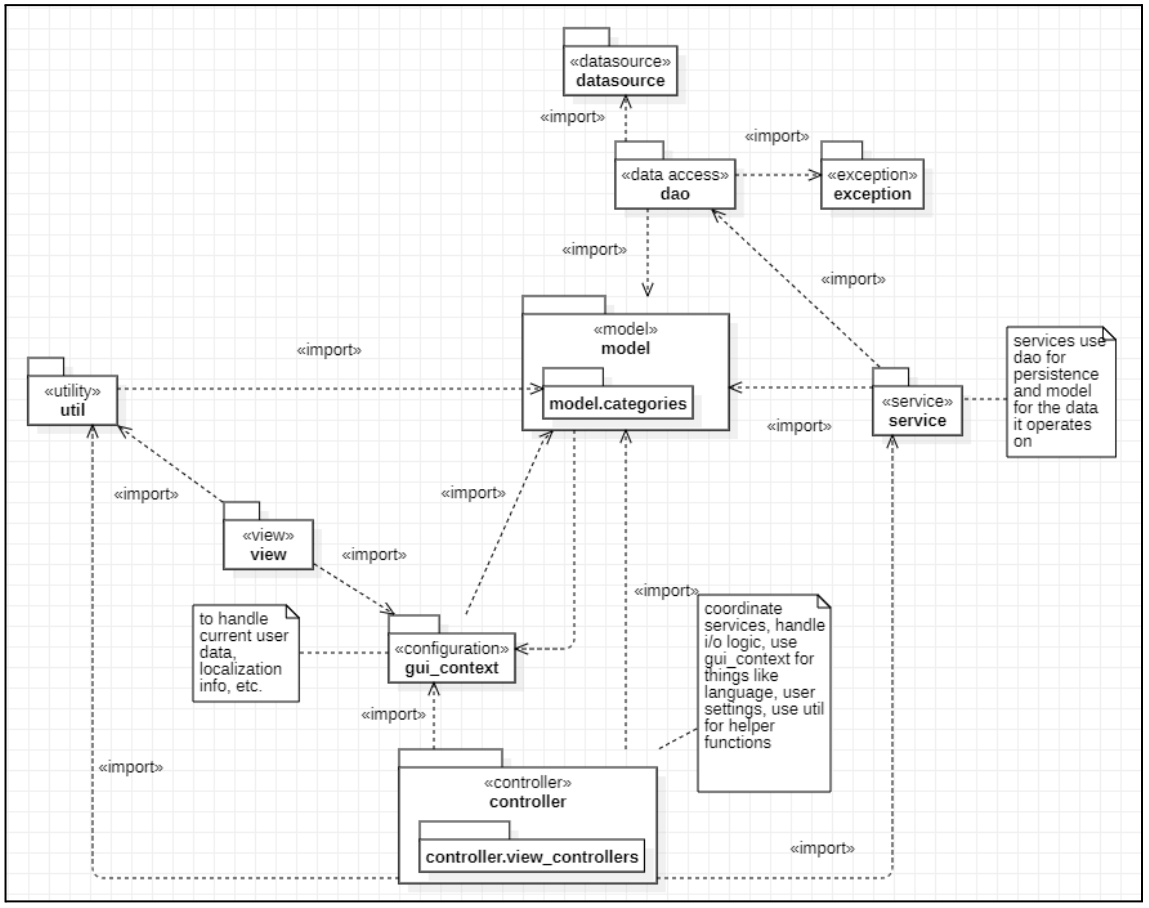

Image 5. Packaging Diagram

2.4.4 Deployment Diagram

The below diagram showcases how hardware and software components interact together in the application. Everything runs locally on a computer-device. The application's source code runs on a Java Virtual Machine (JVM) layer. Therefore, the application can compile on any device with JDK-21 installed. The database also runs locally on the same device on a MariaDB-layer. Users need to run the database_script.sql file to initialize the database on their device to start using the application locally.

