TENDERPOINT

e-Tendering System for X City

Team 2

Dario Mathys – Scrum Master
ChengYao Kuang – Product Owner
Sanuji Gamagedara – Frontend Developer
Prabuddhi Kande Gamaralalage – Database Developer
Samudika Sooriyamudiyanselage – Backend Developer
Yasodha Buthpitiya Lekamalage – Documentation & Testing

Supervisor

Dan Duong Dang

VAMK University of Applied Sciences
Agile Software Development

Table of Contents

1. Executive Summary 3
2. Introduction 4
3. Project Management & Agile Process 5
   - Project Timeline 5
   - Requirements Engineering 6
     - Major User Requirements 6
     - Additional User Requests 6
     - Operational / Performance / Security Requirements 7
   - Agile Framework 8
   - Sprint Planning 9
   - Backlog Documentation 10
     - Epic 1: Tender Management & Administration 10
     - Epic 2: Bidding & Transparency Portal 11
   - Meetings & Communication 12
   - Kanban Board 13
   - Definition of Done (DoD) 14
4. System Design & Architecture 15
   - System Overview 15
   - Database Design (ERD) 16
     - Company and Category Structure 17
     - Tender and Bidding Structure 18
     - Notes on Implementation 19
   - UML Class Diagram 20
   - Sequence Diagrams 22
     - Browsing and Searching Tenders 22
     - Registering, Updating, and Deleting Tenders 23
     - Bidding and Awarding a Tender 24
5. Implementation 25
     - Development Environment 25
     - Project Structure 25
     - Backend Implementation 26
     - Frontend Implementation 28

6. Testing & Quality Assurance 31
Functional Test Cases 31
Integration Testing 34
7. Results & Evaluation 35
Achieved Functionalities 35
Comparison with Expectations 36
Limitations 36
8. Conclusion & Future Work 37
Project Reflection 37
Lessons Learned 37
Future Improvements 37
Final Statement 38
9. References 39
10. Table of figures 40
11. Table of tables 40
12. Appendix 41
Appendix A – Working Documents 42
Working Agreement – Team 2 42
Team Role Overview and Responsibilities 44
Weekly Reports 45
Weekly Report – 09.10.2025 45
Weekly Report – 15.10.2025 46
Weekly Report – 22.10.2025 47
Weekly Report – 29.10.2025 48
Weekly Report – 05.11.2025 49
Weekly Report – 12.11.2025 (Final Sprint Completion) 50
Meeting Minutes 51
Appendix B – Technical Files 57
Maven Configuration 57
Database Configuration 58
Example Code Snippets 61

1. Executive Summary

The project TenderPoint (e-Tendering System for X City) was developed as part of the Agile Software Development course at VAMK (University of Applied Sciences). The goal of this project was to design and implement an application that allows a city to manage public tenders efficiently and transparently while allowing companies to participate in the bidding process online. The system was built to make the tendering workflow more structured, reduce paperwork, and increase accessibility for both city staff and companies.

Our development followed an agile approach, divided into four sprints over eight weeks. Each sprint focused on different functional parts of the system, according to the product backlog ans userstories. The team used Trello for task tracking, GitHub for version control, and regular weekly meetings to plan, review, and adjust our work. After some challenges in assigning tasks and understanding each other's strengths, communication and teamwork improved fast and allowed us to complete all planned sprints successfully.

During the first sprint, the team implemented the login functionality for city staff and companies to make sure the access to the system works. In the second sprint, the main tender page was created, allowing users to browse, search, and filter tenders by category, type, and value. The third sprint focused on administrative features, such as creating, editing, and deleting tenders, as well as managing companies. In the final sprint, the team finalized all remaining functions, performed integration testing and improved the user interface to provide a professional experience.

The finished application, provides access for city administrators and companies. It supports complete tender lifecycle management, and includes public transparency features through open tender listings and results. The frontend was developed using HTML, CSS, and JavaScript, while the backend was implemented in Java with Servlets and integrated with a relational database. The system architecture follows a clear structure, ensuring separation of concerns and scalability.

Throughout the project, the team demonstrated strong collaboration and problem-solving skills. We encountered technical challenges such as API synchronization and database integration, but these were resolved during the sprints. By the end of the final sprint, all acceptance criteria were met, and the system passed internal testing.

Overall, this project successfully achieved its objectives. TenderPoint offers a functional and user friendly platform for managing tenders digitally. It shows how agile methods and teamwork can lead to an efficient and well designed software solution. The experience also provided valuable insights into agile project management, software development, and effective communication within a diverse team.

2. Introduction

The e-Tendering System for X City was created to support the citys plan to move its tendering process online. Until now, many of these tasks were managed manually or through basic paper work, which made the process time consuming and difficult to follow. By introducing an electronic tendering system, the city tries to make the process faster, better organized and easier accessible for companies and citizens.

The main goal of the project focused on five key areas. First, the decision of tendering, where city staff define tender items such as the name, description, schedule, and estimated price. Second, the tender notice, which involves publishing. Third, the bidding stage, where companies submit their offers and application documents in the system. Fourth, the selection of the winning bid, which makes sure that the best or lowest suitable offer is chosen within the restrictions. Finally, the disclosure of tender information, where the tender details and results are made available to the public.

The system provides different access levels for three user types. City staff can create, update, and manage tenders. Companies can browse available tenders and place their bids and citizens can view all published tenders and their results. This setup reduces paperwork, minimizes manual effort and brings more transparency to the citys tendering process.

Our team worked in an agile way. We divided the development into four sprints, each focused on a different part of the system. In the first sprint, we developed the login system for city staff and companies. The second sprint focused on browsing and searching tenders. The third sprint was about managing tenders and company information, and the final sprint completed the remaining features, testing and overall system integration.

To organize our work, we used Trello to manage the backlog and sprint progress, GitHub for version control and met weekly to discuss our progress, challenges, and next steps. This approach allowed us to adjust quickly whenever something changed and helped everyone stay on the same page.

By the end of the project, TenderPoint turned into a fully functional e-tendering platform that meets the original requirements. It offers a simple and secure way to manage tenders digitally, increases transparency for citizens and makes it easier for companies to participate in city projects.

3. Project Management & Agile Process

The development process was divided into four sprints, each lasting two weeks. Every sprint focused on specific functionality defined in the product backlog and ended with a review and retrospective. This approach allowed the team to stay flexible, respond to challenges quickly, and deliver working software at the end of each iteration.

Project Timeline

The project followed a clear timeline with four sprints. Each sprint built up on the previous one. Starting with basic system setup and user authentication, followed by tender and company management features and ending with more specific features.
The following Gantt chart illustrates the planned and executed phases of the project, showing key tasks, dependencies, and their durations.

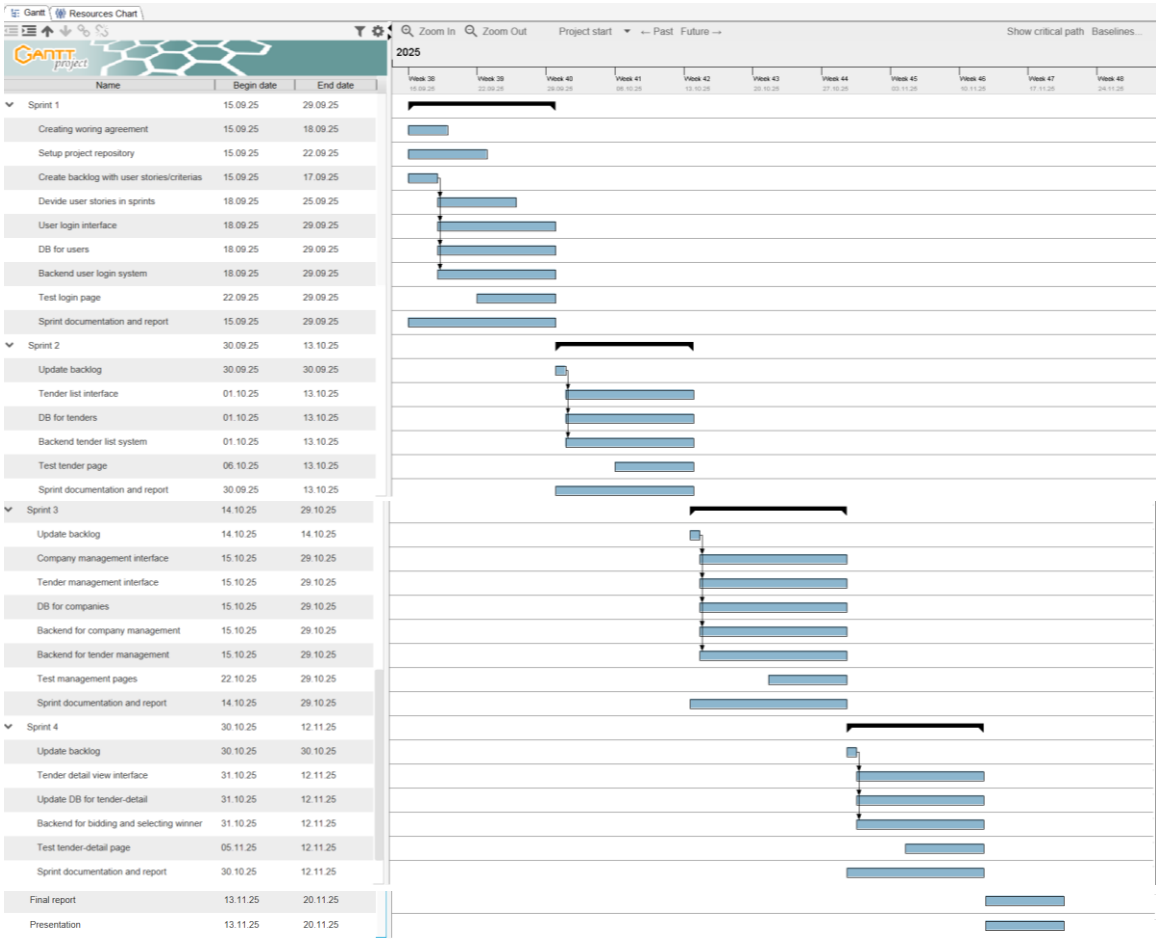

Figure 1 Gantt chart with the four sprints

This structure ensured steady progress and continuous delivery of working software throughout the project.

Requirements Engineering

The requirements for the e-Tendering System were taken directly from the project guideline. They define the core functionality, performance and operational expectations of the system and served as the basis for the product backlog and user stories.

Major User Requirements

The following are the major user requirements for the computerization of the tendering work:

1. It can browse and search the tender proposal.
2. It can register, update, and delete the tender.
3. The company can bid for the tender by inputting the bidding price and application document.
4. The city can open the tender and decide the winning company for the tender. Before deciding the winner, the city can refer to the company registration information and application documents of the company that bids for the tender.
5. The citizen can browse the tender information and refer to the list of attending companies and bidding prices, and the name of the winning company and its bidding price.

Additional User Requests

The following items are also requested by the users for the systematization:

1. The public time should be displayed on the screen of the system, and all time stamps should be managed by the public time server.
2. From the point of security, city staff and companies should log in to the system using a distributed ID and password. Each staff member of the city has their own user ID, but one company has only one user ID.
3. The system should be user-friendly because some companies are not familiar with computers.
4. The system can utilize company registration information that is registered by the company at the beginning of every financial year.
5. The system should provide a confirmation step before users actually register, update, or delete information.

Operational / Performance / Security Requirements

In addition, the following items are requirements for the operation, performance, and security of the system:

Operation

- The system is operated for 24 hours.
- When deciding user access time, consider the time required for system maintenance.

Performance

- The response should be completed within three seconds for all transactions.

Security

- Users (city staff and companies) must implement authentication by ID and password.
- The password can be input incorrectly up to five times. After five failures, the account is locked and can only be unlocked by city staff.
- Encryption and an authentication server should be considered in future improvements.

Reliability

- The system should be backed up every night.
- Duplicating or utilizing RAID should be considered in the future.

Migration

- Since this is a new system, all current paper-based work will be transferred to the digital system in one step. Old paper information will not be migrated.

These requirements define all necessary features, constraints and quality standards for the TenderPoint system and were later transformed into epics and user stories, which guided the development through all four sprints.

Agile Framework

The development of the TenderPoint system followed an agile software development approach inspired by the Scrum framework. This method was chosen because it allows flexibility and continuous improvement, which is perfect for the short project timeline.

The project was divided into four sprints, each lasting two weeks. Every sprint focused on specific functions from the product backlog and ended with a sprint review and retrospective. This iterative process helped the team to deliver working software at the end of each sprint and to make quick adjustments whenever issues come up.

The agile process included all the Scrum elements adapted to the teams workflow:

- Product Backlog:
  The product backlog was created at the beginning of the project based on the official system requirements. It contained all user stories and tasks prioritized by importance and complexity.

- Sprint Backlog:
  For each sprint, a subset of the backlog items was selected, defined as sprint goals and assigned to the team members. The sprint backlog was updated continuously on Trello and discussed at the team meeting.

- Communication:
  The team stayed in constant contact through WhatsApp and Teams. Weekly meetings were held at the library to discuss progress, challenges and next steps.

- Sprint Reviews & Retrospectives:
  At the end of every sprint, the team reviewed the completed work, demonstrated new features and reflected on improvements for the next sprint. These reviews made sure that all acceptance criteria were met.

- Definition of Done (DoD):
  Each task was considered complete only when it met the DoD, which included successful testing, peer review and updated documentation.

The agile framework made sure that the project stayed organized and transparent. Each sprint produced a functional and testable version of the system. Started with the login page in Sprint 1, to the fully integrated platform in Sprint 4. This approach allowed the team to collaborate effectively, manage priorities and deliver the final product within the planned timeline.

Sprint Planning

Sprint planning was a important part in the development of the TenderPoint system. Before each sprint, the team met to review the product backlog and decide which user stories and tasks would be implemented during the next sprint. The main goal of each sprint was to deliver a working version of the system that extended the functionality from the previous sprint.

The sprint planning meetings were led by the Scrum Master (Dario Mathys), while the Product Owner (ChengYao Kuang) defined the priorities based on the system requirements and backlog. Each team member estimated the effort required for their respective tasks and identifed potential dependencies or challenges. The workload was divided equally across the roles to ensure smooth collaboration between frontend, backend and database development.

During the meetings the team:

- Reviewed the project progress and feedback from the previous sprint.
- Selected user stories for the next sprint based on priority.
- Defined a clear sprint goal and success criteria.
- Assigned tasks to individual members and added them to Trello.
- Estimated the time and complexity of each task.

This planning approach ensured that every sprint had a realistic and achievable scope.
The sprints were organized as follows:

Sprint	Duration	Main Focus
Sprint 1	15.09 – 29.09	Setup of the environment and implementation of login functionality for staff and companies.
Sprint 2	30.09 – 13.10	Development of the tender list page, including search and filtering functions.
Sprint 3	14.10 – 29.10	Implementation of administrative functions for creating, editing, and deleting tenders and companies.
Sprint 4	30.10 – 12.11	Implementing of detail features, final testing and documentation of the complete system.

Table 1 Sprint planning table with sprint durations and main focus

During the project, the sprint planning process helped the team stay focused, react quickly to problems and improve their workflow. The use of Trello and weekly coordination meetings made it possible to keep transparency, track progress efficiently and make sure that all key functionalities were completed on time.

Backlog Documentation

The product backlog for the TenderPoint system was created at the beginning of the project and updated in every sprint. It worked as the base for sprint planning, prioritization and progress tracking. Every backlog item was defined as a user story, grouped under larger epics that represent the systems core features. The backlog was maintained and visualized in Trello, where each card included the story description, responsible members and status.

Epic 1: Tender Management & Administration

This epic covers all functionalities needed by City Staff to create and manage tenders. It ensures that tender data is accurate, secure and easily accessible for both staff and bidders.

ID	User Story	Why (Value)	Acceptance Criteria (DoD)	Priority	Story Points
P1	As a City Staff member, I want to log in to the system so that only authorized users can access administrative functions.	Protect system from unauthorized access.	Users can log in with valid ID and password. Invalid credentials show error. Locked accounts prevent login.	High	3
P2	As a City Staff member, I want to register new tenders with all relevant information so that I can manage upcoming projects.	Ensure structured tender management.	New tender form includes fields (name, description, schedule, price, category). Data stored in database.	High	5
P3	As a City Staff member, I want to update and delete existing tenders so that information remains accurate and up to date.	Keep tender data consistent.	Staff can edit or delete tenders. Changes instantly visible in list. Confirmation required before deletion.	High	8
P4	As a City Staff member, I want to view and manage companies so that I can ensure only registered businesses can bid.	Manage bidder eligibility.	Companies can be listed, edited, and locked/unlocked. Categories assigned per company.	Medium	8

ID	User Story	Why (Value)	Acceptance Criteria (DoD)	Priority	Story Points
P5	As a City Staff member, I want to open tenders and select the winning bid so that the process is transparent and fair.	Guarantee fair awarding.	System displays list of bids with prices. Staff can choose winner and record reason. Tender status updates to Awarded.	High	13

Table 2 Epic 1: Tender Management & Administration with user stories

Epic 2: Bidding & Transparency Portal

This epic includes the functionalities available to Companies and Citizens. It ensures open access to tenders, simple participation for companies and transparency for the public.

ID	User Story	Why (Value)	Acceptance Criteria (DoD)	Priority	Story Points
P6	As a Company user, I want to log in so that I can submit and manage my bids securely.	Provide secure access for companies.	Companies log in using distributed ID/password. Error shown on failure. Access restricted to company dashboard.	High	3
P7	As a Company user, I want to browse and search tenders by category so that I can find relevant projects easily.	Improve accessibility and navigation.	Category and keyword filters work correctly. Matching tenders are listed instantly.	High	5
P8	As a Company user, I want to submit a bid and attach an application document so that I can participate in the tender.	Enable digital bidding process.	Form includes price, company ID, and file upload. Submission stored in DB. Confirmation shown to user.	High	8
P9	As a Citizen, I want to view all public tenders and their results so that I can follow government activities transparently.	Promote openness and trust.	Public page lists tenders and winning companies. Data shown without login.	Medium	5

Table 3 Epic 2: Bidding & Transparency Portal with user stories

Meetings & Communication

Effective communication was a key factor in the success of the project. Since the project was developed by a student team, clear coordination and regular communication were essential to keep everyone on track and maintain progress during the sprints.

The team used a combination of online and in person meetings:

- Weekly Meetings:
  The main meetings took place nearly every wednesday in the library. These sessions were used for sprint reviews, planning the next tasks and resolving open questions. All members were expected to attend or inform the team in advance if they were unable to join.

- Ad-hoc Meetings:
  Additional shorter meetings were held when needed, especially before sprint reviews or deadlines, to make sure that all features were completed and integrated correctly.

- Online Communication:
  The team communicated via WhatsApp and Microsoft Teams, which allowed quick communication, file sharing and problem solving. This setup made it easy to ask questions or give updates outside of regular meetings.

- Task Coordination:
  The Trello board was used as the central coordination tool for task management. It was continuously updated with the sprint backlog, task progress and completion status. Each member was responsible for keeping their assigned tasks up to date.

- Version Control and Documentation:
  All source code and related documentation were shared on GitHub to make sure that every team member had access to the latest version of the project. This also simplified collaboration between frontend, backend and database development.

Overall, the communication within the team worked very good. Even though it was sometimes challenging to find meeting times, the team members showed flexibility and commitment. Continuous updates in chat and meetings made sure that issues were quickly identified and resolved.

Kanban Board

The team used Trello as a Kanban board to visualize the workflow, manage tasks and track progress during the development process. The board was used as the central coordination tool, where every team member could view, update and organize their assigned tasks.

The Trello board was divided into four main columns:

- Upcoming: Tasks planned for the next sprint or tasks waiting to be started.
- Ongoing: Tasks that were currently being worked on.
- On Hold: Tasks temporarily paused due to dependencies, feedback or pending review.
- Done: Completed tasks that met the Definition of Done (DoD), including testing and documentation.

Each task was represented as a card with details such as a short description, assigned team members and labels for the type of work (frontend, backend, database, documentation, etc.). This made it easy to see the distribution of work and the current status of every task.

The screenshot below shows the Trello board during Sprint 3, which is representative of how the team used in the project. It shows a balanced distribution of tasks across all stages of progress, including backlog updates, testing activities, frontend and backend development and documentation.

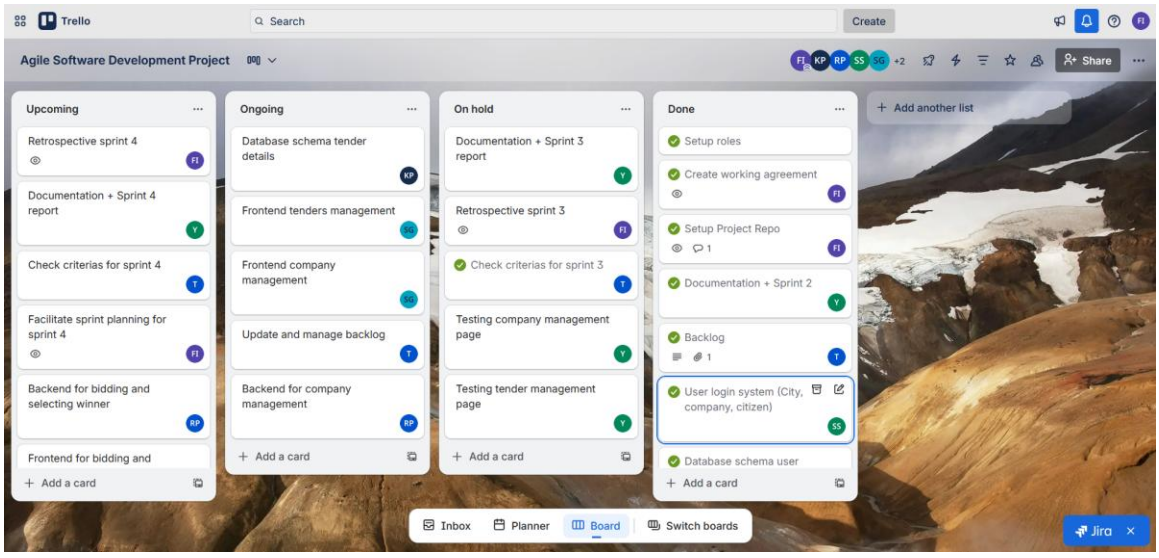

Figure 2 Trello Kanban board during sprint 3

Using Trello helped the team stay organized and prioritize work. It allowed everyone to see the project progress, identify problems early and make sure that no tasks were overlooked. Combined with weekly meetings and GitHub version control, the Kanban board was an important part of maintaining an efficient agile workflow.

Definition of Done (DoD)

To ensure quality and completeness for all tasks, the team established a shared Definition of Done at the beginning of the project. The DoD served as a checklist that guided both development and review. It made sure that each implemented feature was not only functional but also tested, documented and integrated correctly.

A task was marked as Done only when it fulfilled all of the following criterias:

1. Code completed and committed to the GitHub repository without errors.
2. Feature tested successfully.
3. Peer review performed by at least one other team member.
4. No major bugs or open issues remaining.
5. Documentation updated.
6. Trello card updated and moved to the "Done" column.
7. Approved during the sprint review, confirming that the acceptance criteria were met.

The Definition of Done was applied during all four sprints. It helped the team keep a clear standard of quality and avoid incomplete features from moving forward in the development process. This shared understanding made sure that at the end of each sprint, the system was in a stable and demonstrable state. So that its fully aligned with the agile principles of continuous delivery and transparency.

4. System Design & Architecture

System Overview

The TenderPoint system was designed as a web application following the Model-View-Controller (MVC) architecture. This structure ensures a clear difference between user interaction, business logic and data storage. It makes the system modular, maintainable, and easy to extend.

1. Presentation Layer (View)
   The presentation layer provides the user interface of the system. It has web pages used by city staff, companies and citizens to interact with the application. This layer is responsible for displaying tender information, forms and results, as well as handling user iteractions such as login, bid submissions or adding new tender details. It uses standard web technologies such as HTML, CSS, and JavaScript.

2. Application Layer (Controller)
   The application layer acts as the component between the user interface and the data model. It receives requests from the web interface, processes them and coordinates the necessary operations in the business logic (Backend). It also checks user input and makes sure that only authorized users can perform special actions, such as creating tenders or selecting winning bids. This layer returns processed data back to the presentation layer to display.

3. Data Layer (Model)
   The data layer manages all business objects and the connection to the database. It contains the systems core entities, such as companies, tenders and bids. It also has the logic for managing them. Database operations like creating, reading, updating and deleting tenders are implemented here. A centralized database connection manager makes sure that access to the underlying relational database works.

The system follows a client–server architecture, where users interact with the application through a web browser. Requests are processed on the server side, which communicates with the database to retrieve or update information. The modular design allows new functionalities, such as additional reporting or notification features, to be integrated with minimal changes to existing components. It also aligns well with agile development principles, as individual layers and components can be tested, modified and deployed independently.

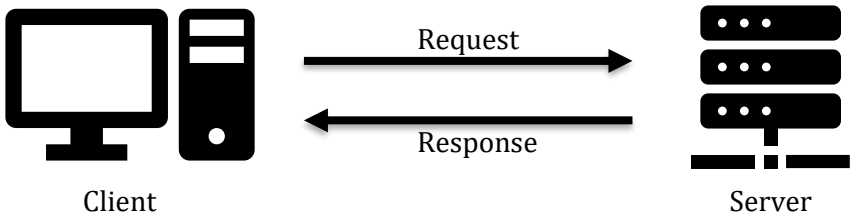

Figure 3 Client-Server arschitecture

Database Design (ERD)

The database of the TenderPoint system was designed to support the main processes defined in the project guideline. Those are managing companies, classifying them into categories, publishing tenders, receiving bids and storing submitted bid documents. The design follows a normalized relational structure and is implemented in an H2 database, which is initialized at application startup with the schema component. This makes sure that all required tables are created, even if the system is started on a new environment.

The database has six main tables:

1. companies
2. categories
3. company_categories (junction table)
4. tenders
5. bids
6. bid_files

These tables can be grouped into two logical parts:

- a company and classification part (for managing companies and in which type of tenders they can paritcipate, based on categories)
- a tender and bidding part (for managing tender publication, bidding, awarding, and attachments).

Following are both parts described in detail.

Company and Category Structure

The system allows the city to keep information about companies that participate in tenders. Each company has a business identifier (used for login), a name, a hashed password and two fields for login security (failed_attempts and locked). Because one company can operate in more than one business area, companies are linked to categories using a separate table.

Tables in this part:

- companies
  - id (primary key, internal)
  - company_uid (unique business / login ID)
  - name
  - password_hash
  - failed_attempts
  - locked
- categories
  - id (primary key)
  - name (unique; the application "merges" categories by name)
- company_categories
  - company_id (foreign key → companies.id)
  - category_id (foreign key → categories.id)

This structure creates a many-to-many relationship between companies and categories. One company can belong to several categories and one category can contain many companies. The table company_categories is only there to connect the two. It does not contain additional business data and is not shown in the chen-notation ERD.