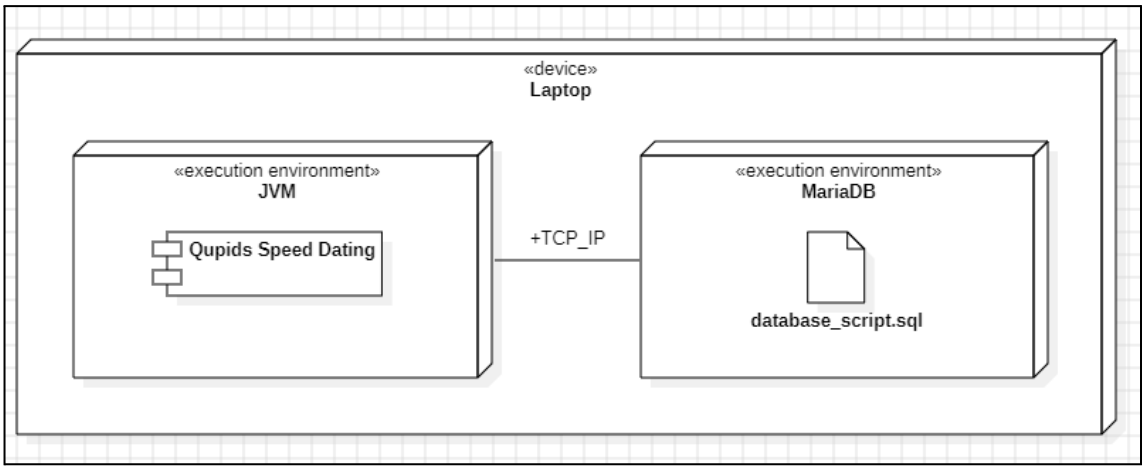

Image 6. Deployment Diagram

2.4.5 Activity Diagram

The below diagram shows how a typical user would interact with the application's "session"-feature. It describes what happens in the application on a higher level when the user interacts with it. It is sectioned into four layers: User, User Interface (UI), Business Logic, and Database. The activity's journey starts and ends at the User. During the activity, different components of the application are responsible for different actions, as is shown in the diagram. The path may encounter a white diamond-shaped object, which indicates the User can choose between multiple options.