The following figure shows the Company-Category ERD in chen-notation:

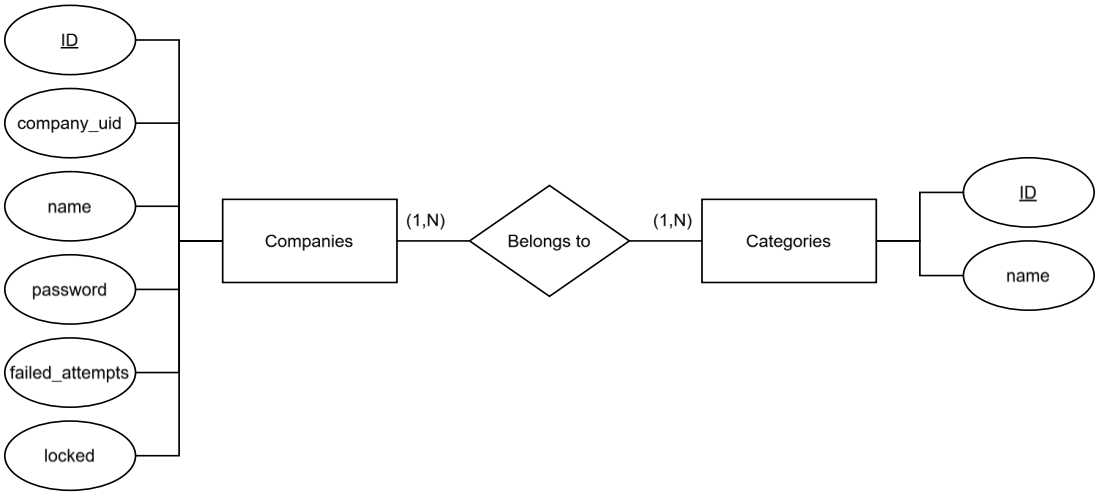

Figure 4 Company-Category ERD

Tender and Bidding Structure

The second part of the database covers the actual tendering process. A tender is created by the city and contains all relevant information such as name, publication date, closing date, description, construction term, estimated price and the current status (Open, Closed, Awarded). Companies can bid for a tender and each bid can include an attachment. When the city selects a winning bid, the tender stores a reference to that bid.

Tables in this part:

- tenders
  - id (primary key)
  - name
  - notice_date
  - close_date
  - disclose_date
  - status
  - staff_email
  - description
  - term_of_construction
  - estimated_price
  - winner_bid_id (foreign key)
  - category (used for filtering/search, not binded to category table)
- bids
  - id (primary key)
  - tender_id (foreign key)
  - company_id (stores the companies login/UID, not binded to company table)
  - company_name
  - bid_price
  - created_at
- bid_files
  - id (primary key)
  - bid_id (foreign key)
  - filename
  - content_type
  - data
  - created_at

The relationships here are:

- One tender can have many bids.
- Each bid belongs to exactly one tender.
- A bid can have zero or one attached file.
- A tender can optionally point to one of its bids as the winning bid. This will only happen, once the winner is selected by the city staff.

The following figure shows the Tender-Bidding ERD in chen-notation:

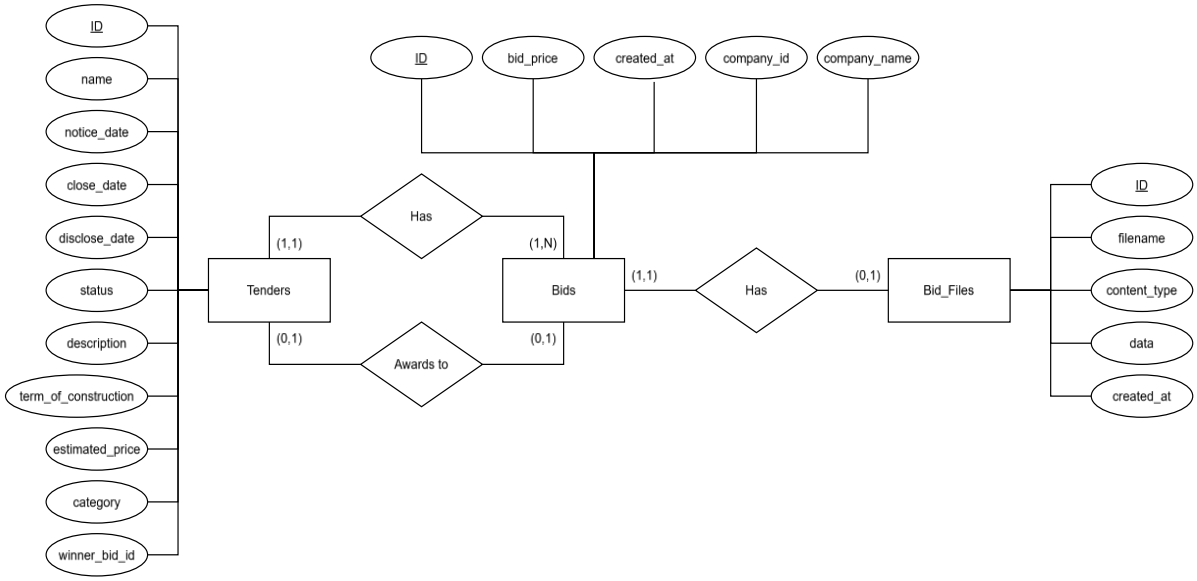

Figure 5 Tender-Bidding ERD

Notes on Implementation

- The database is created automatically at startup, so no manual SQL setup is required.
- Passwords are not stored in plain text. The system stores a SHA-256 hash.
- Login security is handled directly in the companies table. An account is locked after five failed attempts.
- Attachments are stored in the database as BLOBs (Binary Large Object) to keep everything in one place and simplify deployment.
- Some fields (such as bids.company_id) are stored as text instead of a foreign key for simplicity in this student project. This can be normalized in a future version by linking bids directly to companies.id.

UML Class Diagram

The class diagram in Figure 6 provides an overview of the main classes of the TenderPoint system and their relationships. The project follows a Model–View–Controller (MVC) architecture, where each layer is clearly separated in terms of responsibility.

- The Controller layer consists of multiple Servlets that handle incoming HTTP requests and communicate with the DAO (Data Access Object) classes. Each Servlet is responsible for a specific part of the application, such as user authentication, company management or tender handling.

  Representative Servlets include:

  - LoginServlet: manages user login and password validation.
  - CompaniesApiServlet: provides API endpoints for listing, creating and editing companies.
  - TendersApiServlet, TenderFormServlet, and TenderEditServlet: handle the creation, editing and retrieval of tenders.
  - BidsApiServlet and AdminDbServlet: handle bid submission, evaluation, and database administration.
- The Model layer contains the main domain entities, which correspond directly to the database tables. These include:
  - Company: representing a registered business entity that can participate in tenders.
  - Tender: representing a published tender including all relevant information such as notice dates, closing dates and status.
  - Bid: representing a companys offer for a specific tender.
  - User: representing city staff members or administrators who can access the system.
  - These classes store only data and minimal logic, such as constructors and field accessors.
- The Data Access layer handles all communication with the database. Every entity has a corresponding DAO class that encapsulates SQL operations:
  - CompanyDao, TenderDao, and BidDao implement methods such as find(), listAll(), create(), update(), and delete().
  - BidDao also provides specialized methods for file handling, implemented together with the helper class BidFileDao.

- The DB class provides the central connection point to the H2 database, while Schema ensures that all tables are created and updated at application startup.

- The Service layer contains additional logic, such as authentication and user management. The UserService class handles login validation and credential checks using the User model.

The relationships between these layers are clearly defined:

- Each Servlet depends on one or more DAO classes to perform database operations.
- Each DAO uses the shared DB utility to obtain database connections.
- DAO classes return instances of the corresponding Model classes.
- The UserService operates independently of the DAOs and manages application-level user data in memory.

This modular structure ensures a clear separation of concerns, improves maintainability and allows multiple developers to work on different layers simultaneously without conflicts.

Figure 6 UML Class Diagram

Sequence Diagrams

The following sequence diagrams show the main processes of the TenderPoint system. Every diagram describes how users (City Staff, Company, and Citizen) interact with the system with the web interface and how the system responds to their actions. These diagrams are taken from the project guideline and are based on the requirements. They clearly represent the implemented workflows of the application.

Browsing and Searching Tenders

Actors: City Staff, Company, Citizen

This sequence shows the interaction for all users with the system when browsing or searching tenders. City staff and companies log into the system (citizen can access tenders publicly) and use the search functionality to filter tenders by category or other attributes. The system retrieves and displays the tender list and allows users to view detailed information for each tender.

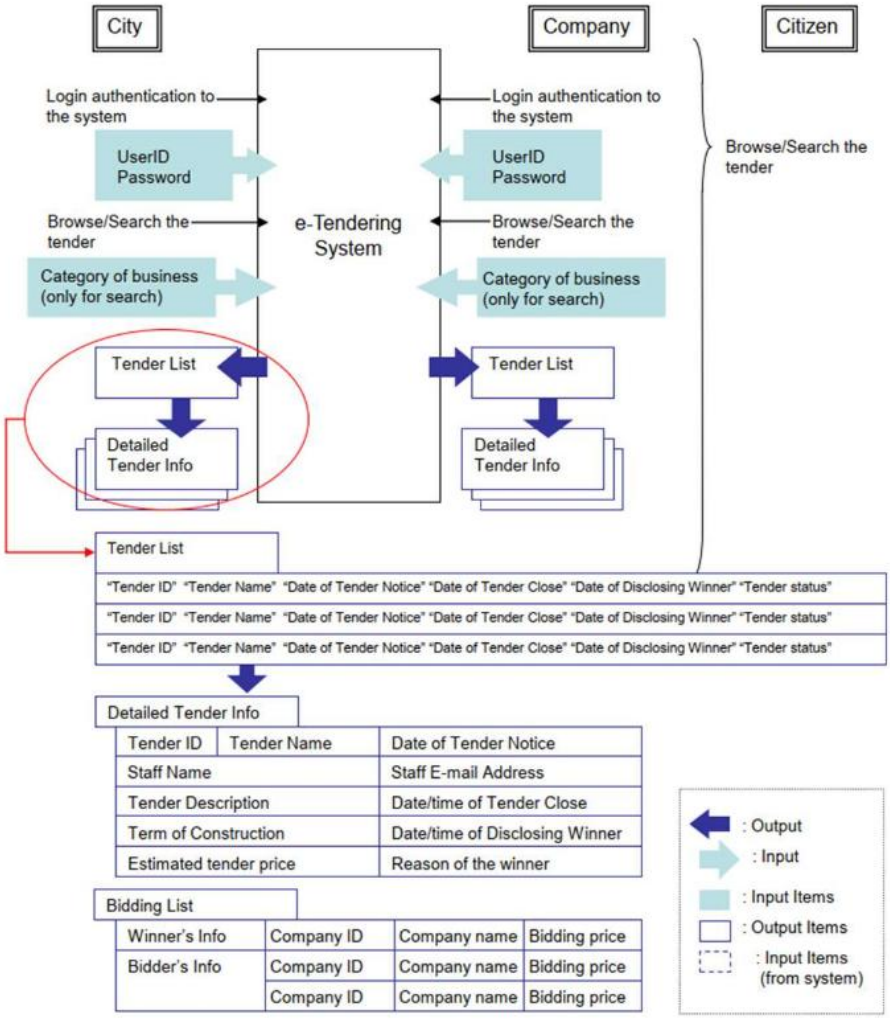

Figure 7 Sequence Diagram - Browse and Search Tender

Registering, Updating, and Deleting Tenders

Actor: City Staff

This sequence shows how city staff members create, modify or remove tenders in the system. After logging in, the city staff enters all necessary details (tender name, description, term of construction, notice and closing dates, disclosure date, category, and estimated price). The system validates the input, stores the data in the database and returns the generated Tender ID and confirmation information.

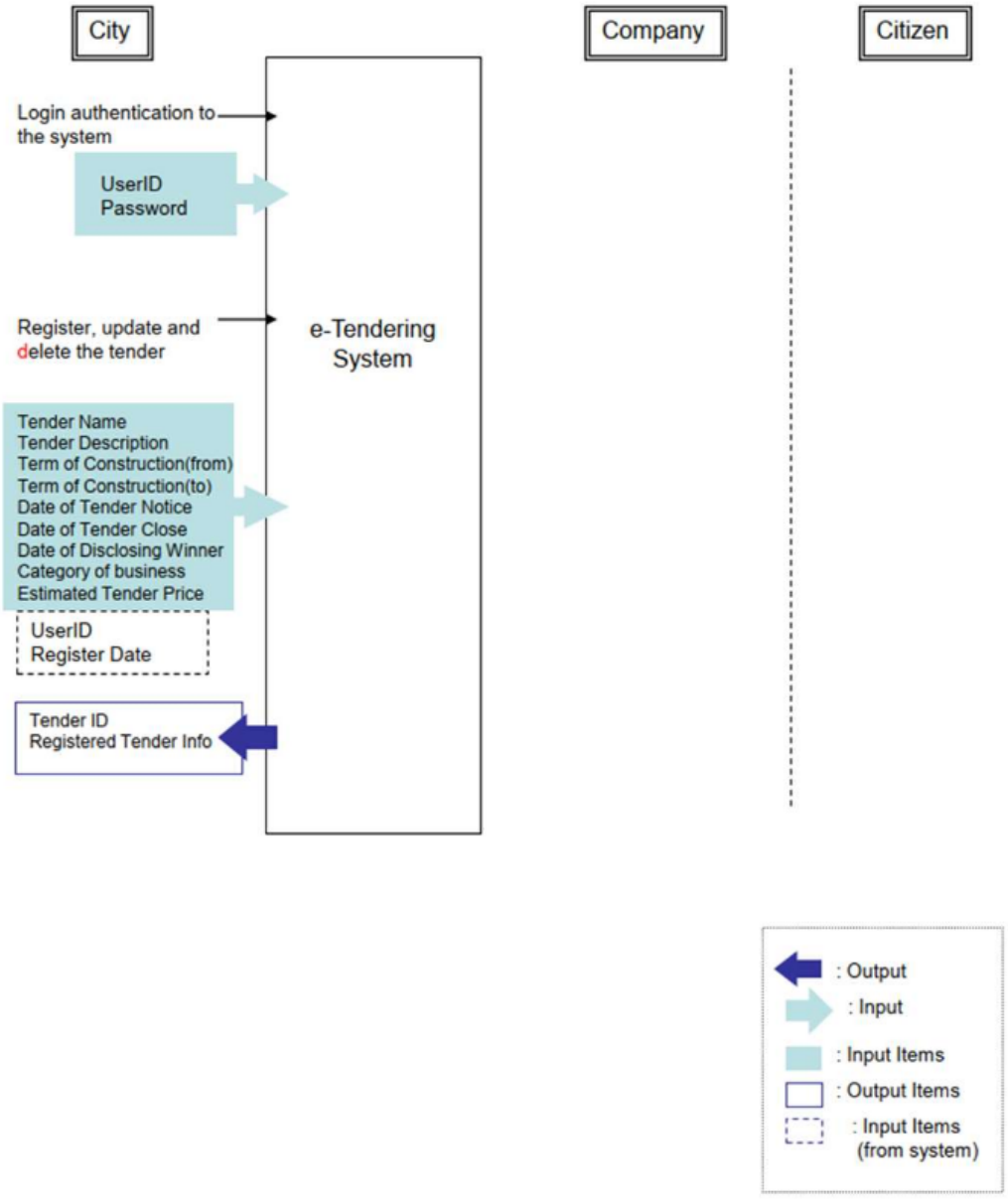

Figure 8 Sequence Diagram - Register, Update and Delete Tenders

Bidding and Awarding a Tender

Actors: City Staff, Company

This diagram shows the full bidding process from submission to awarding. Companies log in, search for tenders and submit bids by entering the bidding price and attaching an application document. Once the bidding period comes to an end, the City Staff opens the tender, review all bids and select the winning company. The system records the winning bidder, the reason for selection and updates the tender status to Awarded.

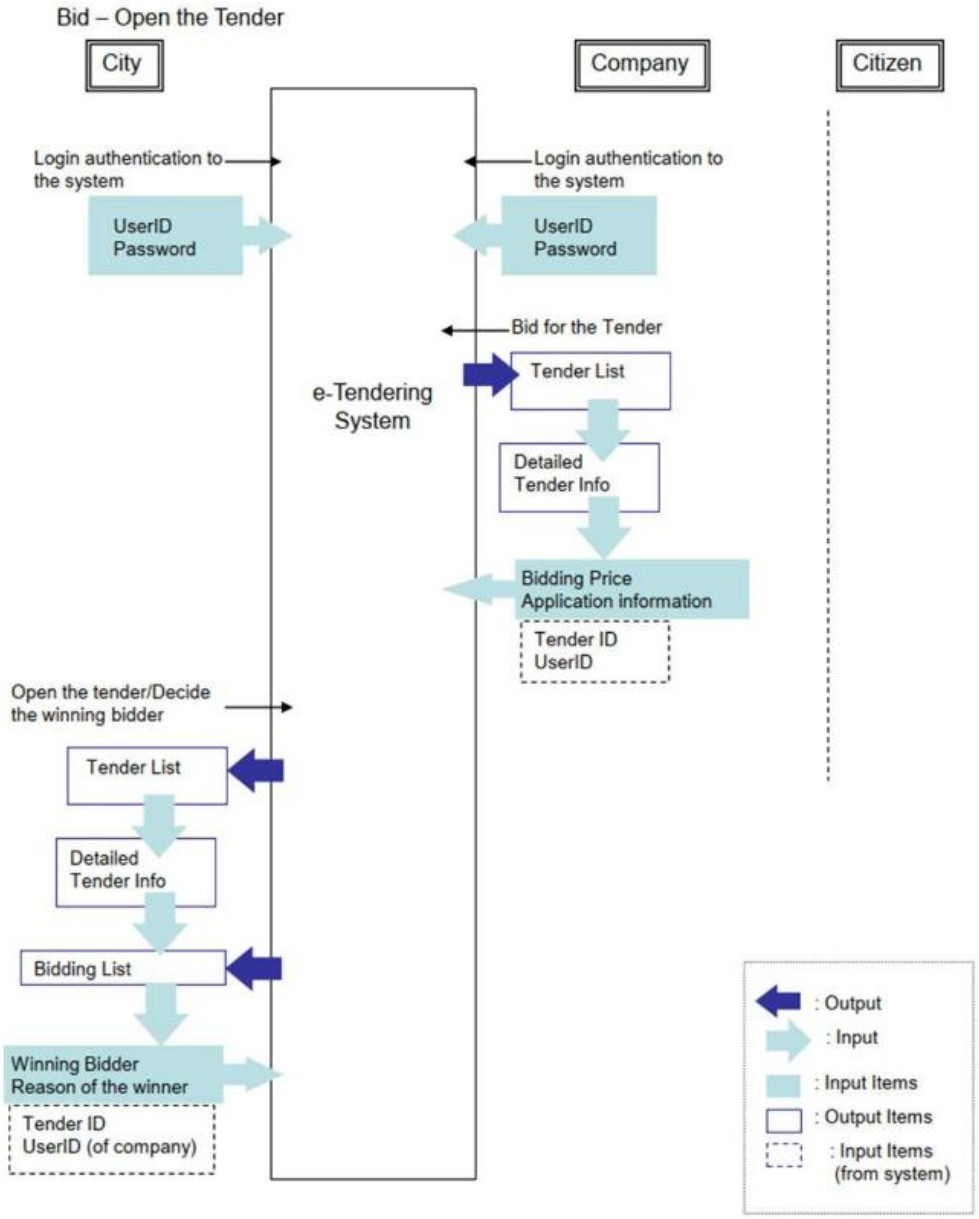

Figure 9 Sequence Diagram - Bid and Open the Tender

5. Implementation

The implementation phases of the project focused on transforming the system design into a fully functional web application. The development followed the previously defined MVC architecture, ensuring a clear separation between the presentation layer, business logic and data access. Every component was implemented to meet the requirements.

Development Environment

The project was developed in Java with Jakarta Servlets and deployed on Apache Tomcat as a web container. All database interactions were handled via JDBC with an embedded H2 database providing persistent local storage. The Maven build tool was used for dependency management, compilation and packaging of the .war file for deployment.

Technologies and tools used:

- Programming language: Java
- Framework: Jakarta Servlet API
- Database: H2 (file-based, persistent)
- Build tool: Maven
- Web server: Apache Tomcat
- Frontend: HTML, CSS, JavaScript
- IDE: IntelliJ IDEA, Visual Studio 2022
- Version control: GitHub

The combination of these technologies helped to make simple setup, portability and compatibility with the testing environment.

Project Structure

The folder structure follows the Model–View–Controller structure to maintain modularity and separation of concerns. The controller package contains all Servlets responsible for handling user requests, the model package contains business entities and DAO classes
and the webapp folder holds all frontend resources.

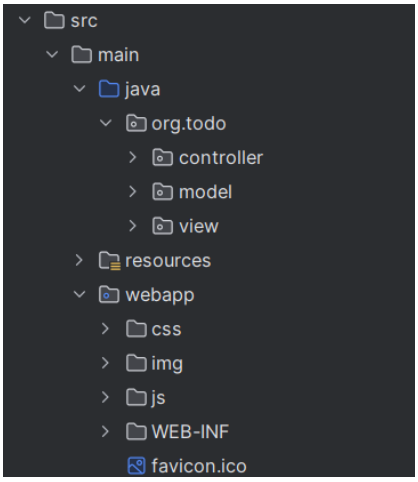

Figure 10 Project folder structure

Backend Implementation

The backend logic was implemented primarily in Java Servlets and DAO classes. The Servlets work as controllers that manage HTTP requests and responses, while the DAO classes encapsulate all database operations.

a) Login and Authentication

The login process supports both City Staff and Company users. Passwords are stored as SHA-256 hashes and the system locks an account after five failed attempts to improve security.

```
public void recordFailedLogin(String uid) throws SQLException {
    try (Connection c = DB.get();
         PreparedStatement ps = c.prepareStatement(
                 sql: "update companies set failed_attempts = failed_attempts + 1, " +
                         "locked = case when failed_attempts + 1 >= 5 then true else locked end " +
                         "where company_uid=?")) {
        ps.setString( parameterIndex: 1, uid);
        ps.executeUpdate();
    }
}
```

Figure 11 Code snippet from CompanyDao of the failed logins

This code snippet shows how the CompanyDao tracks failed logins and locks an account after multiple incorrect password attempts.

b) Tender Management

City staff can create, edit or delete tenders. The TenderFormServlet receives the form data from the frontend, constructs a Tender object and forwards it to the TenderDao class for database storage.

```
public long create(Tender t) throws SQLException {
    try (Connection c = DB.get();
         PreparedStatement ps = c.prepareStatement( sql: """
                 insert into tenders(name, notice_date, close_date, disclose_date, status, staff_email,
                                     description, term_of_construction, estimated_price,
                                    winner_reason, winner_bid_id, category)
                 values(?,?,?,?,?,?,?,?,?,?,?,?)
                 """, Statement.RETURN_GENERATED_KEYS)) {
        ps.setString( parameterIndex: 1, t.name);
        ps.setDate( parameterIndex: 2, Date.valueOf(t.noticeDate));
        ps.setDate( parameterIndex: 3, Date.valueOf(t.closeDate));
        if (t.discloseDate == null) ps.setNull( parameterIndex: 4, Types.DATE); else ps.setDate( parameterIndex: 4, Date.valueOf(t.discloseDate));
        ps.setString( parameterIndex: 5, t.status == null ? "Open" : t.status);
        ps.setString( parameterIndex: 6, t.staffEmail);
        ps.setString( parameterIndex: 7, t.description);
        ps.setString( parameterIndex: 8, t.termOfConstruction);
        if (t.estimatedPrice == null) ps.setNull( parameterIndex: 9, Types.DECIMAL); else ps.setBigDecimal( parameterIndex: 9, t.estimatedPrice);
        ps.setString( parameterIndex: 10, t.winnerReason);
        if (t.winnerBidId == null) ps.setNull( parameterIndex: 11, Types.BIGINT); else ps.setLong( parameterIndex: 11, t.winnerBidId);
        ps.setString( parameterIndex: 12, t.category);
        ps.executeUpdate();
        ResultSet keys = ps.getGeneratedKeys();
        keys.next();
        return keys.getLong( columnIndex: 1);
    }
}
```

Figure 12 Code snippet from TenderDao of the create method

This function inserts a new tender record into the database and returns the generated tender ID. The use of prepared statements ensures data safety and prevents SQL injection.

c) Bidding Functionality

Companies can submit bids for tenders. Each bid includes the company name, bid price and an optional file attachment. The following relationship is maintained:

- One tender → many bids
- One bid → zero or one file

When the City Staff selects a winner, the TenderDao updates the tenders status and stores the ID of the winning bid.

```
public List<Bid> listFor(long tenderId) throws SQLException {
    try (Connection c = DB.get();
         PreparedStatement ps = c.prepareStatement( sql: """
             select b.*,
                    (select id from bid_files bf where bf.bid_id=b.id) as attachment_id
             from bids b
             where b.tender_id=?
             order by b.bid_price asc
         """)) {
        ps.setLong( parameterIndex: 1, tenderId);
        ResultSet rs = ps.executeQuery();
        List<Bid> out = new ArrayList<>();
        while (rs.next()) out.add(map(rs));
        return out;
    }
}
```

Figure 13 Code snippet from BidDao of the list bid method

This function is used to list alle the bids for a selected tender. It gets the complete list from the database based on the tender.

Frontend Implementation

The frontend is built using HTML, CSS and JavaScript. To maintain simplicity and transparency no external frameworks is used. Every page is designed for clarity and responsiveness, featuring consistent UI elements such as forms, tables and filters.