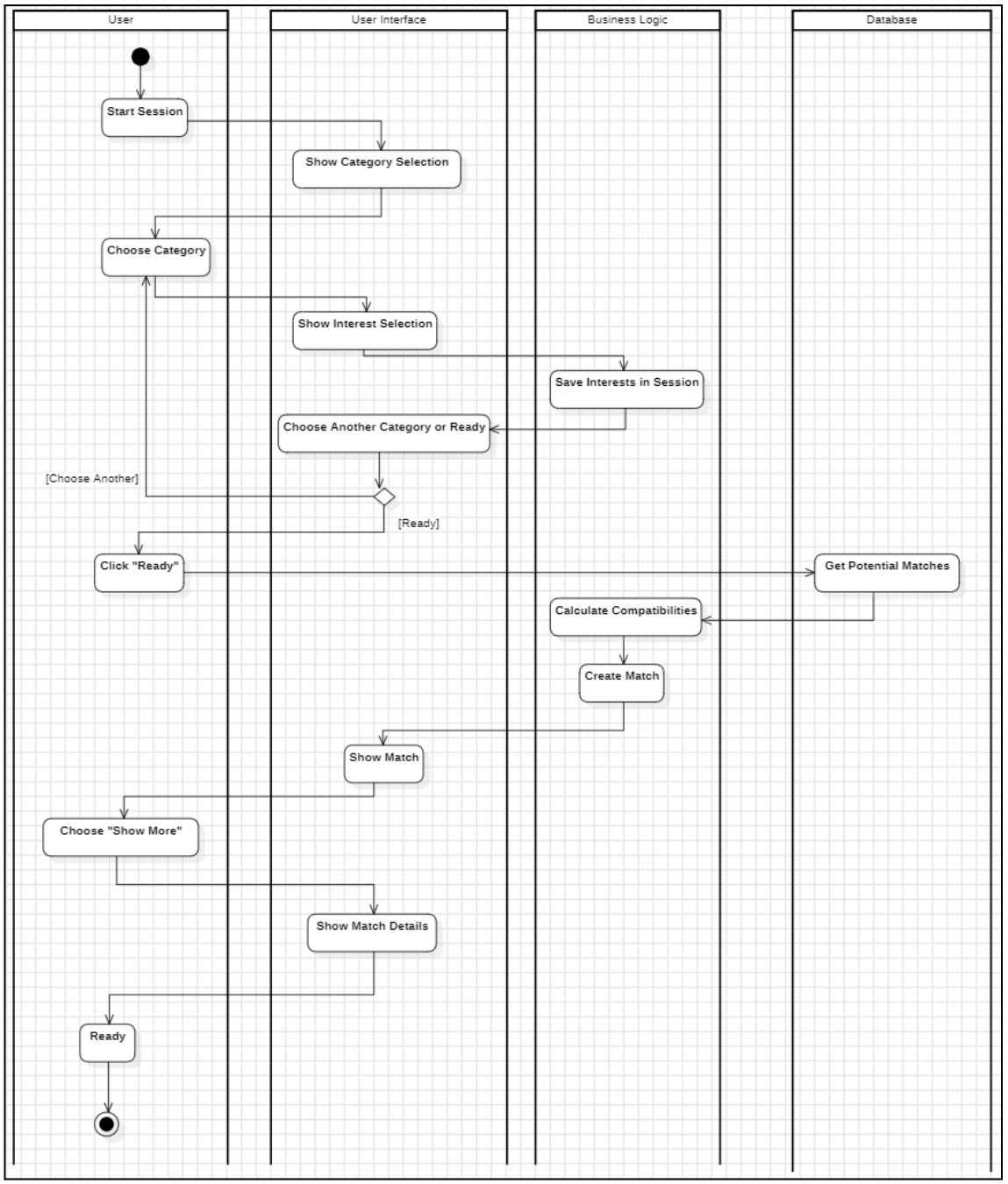

Image 6. Activity Diagram

2.4.6 Sequence Diagram

The below diagram shows the “session” activity in more detail, highlighting how several components interact with each other and what methods they use. Again, the sequence starts at the User (Lifeline 1: Participant). The User interacts with the application UI, from which method calls are sent to classes in the model package from the UI’s controllers. The filled black arrows represent method calls, while the dotted arrows represent replies. The numbers indicate

the order in which methods are called and information travels through the system.