Main pages:

- login.html: authentication page for City Staff and Companies

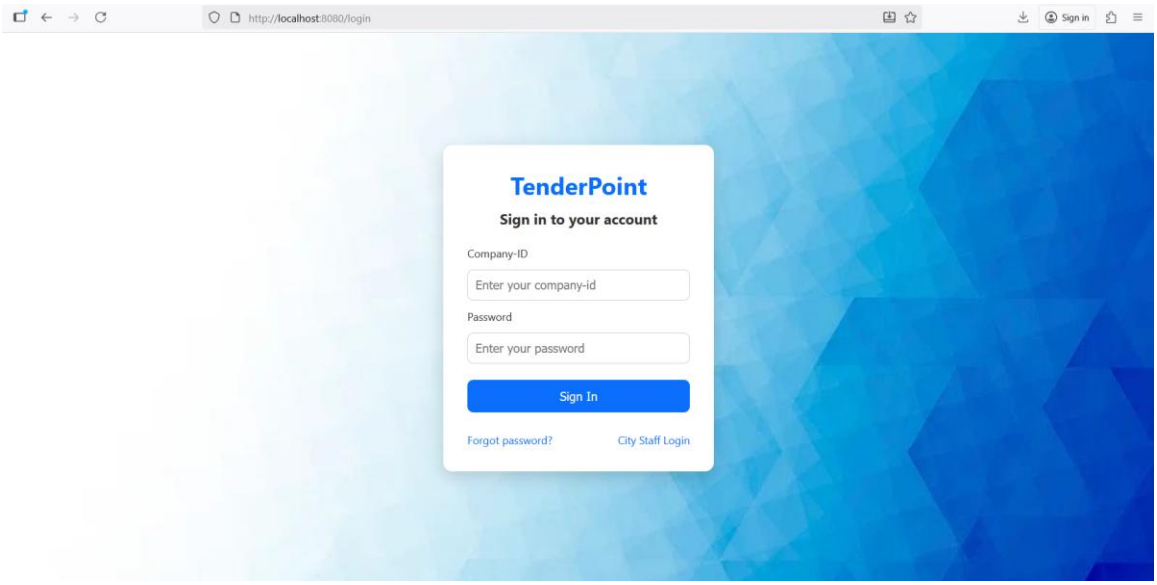

Figure 14 Frontend login.html

- tender.html: main page showing the list of tenders and filters

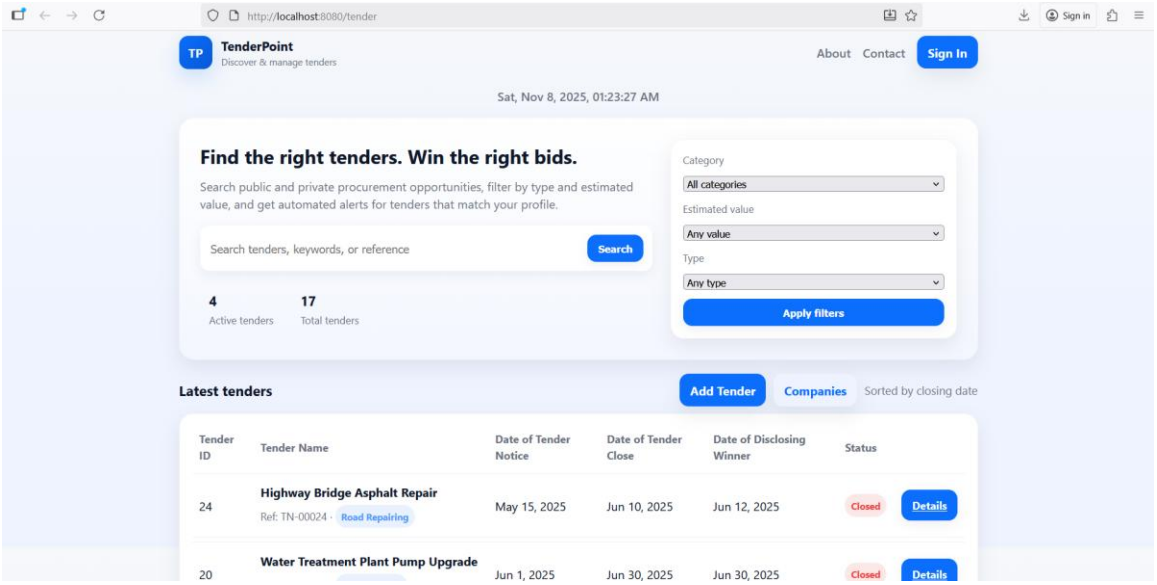

Figure 15 Frontend tender.html

- tender-form.html: form for City Staff to create new tenders

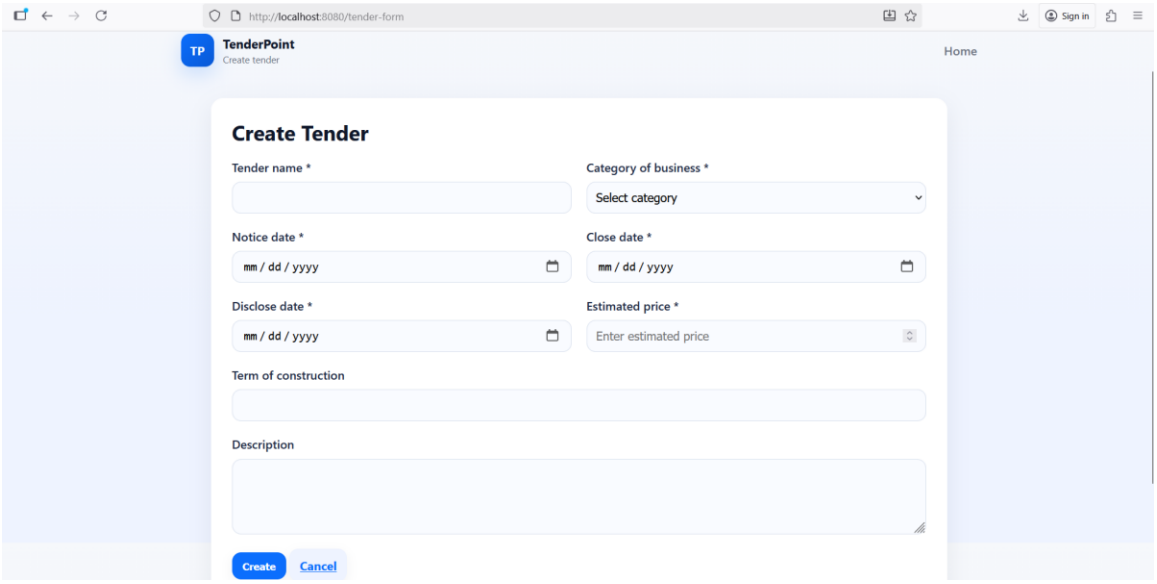

Figure 16 Frontend tender-form.html

- companies.html: page to manage companies and assign categories

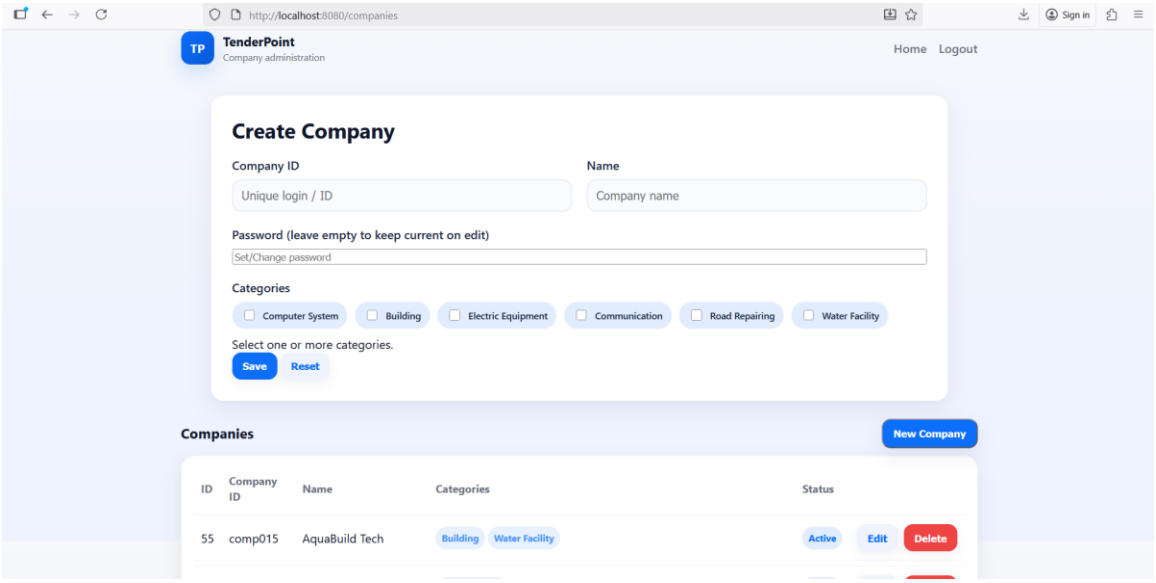

Figure 17 Frontend companies.html

- about.html: static informational pages

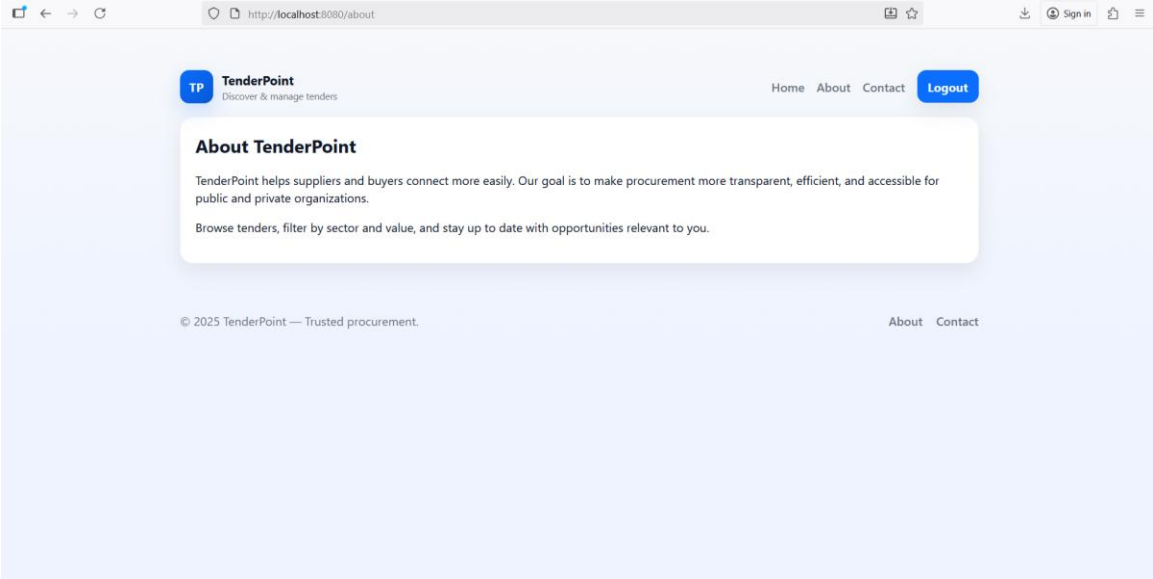

Figure 18 Frontend about.html

- contact.html: static informational pages

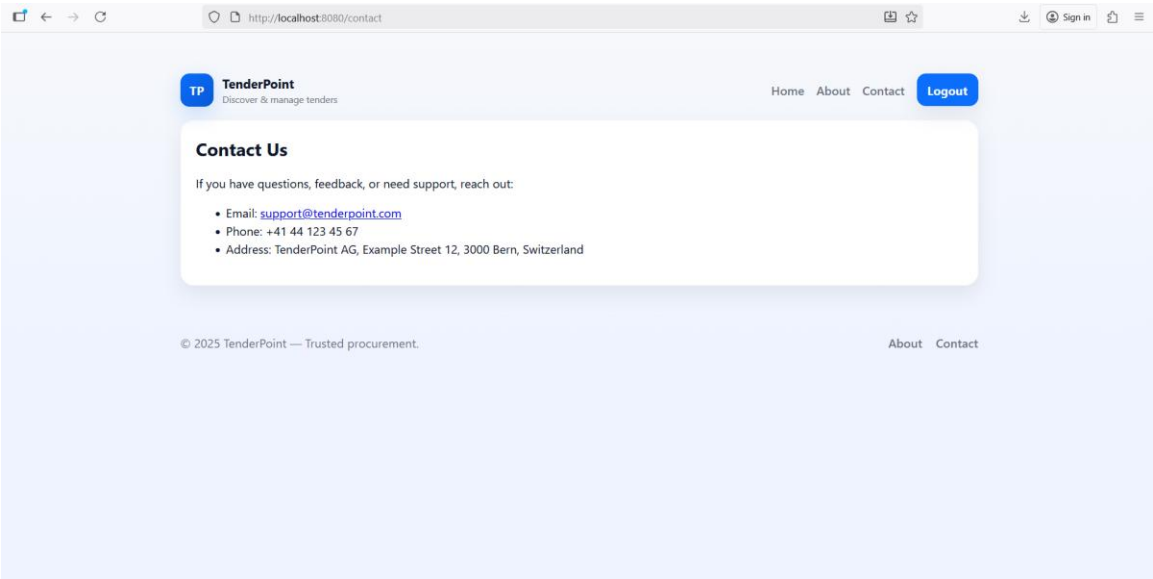

Figure 19 Frontend contact.html

Every page communicates with the backend Servlets via HTTP requests. As example, submitting the “Create Tender” form sends a POST request to /tender-form, handled by TenderFormServlet, which processes the data and stores it in the database.

6. Testing & Quality Assurance

Testing and Quality Assurance were important parts of the development process. The goal was to make sure that all implemented functions fulfilled the requirements and that the application worked correctly under different user scenarios. Testing was performed continuously during the sprints, following Agile principles, with regular sprint reviews and retrospectives to verify functionality and fix problems early.

The testing focused on four main areas:

1. Functional Testing: Verifying that every requirement from the guideline was implemented correctly.
2. Integration Testing: Making sure that communication between the Servlets, DAO classes and database worked correctly.
3. Usability Testing: Checking that users could intuitively navigate through the interface and complete their tasks.
4. Security Testing: Validating password hashing, account lockout and input validation.