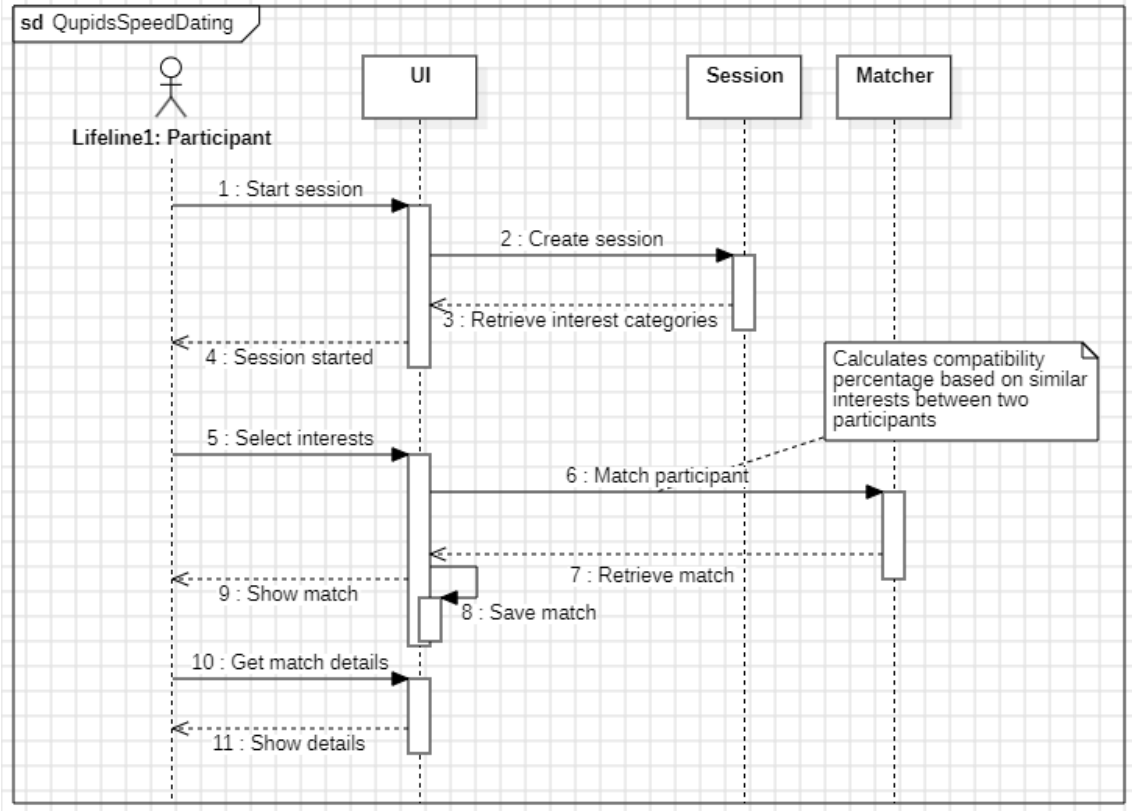

Image 7. Sequence Diagram

2.5 UI Mockups

The language selection view is the first interactive screen users encounter. It presents four toggle switches for the supported languages: Finnish, English, Japanese, and Simplified Chinese. Users can activate one language at a time, and the interface updates instantly upon selection. A **"**ready**"** button at the bottom finalizes the choice and navigates the user to the next screen. The layout is minimal yet friendly, ensuring accessibility regardless of the user's language background, as the page defaults to English.

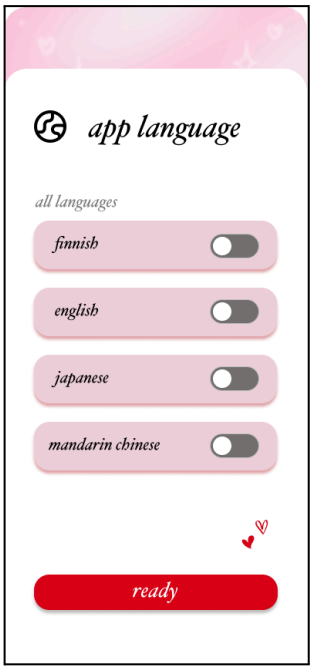

Figure x. The language selection page.

After selecting a language, users are taken to the login options view. The Tatskatytöt logo is displayed prominently at the top, followed by a title that reads “speed dating” in lowercase, reflecting the brand’s visual identity. The page offers three clear actions: users can register as new participants, log in if they already have an account, or continue as a guest using just a phone number. A back arrow at the top allows easy navigation back to the language selection screen, in case the user chooses a wrong language.

Figure x. The registration selection page.

The session page allows users to begin matchmaking by selecting their interests. It contains a main heading and a subheading that prompt users to select relevant categories such as hobbies, food, and science. Each category includes multiple clickable options that the user can select. Once selections are made, a “ready” button submits the preferences and initiates the matching algorithm. A persistent bottom navigation bar offers access to their profile, start a new session, exit or see the help section.

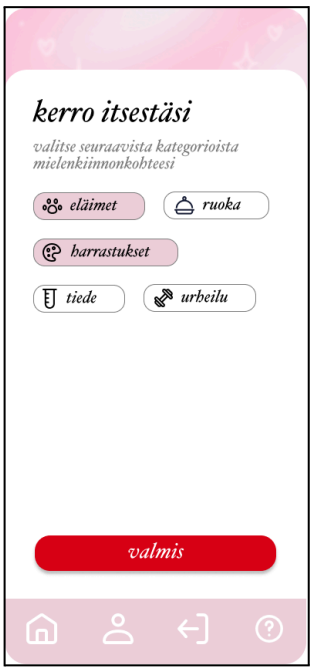

Figure x. The session page.

After matching, users are shown the match result view. The screen displays your name and your match’s name, with a large heart icon in the center showing the compatibility percentage based on shared interests. Below the heart, a short summary lists the common interests that contributed to the match. The bottom navigation bar remains visible, allowing users to return to their profile, start a new session, exit or see the help section.

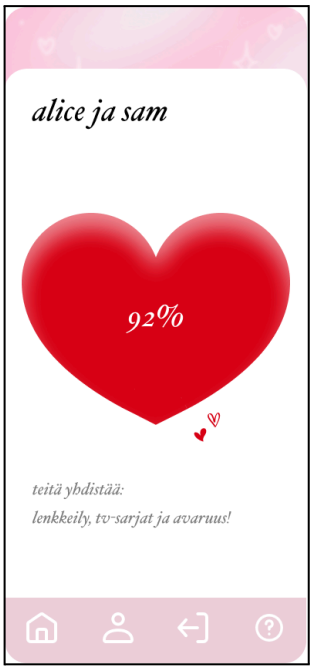

Figure x. The match percentage page.

2.6 Features Overview

Qupids Speed Dating has features that have been developed based on user stories. These features fulfill the acceptance criteria set by each user story and they will make the user experience intuitive and accessible for everyone.