All tests were performed manually by team members through the web interface in a local Tomcat environment using the embedded H2 database. Every test case includes a description, input data, expected result and actual result after execution.

Functional Test Cases

Test Case ID	Description	Input / Action	Expected Result	Actual Result	Status
TC-01	Login with valid credentials	Enter valid user ID and password for City Staff or Company	User is authenticated and redirected to dashboard	Works as expected	✅ Passed
TC-02	Login with invalid password	Enter wrong password five times	Account is locked; error message shown	Account locked after 5 attempts	✅ Passed
TC-03	Login with locked account	Try logging in with a locked account	Login is blocked and “Account locked” message displayed	Correct message shown	✅ Passed

Table 4 Test cases with description and results

TC-04	City Staff creates new tender	Fill out and submit “Create Tender” form	New tender appears in tender list and are stored in DB	Works as expected	✅ Passed
TC-05	City Staff edits existing tender	Open tender, change description, save	Updated tender info shown and persisted	Correctly updated	✅ Passed
TC-06	City Staff deletes tender	Click “Delete” on tender entry	Tender removed from list and database	Works as expected	✅ Passed
TC-07	Company logs in and views tenders	Company enters credentials and navigates to tender page	All open tenders displayed with filter options	Works as expected	✅ Passed
TC-08	Company searches tenders by category	Enter “Building” as category	Only tenders in category “Building” displayed	Works as expected	✅ Passed
TC-09	Company submits bid	Open tender, enter bid price, upload file, submit	Bid stored in DB and visible in admin view	Works as expected	✅ Passed
TC-10	Company uploads bid file	Submit bid with attached PDF	File saved in DB and linked to bid	Correct file stored	✅ Passed
TC-11	City Staff views all bids for tender	Open tender details	List of all bids shown sorted by price	Works as expected	✅ Passed
TC-12	City Staff selects winning bid	Click “Award” on chosen bid	Tender status changes to “Awarded” and reason stored	Works as expected	✅ Passed
TC-13	Citizen views tender information	Visit tender page without login	Public tenders visible with details and results	Works as expected	✅ Passed

Table 5 Test cases with description and results

TC-14	System prevents SQL injection	Enter malicious input in login or search	Input sanitized and no system error or data exposure	Input handled safely	✅ Passed
TC-15	Account unlock by admin	City Staff unlocks company account	Account becomes active again	Works as expected	✅ Passed
TC-16	Database initialization	Start server with empty DB	Tables auto-created, so app is functional	Schema created successfully	✅ Passed
TC-17	File integrity check	Upload and re-download file	File content identical to original	Works as expected	✅ Passed
TC-18	Session timeout	Leave session idle for 30 min	Automatic logout and redirect to login page	Works as expected	✅ Passed
TC-19	Input validation	Leave mandatory field empty	Warning message displayed; no submission	Validation works correctly	✅ Passed
TC-20	Cross-browser test	Test Chrome, Firefox, Edge	Layout and behavior consistent across browsers	Works as expected	✅ Passed

Table 6 Test cases with description and results

Integration Testing

Integration testing made sure that all modules and layers of the TenderPoint system worked together as planned. Integration testing verified that data and control flow between Servlets, DAO classes and the H2 database remained consistent through the application.

The system architecture naturally divides integration testing into three key layers:

1. Presentation Layer (Servlets)
2. Application / Business Logic Layer (DAOs and Services)
3. Persistence Layer (H2 Database via DB Utility)

1) Servlet–DAO Integration

Each Servlet communicates with one or more DAO classes to perform CRUD (Create, Read, Update, Delete) operations. The goal of this test phase was to verify that:

- Data entered through HTML forms was correctly validated and forwarded to DAO classes.
- DAO operations returned accurate data to the Servlet for display or redirection.
- Database transactions were committed successfully and released after completion.

Example scenario: Tender creation workflow

1. TenderFormServlet receives an HTTP POST request with tender data.
2. The Servlet creates a Tender object and calls TenderDao.create(tender).
3. TenderDao obtains a JDBC connection from DB.get() and executes an INSERT statement.
4. The generated ID is returned to the Servlet and shown on the frontend.
5. The new tender immediately appears in the tender list when reloading the page.

Result:
All interactions between Servlets and DAO classes executed successfully. Connection was managed automatically by H2 and no deadlocks, issues or data loss were observed.

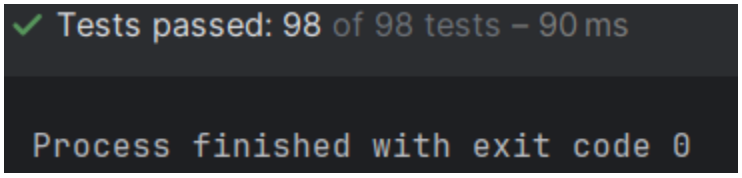

Figure 20 Test results with a completion of 98 integration tests

7. Results & Evaluation

The development of the TenderPoint system successfully fulfilled all the goals and requirements defined at the start of the project. At the end of Sprint 4, all functionalities had been implemented and tested according to the user stories and acceptance criteria.

Achieved Functionalities

The following core features were implemented and work in the final system:

Area	Implemented Functionalities
Authentication	Secure login for both city staff and companies, including password hashing and account lock after five failed attempts.
Tender Management	City staff can create, update and delete tenders with detailed information such as description, schedule, estimated price and category.
Company Management	Companies can be added, edited and assigned to multiple categories. Locked companies cannot log in or bid until a city staff unlocks the account.
Bidding System	Companies can browse tenders, submit bids with documents and view their submitted offers.
Awarding Process	City staff can review all bids, select a winner and record the reason for the selection. Tender status updates to Awarded after the winner was selected.
Public Transparency	Citizens can view open and awarded tenders and see company names and bidding prices without logging in.
Database Automation	The system automatically initializes the H2 database and maintains schema consistency on startup.

These results demonstrate that the system meets both the functional and non-functional requirements. Every major use case from the guideline was implemented and tested successfully.

Comparison with Expectations

The final system met or exceeded expectations in most aspects:

Category	Expectation	Result
Functionality	Complete implementation of all required features	✅ Achieved
Performance	System should respond within 3 seconds	✅ Achieved on local deployment
Security	Authentication and account lockout	✅ Implemented successfully
Usability	User-friendly interface and clear navigation	✅ Verified through testing
Transparency	Public access to tender and bidding data	✅ Fully available
Data Integrity	Database must remain consistent	✅ Confirmed in all integration tests

The actual performance and reliability matched the planned requirements. No major functional gaps remained at the end of the final sprint.

Limitations

Although the system functions correctly and meets all requirements, some limitations are still present due to the project scope and timeline:

- Single-user environment: The application was tested locally and not in a multi-user production environment.
- Simplified file management: Uploaded files are stored directly as BLOBs in the database, which is suitable for small projects but not optimal for scalability.
- No encryption on transport level: Communication is done via http. In a real deployment, HTTPS and encryption certificates should be used.
- Limited UI polish: The user interface is functional but could be improved with more modern design elements and accessibility standards.

Despite these limitations, the project demonstrates a complete, functional system that fulfills all given requirements.

8. Conclusion & Future Work

The project marked the successful completion of a full agile software development cycle. Starting from requirement analysis and design to implementation, testing, and delivery. Working in an interdisciplinary team allowed the members to gain practical experience with development, communication, and planning while producing a functional e-Tendering system.

Project Reflection

During the four sprints, the team demonstrated strong collaboration and adaptability. Regular meetings, sprint reviews and retrospectives helped to make sure that issues were found quickly and that the product evolved steadily. The modular system architecture proved beneficial that separating Servlets, DAOs, and Models simplified debugging, improved clarity and made integration between the frontend, backend and database easier.

Beyond the technical progress, the project also improved soft skills such as teamwork, task coordination and documentation quality. All essential aspects of software engineering. The use of tools like GitHub and Trello supported transparent collaboration and traceable decision-making.

Lessons Learned

Several key lessons could be learned from the project experience:

- A well-structured backlog and regular sprint reviews kept the project focused
- Early clarification of roles and tasks avoided duplication of work and improved team productivity.
- Frequent testing and feedback allowed early fixes and higher software quality.
- Keeping project documents aligned with implementation made reporting easier.

These insights will serve as valuable guidelines for future agile projects.

Future Improvements

Although the TenderPoint system is fully functional within the academic scope, several improvements could transform it into a more robust and better system:

1. Enable access for multiple staff and company users at the same time with connection pooling and transaction synchronization.
2. Add HTTPS communication and improved session management.
3. Replace database BLOB storage with a secure file server or cloud-based storage.
4. Modernize the frontend and implement accessibility standards.
5. Create a CI/CD workflow to automatically build, test and deploy updates.

Final Statement

In conclusion, this project demonstrates how agile methods and structured collaboration can lead to the creation of a complete, working web application. The project not only met its functional requirements, but also gave valuable practical experience for all team members in software engineering, agile project management and teamwork. With further refinement and expansion, the system could easily evolve into a scalable platform for real-world tendering processes.

9. References

VAMK University of Applied Sciences. (2025). *Agile Software Development Project – Project Guideline*. Internal document, Vaasa, Finland.

Draw.io. (2024). *diagrams.net – Free Diagram Software*. Retrieved from https://www.diagrams.net/

GeeksforGeeks. (2024). *Model View Controller (MVC) Architecture in Java*. Retrieved from https://www.geeksforgeeks.org/mvc-design-pattern/

Oracle Corporation. (2024). *Understanding Web Application Architecture*. Retrieved from https://docs.oracle.com/javaee/7/tutorial/overview003.htm

Apache Software Foundation. (2024). *Apache Tomcat 11 Documentation*. Retrieved from https://tomcat.apache.org/tomcat-11.0-doc/

Baeldung. (2024). *Introduction to Java Servlets*. Retrieved from https://www.baeldung.com/servlets

GeeksforGeeks. (2024). *Servlets in Java*. Retrieved from https://www.geeksforgeeks.org/servlets-in-java/

Eclipse Foundation. (2024). *Jakarta Servlet API 6.0 Specification*. Retrieved from https://jakarta.ee/specifications/servlet/6.0/

Oracle Corporation. (2024). *Java Platform, Standard Edition 21 API Specification*. Retrieved from https://docs.oracle.com/en/java/javase/21/docs/api/index.html

H2 Database Engine. (2024). *H2 Database Documentation*. Retrieved from https://h2database.com/html/main.html

Baeldung. (2024). *Guide to JDBC in Java*. Retrieved from https://www.baeldung.com/java-jdbc

JUnit Team. (2024). *JUnit 5 User Guide*. Retrieved from https://junit.org/junit5/docs/current/user-guide/

GitHub, Inc. (2025). *GitHub Documentation*. Retrieved from https://docs.github.com/

Trello, Inc. (2025). *Trello Help & Documentation*. Retrieved from https://support.atlassian.com/trello/

10. Table of figures

Figure 1 Gantt chart with the four sprints 5
Figure 2 Trello Kanban board during sprint 3 13
Figure 3 Client-Server arschitecture 15
Figure 4 Company-Category ERD 17
Figure 5 Tender-Bidding ERD 19
Figure 6 UML Class Diagram 21
Figure 7 Sequence Diagram - Browse and Search Tender 22
Figure 8 Sequence Diagram - Register, Update and Delete Tenders 23
Figure 9 Sequence Diagram - Bid and Open the Tender 24
Figure 10 Project folder structure 25
Figure 11 Code snippet from CompanyDao of the failed logins 26
Figure 12 Code snippet from TenderDao of the create method 26
Figure 13 Code snippet from BidDao of the list bid method 27
Figure 14 Frontend login.html 28
Figure 15 Frontend tender.html 28
Figure 16 Frontend tender-form.html 29
Figure 17 Frontend companies.html 29
Figure 18 Frontend about.html 30
Figure 19 Frontend contact.html 30
Figure 20 Test results with a completion of 98 integration tests 34
Figure 21 Meeting Minute for 24.09.2025 51
Figure 22 Meeting Minute for 13.10.2025 52
Figure 23 Meeting Minute for 22.10.2025 53
Figure 24 Meeting Minute for 29.10.2025 54
Figure 25 Meeting Minute for 05.11.2025 55
Figure 26 Meeting Minute for 12.11.2025 56

11.Table of tables

Table 1 Sprint planning table with sprint durations and main focus 9
Table 2 Epic 1: Tender Management & Administration with user stories 11
Table 3 Epic 2: Bidding & Transparency Portal with user stories 11
Table 4 Test cases with description and results 31
Table 5 Test cases with description and results 32
Table 6 Test cases with description and results 33
Table 7 Members and Roles Overview 44

12. Appendix

This appendix contains supplementary materials that support the project documentation. It provides evidence of the agile development process, technical configuration and team collaboration that were used during the project. The documents are not discussed in detail in the main report, but serve as additional proof of planning, communication and implementation activities.

The appendix is divided into three main sections:

Appendix A – Working Documents

- Team Working Agreement
- Team Role Overview and Responsibilities
- Weekly Reports
- Meeting Minutes and Sprint Notes

Appendix B – Technical Files

- pom.xml (Maven configuration file)
- jdbc.properties and logging.properties (database and logging setup)
- Example Code Snippets (TenderDao.java, TenderDaoTest.java)
- H2 configuration and SQL Dialelect config

Together, these materials demonstrate the completeness of the development process and provide transparency for both, the organization and technical parts of the TenderPoint system.

Appendix A – Working Documents

Working Agreement – Team 2

Introduction

This working agreement has been created by Team 2 to ensure smooth collaboration during our Agile software development project at VAMK. It defines our common expectations, responsibilities, and principles for working together. The agreement will help us stay organized, respect each other's contributions, and deliver high-quality results as a team.

The rules and principles outlined here are flexible and can be adapted if the team collectively agrees on changes.

1. Time and Location

Weekly meeting: Wednesday, 14:30, Library

Additional ad-hoc meetings scheduled as needed (decided together)

2. Team Norms

Attendance at meetings is expected - inform the team early if absent

Be respectful, supportive, and help when needed

All team members contribute equally to tasks

Decisions are made collectively

Use agreed communication channels and version control (GitHub)

Keep task board/backlog up to date (Trello)

3. Addressing Bugs and Problems

Address problems as early as possible

Discuss problems openly, respectfully, and constructively

Raise blockers during stand-ups or immediately in chat

Ask for help when stuck at a problem

4. Working Principles

Deliver what you commit to in each sprint

Code reviews, testing, and adherence to best practices are mandatory

Follow Definition of Done (DoD):

- Code reviewed by at least 1 peer
- Tests implemented and passed
- Documentation updated (if needed)

Be flexible for changes (adapt backlog when priorities shift)

5. Roles and Responsibilities

Scrum Master: Keeps meetings structured and timeboxed

Product Owner: Ensures backlog clarity and priorities

Developers: Responsible for implementation, testing, and quality

6. Communication and Tools

Communication via WhatsApp or Teams

File sharing via GitHub

Meetings documented briefly (decisions, tasks, next steps)

7. Conflict Resolution

Address conflicts directly and respectfully

If unresolved, bring in Scrum Master for mediation

Focus on solutions, not blame

Closing Statement

By agreeing to this document, Team 2 commits to working in a respectful, transparent, and goal-oriented manner. This agreement is a living document and may be updated as the project evolves, ensuring that our collaboration remains effective and enjoyable.

Team Role Overview and Responsibilities

The TenderPoint project was developed by Team 2. Every member was assigned a clear role and set of responsibilities based on individual strengths and project needs. The overview below shows these roles and their main duties within the team:

Team Member	Role	Main Responsibilities
ChengYao Kuang	Product Owner	Defined product vision and priorities, maintained the backlog, validated deliverables, mad sure that project outcomes met requirements.
Dario Mathys	Scrum Master	Facilitated sprint planning and reviews, ensured adherence to Agile principles, removed impediments, supported team communication and maintained documentation quality.
Sanuji Gamagedara	Frontend Developer	Designed and implemented the user interface using HTML, CSS and JavaScript. Ensured usability and responsive design. Collaborated with backend developers to integrate endpoints.
Samudika Sooriyamudiyanselage	Backend Developer	Developed core application logic and API endpoints using Java Servlets. Implemented authentication and tender management features.
Prabuddhi Kande Gamaralalage	Database Developer	Designed and maintained the H2 database schema. Implemented SQL logic, constraints and data relationships. Optimized database integration through DAO classes.
Yasodha Buthpitiya Lekamalage	Documentation & Testing Engineer	Prepared system documentation and testing plans. Conducted manual and functional tests. Recorded results for reports.

Table 7 Members and Roles Overview

This distribution of roles ensured a balanced workload and smooth collaboration across all development stages.

Weekly Reports

Below are all weekly status reports sent every Wednesday as current status updates. Every report documents the activities completed, encountered challenges and planned next steps for the following week/sprint.

Weekly Report – 09.10.2025

Team Name: Team 2
Team Members: Dario Mathys, Samudika Sooriyamudiyanselage, Yasoda Buthpitiya Lekamalage, Prabuddhi Kande Gamaralalage, Chengyao Kuang, Sanuji Gamagedara

Summary of Activities This Week (8.10):
We have started our project work. The tasks were distributed, and the first sprint has been completed. During the first sprint, we implemented the basic login functionality for the city staff and companies. This included developing the frontend, backend, and a database to store the account information. For collaboration and sharing documents or code, we use GitHub. On September 24th, we met in the library to review the first sprint and discuss the next steps.

Challenges or Question:
At the beginning, it was not easy to divide the tasks effectively since the strengths of the team members were not yet known. However, the team communicated very well, and we were able to overcome this challenge successfully.

Next Steps:
In the next step, we planned the implementation of the tenders and the search function. The tenders will be stored in a database, and a corresponding frontend will also be developed. This sprint will run until October 13th, when the next meeting will take place.

Weekly Report – 15.10.2025

Team Name: Team 2
Team Members: Dario Mathys, Samudika Sooriyamudiyanselage, Yasoda Buthpitiya Lekamalage, Prabuddhi Kande Gamaralalage, Chengyao Kuang, Sanuji Gamagedara

Summary of Activities This Week (15.10):
This week, we completed Sprint 2. We implemented the tenders main page, which now displays a list of tenders. These tenders are stored in a separate database and can be filtered. It is possible to search for them by their names or by category. Additionally, they can be filtered by price. The frontend for the main page is complete, and the buttons linked to other pages (such as Create Tender, Create Company, etc.) currently lead to placeholder pages. Our sprint review and planning meeting, to close Sprint 2 and start Sprint 3, was successful. We went through all the stories from Sprint 2 and decided that we can proceed. We also discussed the new stories and started working on the new features.

Challenges or Question:
The main challenge this week was finding a suitable date for a meeting due to the holidays. However, we eventually found a time when everyone was able to attend.

Next Steps:
In the next phase, we plan to implement the administrator functions, including creating, editing, and deleting tenders, as well as creating new companies. For this step, teamwork is crucial, as all the functions are tightly connected across the frontend, backend, and database layers, and therefore require clear communication. Date for the next meeting is 20.10.2025, but might change to the 22.10.2025.

Weekly Report – 22.10.2025

Team Name: Team 2
Team Members: Dario Mathys, Samudika Sooriyamudiyanselage, Yasoda Buthpitiya Lekamalage, Prabuddhi Kande Gamaralalage, Chengyao Kuang, Sanuji Gamagedara

Summary of Activities This Week (22.10):
This week, we continued working on the implementation of the administrator functions as planned for Sprint 3. We focused on creating, editing, and deleting tenders, as well as implementing the functionality for creating new companies. These features required close collaboration between team members, as they involved changes across the frontend, backend, and database layers. The integration between these components is progressing well, and we have already established most of the necessary connections. Our weekly meeting took place on 22.10.2025 in the library, where we discussed the current progress, clarified responsibilities, and aligned on the remaining tasks for the sprint.

Challenges or Question:
The main challenge this week was synchronizing the different parts of the implementation, especially to make sure that the data flow between frontend, backend, and database works correctly. We have some minor issues with the API communication, which the developing team is working on to fix.

Next Steps:
In the next phase, we plan to finalize the implementation of all administrator functions and perform testing to ensure that the new features work as planned. We will also prepare for the Sprint 3 closing meeting, which will take place on 29.10.2025 in the library. During that meeting, we will review the completed work, demonstrate the new features, and start the 4th and also last sprint.

Weekly Report – 29.10.2025

Team Name: Team 2
Team Members: Dario Mathys, Samudika Sooriyamudiyanselage, Yasoda Buthpitiya Lekamalage, Prabuddhi Kande Gamaralalage, Chengyao Kuang, Sanuji Gamagedara

Summary of Activities This Week (29.10):
This week, we finalized the implementation of all administrator functions, including creating, editing, and deleting tenders, as well as adding new companies. All features were tested and the results were successful in all the frontend, backend, and database layers. We had our Sprint 3 closing meeting on 29.10.2025 in the library, where we reviewed and demonstrated the completed work. The team was satisfied with the progress and overall functionality done in this sprint. After completing Sprint 3, we officially started with Sprint 4, the last sprint of the project.

Challenges or Question:
We did not encounter any major challenges this week. Communication within the team was very good, which helped us to make sure the integration across all layers works well. As a result, only a few minor adjustments were needed after testing.

Next Steps:
In the next phase, we will work on the remaining requirements and implement the final functions. We will also check the complete functionality of the application and possibly improve certain parts. Our next meeting will take place on 05.11.2025 to review the current status of the sprint. If everything goes well, we plan to finish the sprint by 12.11.2025. Once we are satisfied with the results, we will conclude the project and begin preparing for the final presentation.

Weekly Report – 05.11.2025

Team Name: Team 2
Team Members: Dario Mathys, Samudika Sooriyamudiyanselage, Yasoda Buthpitiya Lekamalage, Prabuddhi Kande Gamaralalage, Chengyao Kuang, Sanuji Gamagedara

Summary of Activities This Week (05.11):
This week, we continued working on the remaining requirements as planned in Sprint 4. Our focus was on implementing the final functions and updating the existing features. We also performed system-wide tests to make sure that all components, from the frontend through the backend to the database, are well integrated. Additionally, we started planning the final report and presentation based on our meeting at 03.11.2025. The team is on track to complete the sprint by 12.11.2025, as planned.

Challenges or Question:
No major challenges occurred this week. Some minor layout adjustments were found during testing, but they have already been fixed. Overall, collaboration and communication within the team remain very effective.

Next Steps:
In the next week, we will finalize all remaining functions and run the final tests to make sure the application runs smoothly. We will also continue working on the final report and presentation. Our goal is to finish Sprint 4 and close the project by 12.11.2025, so we can work on the presentation afterwards.

Weekly Report – 12.11.2025 (Final Sprint Completion)

Team Name: Team 2
Team Members: Dario Mathys, Samudika Sooriyamudiyanselage, Yasoda Buthpitiya Lekamalage, Prabuddhi Kande Gamaralalage, Chengyao Kuang, Sanuji Gamagedara

Summary of Activities This Week (12.11):
This week marked the completion of Sprint 4 and the official end of the TenderPoint project development phase. All remaining functionalities were finalized and the system was completely tested to ensure smooth operation across all components. The team successfully met all requirements defined in the backlog and verified the acceptance criteria for every user story. After completing the technical implementation, we focused on finalizing the project documentation. The final project report was structured, reviewed and completed collaboratively.

Challenges or Question:
No issues appeared during this week. The team maintained effective communication and workflow, which allowed us to complete the sprint smoothly and on schedule.

Next Steps:
With the development and testing phases successfully completed, the next step is to submit the final project deliverables, including the written report and the presentation video. The video will be recorded in the week of the 17th november. The project will then be ready for evaluation.

Meeting Minutes

Below are all meeting minutes for all of the happened meetings:

MEETING MINUTES

Date	24.09.2025
Time	14:00 - 15:10

Who was present?

Name		**Present?**
Dario Mathys		Present
ChengYao Kuang		Present
Sanuji Gamagedara		Present
Samudika Sooriyamudiyanselage		Present
Prabuddhi Kande Gamaralalage		Present
Yasodha Buthpitiya Lekamalage		Present

What happened since the last meeting?

Initial project setup completed.
GitHub repository and shared documentation space created.
Login functionality for city staff and companies implemented and tested.
Database structure for login data designed and created.
Backlog is updated.

Issues that need special attention

Task distribution was initially challenging, as team members strengths
were not yet fully clear.

What is planned for the next meeting - who does what?

Implement the tender list and tender search functions.
Prepare frontend components for tenders and connect them to the backend
Create database structure for tenders.

Figure 21 Meeting Minute for 24.09.2025

MEETING MINUTES

Date	13.10.2025
Time	14:00 - 14:35

Who was present?

Name		**Present?**
Dario Mathys		Present
ChengYao Kuang		Present
Sanuji Gamagedara		Present
Samudika Sooriyamudiyanselage		Present
Prabuddhi Kande Gamaralalage		Present
Yasodha Buthpitiya Lekamalage		Present

What happened since the last meeting?

Tender main page implemented with search and filter options.
Database for tenders successfully integrated.
Backlog is updated.
Buttons linking to other pages added.

Issues that need special attention

Finding a common meeting time during holidays was difficult.

What is planned for the next meeting - who does what?

Implement administrator features.
Create database structure for companies.
Add documentation for sprint 2.
Prepare Sprint 3 backlog.

Figure 22 Meeting Minute for 13.10.2025

MEETING MINUTES

Date	22.10.2025
Time	14:30 - 15:10

Who was present?

Name		**Present?**
Dario Mathys		Present
ChengYao Kuang		Present
Sanuji Gamagedara		Present
Samudika Sooriyamudiyanselage		Present
Prabuddhi Kande Gamaralalage		Present
Yasodha Buthpitiya Lekamalage		Present

What happened since the last meeting?

Work continued on administrator functionalities.
Basic CRUD operations implemented for tenders.
Initial frontend integration with backend API tested successfully.
Backlog is updated.

Issues that need special attention

Minor API synchronization issues between backend and frontend.

What is planned for the next meeting - who does what?

Finalize admin functions.
Perform integration testing.
Prepare documentation for Sprint 3 review.

Figure 23 Meeting Minute for 22.10.2025

MEETING MINUTES

Date	29.10.2025
Time	14:30 - 15:10

Who was present?

Name		**Present?**
Dario Mathys		Present
ChengYao Kuang		Present
Sanuji Gamagedara		Present
Samudika Sooriyamudiyanselage		Present
Prabuddhi Kande Gamaralalage		Present
Yasodha Buthpitiya Lekamalage		Present

What happened since the last meeting?

All administrator functionalities implemented and tested.
Sprint 3 successfully completed.

Issues that need special attention

Nothing

What is planned for the next meeting - who does what?

Implement remaining functionalities.
Prepare final integration tests.
Begin preparing project documentation and presentation material.

Figure 24 Meeting Minute for 29.10.2025

MEETING MINUTES

Date	05.11.2025
Time	14:30 - 15:00

Who was present?