The user interface is available in multiple languages such as Finnish, English, Japanese and simplified Chinese. Language can be chosen at start up or after login in the edit profile view. After selecting a language the UI will be updated instantly. All elements from simple buttons to error messages are translated to the user's preferred language.

There are different options for registration before application use. Users can either choose to register with full information such as email, phone number, and password. There is also a way to register as a guest, which is a temporary account that only requires a phone number.

Users that have registered with full information can edit their profile information from the profile page and also change their current language if they so please. Exiting the profile edit updates the information instantly and will be remembered after a logout.

Matching with other users is the main goal of the application. This feature is based on the user starting a matching session and choosing interests from different categories. These interests are saved and after the user finishes the matching session the application searches for users with similar interests and calculates the best match.

After the matching has finished, the user can choose to see more information about the match. This page will display the percentage of your matching result that has been calculated. It also displays all the interests that you two have shared.

A special type of user exists as an admin that is able to view and remove users and matches from the admin profile page. This way anyone with admin credentials can control the users and matches in the database.

The database has been designed with localization in mind. This accessibility is demonstrated by using UTF-8 encoding to store data of different languages. For example, Qupids Speed Dating offers the language options of Japanese and simplified Chinese. The database is able to store these non-latin characters without issues and is ready for further localization.

The UI has been designed with the user in mind. It is very intuitive and easy to use from start to finish. Users will find it easy to register and move onto starting the matching session and viewing their matching results. Elements are also done with localization in mind so it is accessible for all the localized languages and cultures.

2.7 Implementation Details

The structure of Qupids Speed Dating is described here with the technologies used during the project development.

Architecture Overview

The application architecture is designed based on the MVC- pattern. It also has layers such as service, DAO-, and database. With these additional layers, the system is easier to maintain, manage, test, and scale in the future.

- View acts as the interface for user input and displaying of information with JavaFX.
- Controller handles user input and passes tasks to model and service layer.
- Service is the access point for the DAO. For database interaction it gets calls from both controllers and model classes and passes them to the DAO if needed.
- DAO uses Hibernate to interact with the database.
- Database is MariaDB that stores needed information about users, guests, participants, sessions and matches.
- This separation improves maintainability, testing, and scalability in the future.

Modular Structure

The application's source code is sectioned into the following packages:

- controller: Includes Controller-classes, which act as a mediator between the model, view, and database-layer packages. It also includes FXML-controllers to manage different scenes.
- model: Has entity-classes which contain the application's business logic, such as the User- and Session-classes.
- service: Contains classes that act as a mediator between the controller- and DAO-classes.
- dao: Accesses the application's database through a JPA-entitymanager-object. It contains classes that provide model-classes CRUD operations.
- datasource: Has one singleton-class that creates the database-connection.
- view: Contains one class that initializes the graphical user interface.
- gui_context: Has two singleton-classes for managing local data. The first is for managing user- and session-related data, while the second is for managing localization settings.
- util: Contains utility classes, such as a database-seeder-class and a class for managing profile pictures.

- resources: Contains resource-files, such as configuration, FXML, and database-artefact files.
- exception: Contains one custom-exception-class for classes in the dao-package.

Technologies Used

Qupids Speed Dating was developed with and uses the technologies described below.

The software for Qupids Speed Dating was developed with Java 21. Graphical user interface was developed with JavaFX using SceneBuilder, which is used to quickly prototype and design user interfaces. The project is built using Maven, which also handles all the dependencies for the project. Hibernate is used for object-relational mapping and it handles persisting data between objects and the database. The application uses a relational MariaDB database for all the data operations.

The unit testing for the application functionality was done with JUnit 5, Mockito, and TestFx. JaCoCo is used in the pipeline to automatically generate a code coverage report after each build, which tells how much the codebase is tested. Docker is used to containerize the project and then it can be run on different environments consistently. Jenkins pipelines are used for continuous integration, which means automation of building, testing and reports.

Project management and teamwork was done with GitHub. Members worked on their personal branches for separate features to be implemented. GitHub was also used to manage documentation. Trello was used to keep track of the user stories, sprint tasks in their separate boards. This Kanban style progress board is easy to manage and control what needs to be done, what is currently being worked on, what has been done, and what is to be postponed or removed.

Several statistical code analysis tools were applied during the project. These tools analyse code without executing it. SpotBugs focuses on finding bugs and bug patterns from the source code, it categorizes these bugs and suggests changes. PMD checks the project for common flaws and issues like empty catch blocks, unused variables, and possible bugs. CheckStyle is another code

analysis tool that checks the code for proper formatting and standards. Finally there is SonarQube that is integrated with Jenkins pipeline to continuously check the code for bugs, code smells, and potential security issues.

2.8 Testing Strategy and results

The testing strategy for the Qupids Speed Dating application combined four complementary methods: User Acceptance Testing, Unit Testing, statistical code review, and usability evaluation. Together, these approaches ensured that the application met both functional and non-functional requirements, providing insight into performance, maintainability, and user experience.

Unit Testing

Unit Testing was done to test specific functions, parts and classes of the source code. All major classes from the Model and Controller were tested, including the GuiContext singleton class and some classes from the Utility package. The goal of the unit tests was to ensure all specific areas of the code worked correctly and to achieve at least 80% code coverage for the Model Package.

Used technologies included JUnit 5 for writing and structuring the tests, Mockito for generating mock data and classes to not stress or alter the database, and testFx for testing the graphical user interface (GUI). As SceneBuilder was used to build the GUI, traditional JUnit testing of the view's controller-classes was not possible, as all important methods in those classes were private. Therefore, testFx Monocle was used to test specific JavaFx scenes by interacting directly with the GUI, confirming intended functionality. They were run in headless mode as Jenkins does not have its own graphical display. In total, 88 JUnit tests were written across all tested classes. Current total coverage stands at 72%, while Model coverage stands at 88%.

User Acceptance Testing

User Acceptance Testing (UAT) was used to evaluate whether the application fulfilled its intended purpose from a user perspective. The team designed ten test cases based on the softwares key features, including registration, login, interest selection, profile editing, language settings, and administrative tasks

such as removing users or matches. Each test case included a clear description, steps, and expected outcomes. All four team members executed the tests, and every test case passed across all testers. These results suggest that the application performs as intended. However, as the testers were also the developers, the results may not reflect the experience of first-time or external users. The team was aware of this issue and recognized the value of testing with unbiased participants in the future for more realistic usability issues.

Statistical Code Review

A statistical code review was performed using three static analysis tools: Checkstyle, PMD, and SpotBugs. Checkstyle reported 328 warnings, most of which were related to formatting, naming conventions, and method complexity. While there were no outright errors, the number of warnings indicated areas where code readability and structure could be improved. PMD identified 122 issues, including some non-issues like the use of multiple variable declarations in a single line, and more interesting findings such as unused private methods and parameters, and frequent use of ‘System.out.println’ for logging. These findings pointed to opportunities for cleaning up unused code and improving logging practices. SpotBugs revealed 115 potential bugs, with the largest category being malicious code vulnerabilities. These primarily involved methods exposing or storing mutable object references, which can pose security risks if left unaddressed. Other categories included correctness, internationalization, bad practices, and thread-safety concerns. The most severe bugs have been addressed during the sprint. Altogether, the static analysis revealed that while the application is functional, there is room for improvement in maintainability, security, and in following clean coding standards.

Usability Evaluation

The third element of the testing strategy was a usability evaluation based on Nielsen's heuristics. Each team member independently evaluated the interface, identified problems, rated their severity, and suggested improvements. These were then compiled together and talked over within the team. Several issues were identified, such as missing navigation elements like back arrows on key

screens, unclear terminology in the help section, and inadequate handling of session exits. One critical finding allowed users to save empty profile fields, which could lead to broken functionality, such as being unable to log in. This issue was given the highest severity rating and was fixed right away. Other issues, such as the lack of hover effects in navigation elements and locale-insensitive string casing, were less severe but still noted for future improvement. The heuristic evaluation proved useful for uncovering design inconsistencies and potential problems in the user flow that would not necessarily show from functional testing alone.

In summary, the combined testing strategy offered a well-rounded view of the application's state. The user acceptance tests demonstrated that all core features functioned as expected. The statistical code review highlighted maintainability and security risks that are being systematically addressed. The usability evaluation provided concrete steps to enhance the user experience. Together, these efforts significantly improved the application's quality.

2.9 Installation Guide

To run the Qupids Speed Dating application on your device, your environment needs to be set up right with the required tools and configurations.

Before installation you need to have the following installed to use and run the application:

- Java 21
- Maven
- MariaDB
- Docker

Note for Windows/macOS Users:
If you're running the application with Docker locally it uses JavaFX and you need an X server installed to display the UI. VcXsrv is recommended on Windows.

After installing VcXsrv launch it with the following settings:

- *Disable access control*
- *Multiple windows* or *One large window*

Without these settings the GUI may not launch correctly from inside the Docker container.

Clone the Repository

Open a terminal and clone the project:

git clone https://github.com/Goliathuzzzz/OTP_Group6.git
cd OTP_Group6

Configure the Database

1. Start your MariaDB.
2. Create a database named:

   CREATE DATABASE qupids_db;

3. Open src/main/resources/META-INF/persistence.xml Update the username and password fields to match your local MariaDB credentials.

Build and Run the Application

To compile and start the application locally:

mvn clean install
mvn javafx:run

Running the App with Docker

Optionally you can run the application in a Docker container.

1. Ensure **Docker** and **VcXsrv** (on Windows) are running.
2. From the project root directory run:

   docker-compose up --build

This will build and start the application inside a Docker container.

2.10 Usage Instructions

Follow these steps to use Qupids Speed Dating application:

1. Launch the application
    a. Open the application via your system or by running the packaged executable.
    b. After the application starts up, select the screen to continue.
2. Select language
    a. Choose your preferred language Finnish, English, Japanese, or Chinese from the language selection screen and select “ready”.
3. Choose a login option
    a. “login” if you already have a user.
    b. “register” a new user by entering your name, email, phone number, and password.
    c. “continue” as a guest with just a phone number.
4. Start a matching session
    a. Click “start session” to begin.
    b. Choose one interest category at a time and select “ready”.
    c. Choose interests from the selected category and select “continue”.
    d. After each selection you will return to the category menu until all desired categories are completed.
5. Complete session
    a. Choose “ready” once you are done selecting interests.
    b. You will be directed to the matchmaking view.
6. View match results
    a. Select “more details” to see your best match, including compatibility percentage and shared interests.
7. Edit your profile (Users only)
    a. Select profile icon from the navigation bar down below.
    b. Select the edit icon in the profile view.
    c. Update your personal information and exit via back arrow or navigation bar to save settings.
8. Admin features (Admin users only)
    a. From the profile select “pair history” or “users”.
    b. View and delete pairs or users as needed.
9. Change language (Optional)

a. At any time, go to your profile and change your preferred language by selecting “change language?”.
b. The app updates the UI immediately to the chosen language.

2.11 Troubleshooting

This section lists some common issues users may encounter while using the Qupids Speed-Dating Application. It also introduces some common solutions to these problems, although they may not work guaranteed. Should you encounter an issue that is not in this section or the presented fixes do not work, contact qupids.troubleshooting@helpdesk.com.

Issue 1: GUI launch failure in Docker

Users may encounter a display-error when trying to launch the application from a docker container. This is because the Docker container does not have a native display, and a display needs to be forwarded to it. To fix this issue:

1. Download VcXsrv on your device.
2. Launch XLaunch. Make sure to disable access control.
3. If the docker image still fails to run, try running it with the following command: docker run -e DISPLAY=host.docker.internal:0 image_name

Issue 2: Database Connection Failure

When initially running the application, users may be unable to connect to the database. The most simple way to fix this issue is to drop and create a new database.

1. Make sure MariaDB drivers are installed on your device.
2. Run the database_script.sql script.

Issue 3: Cannot load view alert after clicking ready during session

After completing a session, you may receive an alert indicating that the next view cannot be loaded. This alert pops up if no matches are found. It could be due to the database being empty (minus current user) or that no-one else

selected any of the currently selected interests. There are two ways to resolve this issue.

Solution 1:

1. Select some more interests.
2. Try to click ready again and see if you match.

Solution 2:

1. Run the Seeder java-file from the utils package.
2. Try to complete a session and find a match again.

Issue 4: Missing JDK-21 if running application outside of docker image

Users may not have JDK-21 installed on their device. It is required to run the application outside of a docker container, as otherwise the device cannot interpret the Java-code.

Download JDK-21 from the following location and follow installer instructions: https://www.oracle.com/java/technologies/downloads/. Make sure JDK-21 is in the System Path. You may need to restart your device after installation.