Name		**Present?**
Dario Mathys		Present
ChengYao Kuang		Present
Sanuji Gamagedara		Present
Samudika Sooriyamudiyanselage		Present
Prabuddhi Kande Gamaralalage		Present
Yasodha Buthpitiya Lekamalage		Present

What happened since the last meeting?

Continued implementing final functions.
Performed overall system testing and layout adjustments.
Started structuring the final report.

Issues that need special attention

Nothing

What is planned for the next meeting - who does what?

Finish all implementation and documentation tasks.
Work on final project report.

Figure 25 Meeting Minute for 05.11.2025

MEETING MINUTES

Date	12.11.2025
Time	15:30 - 16:30

Who was present?

Name		**Present?**
Dario Mathys		Present
ChengYao Kuang		Present
Sanuji Gamagedara		Present
Samudika Sooriyamudiyanselage		Present
Prabuddhi Kande Gamaralalage		Present
Yasodha Buthpitiya Lekamalage		Present

What happened since the last meeting?

All project functionalities completed.
Full testing performed, all acceptance criteria met.
Sprint 4 officially closed.

Issues that need special attention

Nothing

What is planned for the next meeting - who does what?

Record and finalize the project presentation video.
Submit final project report and deliverables.

Figure 26 Meeting Minute for 12.11.2025

Appendix B – Technical Files

This appendix provides an overview of the technical artifacts that support the TenderPoint system. They document the configuration, database setup and code structure that allowed the successful implementation of the web application.

Maven Configuration

The project uses Apache Maven for build automation and dependency management. The pom.xml file defines the project structure, Java version, dependencies (Servlet API, H2 database, JUnit tests) and build plugins. It ensures consistency across development environments and allows for simple compilation and deployment to Tomcat.

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>todo-web-app</artifactId>
    <version>1.0</version>
    <packaging>war</packaging>

    <properties>
       <maven.compiler.source>21</maven.compiler.source>
       <maven.compiler.target>21</maven.compiler.target>
       <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
       <servlet.version>6.0.0</servlet.version>
    </properties>

    <dependencies>
       <dependency>
          <groupId>jakarta.servlet</groupId>
          <artifactId>jakarta.servlet-api</artifactId>
          <version>${servlet.version}</version>
          <scope>provided</scope>
       </dependency>
       <dependency>
          <groupId>com.googlecode.owasp-java-html-sanitizer</groupId>
          <artifactId>owasp-java-html-sanitizer</artifactId>
          <version>20240325.1</version>
       </dependency>
       <dependency>
          <groupId>com.h2database</groupId>
          <artifactId>h2</artifactId>
          <version>2.3.232</version>
       </dependency>
```

```
<dependency>
          <groupId>org.junit.jupiter</groupId>
          <artifactId>junit-jupiter</artifactId>
          <version>5.11.0</version>
          <scope>test</scope>
       </dependency>
       <dependency>
          <groupId>junit</groupId>
          <artifactId>junit</artifactId>
          <version>4.13.2</version>
          <scope>test</scope>
       </dependency>

    </dependencies>

    <build>
       <finalName>${project.artifactId}</finalName>
       <plugins>
          <plugin>
             <groupId>org.apache.maven.plugins</groupId>
             <artifactId>maven-war-plugin</artifactId>
             <version>3.4.0</version>
          </plugin>
       </plugins>
    </build>

</project>
```

Database Configuration

TenderPoint uses an H2 relational database, chosen for its simple integration and compatibility with JDBC. The configuration specifies connection URL, user credentials and automatic schema creation. The database schema is initialized at startup via the Schema.java class, which ensures that all required tables are present.

```
package org.todo.model.db;

import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;

/** Creates/updates the DB schema if missing. Call once at startup. */
public final class Schema {
    private Schema() {}

    public static void ensure() {
       try (Connection c = DB.get(); Statement st =
c.createStatement()) {

          // TENDERS TABLE — covers all fields present in tender-
form.html
          st.execute("""
                create table if not exists tenders(
                  id identity primary key,
                  name varchar(255) not null,
```

```
                    notice_date date not null,
                    close_date date not null,
                    disclose_date date,
                    status varchar(32) not null default 'Open',
                    staff_email varchar(255),
                    description text,
                    term_of_construction varchar(255),
                    estimated_price decimal(18,2),
                    winner_reason text,
                    winner_bid_id bigint
                );
                """);

            // >>> NEW: add tender category (single value) <<<
            st.execute("alter table tenders add column if not exists
category varchar(100);");

            // BIDS TABLE
            st.execute("""
                    create table if not exists bids(
                      id identity primary key,
                      tender_id bigint not null,
                      company_id varchar(64) not null,
                      company_name varchar(255) not null,
                      bid_price decimal(18,2) not null,
                      created_at timestamp not null default
current_timestamp(),
                      foreign key (tender_id) references tenders(id) on
delete cascade
                    );
                """);

            // File attachments table (already added earlier)
            st.execute("""
                    create table if not exists bid_files(
                      id identity primary key,
                      bid_id bigint not null unique,
                      filename varchar(255),
                      content_type varchar(255),
                      data blob not null,
                      created_at timestamp not null default
current_timestamp(),
                      foreign key (bid_id) references bids(id) on delete
cascade
                    );
                """);


        } catch (SQLException e) {
            throw new RuntimeException("Schema initialization failed",
e);
        }
    }
}
```

The DB.java class manages the JDBC connections to the database :

```
package org.todo.model.db;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

/** Central place to obtain JDBC connections. */
public final class DB {
    // File-based H2 database in your user home (persists between
restarts)
    private static final String URL =
"jdbc:h2:~/tenderpoint_db;AUTO_SERVER=TRUE";
    private static final String USER = "sa";
    private static final String PASSWORD = "";

    private DB() {}

    static {
       try {
          Class.forName("org.h2.Driver");
       } catch (ClassNotFoundException e) {
          throw new RuntimeException("H2 driver not on classpath", e);
       }
    }

    public static Connection get() throws SQLException {
       return DriverManager.getConnection(URL, USER, PASSWORD);
    }
}
```

Example Code Snippets

The following files show the projects coding structure and programming style. They are representative examples showing how data access, business logic and presentation layers interact.

TenderDao.java : Data-access class handling Create, Read, Update and Delete operations for tender:

```
package org.todo.model.tender;

import org.todo.model.db.DB;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class TenderDao {

    private Tender map(ResultSet rs) throws SQLException {
        Tender t = new Tender();
        t.id = rs.getLong("id");
        t.name = rs.getString("name");
        t.noticeDate = rs.getDate("notice_date").toLocalDate();
        t.closeDate = rs.getDate("close_date").toLocalDate();
        Date dd = rs.getDate("disclose_date");
        t.discloseDate = (dd == null ? null : dd.toLocalDate());
        t.status = rs.getString("status");
        t.staffEmail = rs.getString("staff_email");
        t.description = rs.getString("description");
        t.termOfConstruction = rs.getString("term_of_construction");
        t.estimatedPrice = rs.getBigDecimal("estimated_price");
        t.winnerReason = rs.getString("winner_reason");
        long wb = rs.getLong("winner_bid_id");
        t.winnerBidId = rs.wasNull() ? null : wb;
        t.category = rs.getString("category");
        return t;
    }

    public List<Tender> listAll() throws SQLException {
        try (Connection c = DB.get();
             PreparedStatement ps = c.prepareStatement("select * from
tenders order by close_date asc")) {
            ResultSet rs = ps.executeQuery();
            List<Tender> out = new ArrayList<>();
            while (rs.next()) out.add(map(rs));
            return out;
        }
    }

    public Tender find(long id) throws SQLException {
        try (Connection c = DB.get();
             PreparedStatement ps = c.prepareStatement("select * from
tenders where id=?")) {
            ps.setLong(1, id);
            ResultSet rs = ps.executeQuery();
```

```
            if (rs.next()) return map(rs);
            return null;
        }
    }

    public long create(Tender t) throws SQLException {
        try (Connection c = DB.get();
             PreparedStatement ps = c.prepareStatement("""
                     insert into tenders(name, notice_date, close_date,
disclose_date, status, staff_email,
                                         description, term_of_construction,
estimated_price, winner_reason, winner_bid_id, category)
                     values(?,?,?,?,?,?,?,?,?,?,?,?)
                     """, Statement.RETURN_GENERATED_KEYS)) {
            ps.setString(1, t.name);
            ps.setDate(2, Date.valueOf(t.noticeDate));
            ps.setDate(3, Date.valueOf(t.closeDate));
            if (t.discloseDate == null) ps.setNull(4, Types.DATE); else
ps.setDate(4, Date.valueOf(t.discloseDate));
            ps.setString(5, t.status == null ? "Open" : t.status);
            ps.setString(6, t.staffEmail);
            ps.setString(7, t.description);
            ps.setString(8, t.termOfConstruction);
            if (t.estimatedPrice == null) ps.setNull(9, Types.DECIMAL);
else ps.setBigDecimal(9, t.estimatedPrice);
            ps.setString(10, t.winnerReason);
            if (t.winnerBidId == null) ps.setNull(11, Types.BIGINT); else
ps.setLong(11, t.winnerBidId);
            ps.setString(12, t.category);
            ps.executeUpdate();
            ResultSet keys = ps.getGeneratedKeys();
            keys.next();
            return keys.getLong(1);
        }
    }

    public void update(Tender t) throws SQLException {
        try (Connection c = DB.get();
             PreparedStatement ps = c.prepareStatement("""
                     update tenders set name=?, notice_date=?,
close_date=?, disclose_date=?, status=?, staff_email=?,
                                        description=?, term_of_construction=?,
estimated_price=?, winner_reason=?, winner_bid_id=?, category=?
                     where id=?
                     """)) {
            ps.setString(1, t.name);
            ps.setDate(2, Date.valueOf(t.noticeDate));
            ps.setDate(3, Date.valueOf(t.closeDate));
            if (t.discloseDate == null) ps.setNull(4, Types.DATE); else
ps.setDate(4, Date.valueOf(t.discloseDate));
            ps.setString(5, t.status);
            ps.setString(6, t.staffEmail);
            ps.setString(7, t.description);
            ps.setString(8, t.termOfConstruction);
            if (t.estimatedPrice == null) ps.setNull(9, Types.DECIMAL);
else ps.setBigDecimal(9, t.estimatedPrice);
            ps.setString(10, t.winnerReason);
```

```
            if (t.winnerBidId == null) ps.setNull(11, Types.BIGINT); else
ps.setLong(11, t.winnerBidId);
            ps.setString(12, t.category);
            ps.setLong(13, t.id);
            ps.executeUpdate();
        }
    }

    public void delete(long id) throws SQLException {
        try (Connection c = DB.get();
             PreparedStatement ps = c.prepareStatement("delete from
tenders where id=?")) {
            ps.setLong(1, id);
            ps.executeUpdate();
        }
    }

    public void close(long id) throws SQLException {
        try (Connection c = DB.get();
             PreparedStatement ps = c.prepareStatement("update tenders
set status='Closed' where id=?")) {
            ps.setLong(1, id);
            ps.executeUpdate();
        }
    }

    public void award(long tenderId, long bidId, String reason) throws
SQLException {
        try (Connection c = DB.get();
             PreparedStatement ps = c.prepareStatement(
                     "update tenders set status='Awarded',
disclose_date=current_date, winner_bid_id=?, winner_reason=? where
id=?")) {
            ps.setLong(1, bidId);
            ps.setString(2, reason);
            ps.setLong(3, tenderId);
            ps.executeUpdate();
        }
    }
}
```

CompanyDao.java : Manages company information and authentication (This code snippet only shows a small part of this class):

```
package org.todo.model.company;

import org.todo.model.db.DB;

import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.sql.*;
import java.util.*;
import java.util.stream.Collectors;

public class CompanyDao {

    private Company mapCompany(ResultSet rs) throws SQLException {
       Company c = new Company();
       c.id = rs.getLong("id");
       c.companyUid = rs.getString("company_uid");
       c.name = rs.getString("name");
       c.passwordHash = rs.getString("password_hash");

       // tolerate DBs that don't have the new columns yet (first run
before Schema.ensure)
       try { c.failedAttempts = rs.getInt("failed_attempts"); } catch
(SQLException ignore) { c.failedAttempts = 0; }
       try { c.locked = rs.getBoolean("locked"); }            catch
(SQLException ignore) { c.locked = false; }
       return c;
    }

    private static String sha256(String s) {
       try {
          MessageDigest md = MessageDigest.getInstance("SHA-256");
          byte[] d = md.digest(s.getBytes(StandardCharsets.UTF_8));
          StringBuilder b = new StringBuilder(d.length * 2);
          for (byte x : d) b.append(String.format("%02x", x));
          return b.toString();
       } catch (Exception e) { throw new RuntimeException(e); }
    }

    /* ---------------- helpers for category normalization ------------
---- */

    /** Split a category name that may itself contain a packed list. */
    private static List<String> splitMaybePacked(String raw) {
       if (raw == null) return List.of();
       String s = raw.trim();
       if (s.isEmpty()) return List.of();

       // Looks like ["A","B"]
       if ((s.startsWith("[") && s.endsWith("]"))) {
          try {
             // very small permissive parser: strip [ ], split on
```

BidDao.java : Stores and retrieves bid data including attachments:

```
package org.todo.model.tender;

import org.todo.model.db.DB;

import java.math.BigDecimal;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class BidDao {

    private Bid map(ResultSet rs) throws SQLException {
       Bid b = new Bid();
       b.id = rs.getLong("id");
       b.tenderId = rs.getLong("tender_id");
       b.companyId = rs.getString("company_id");
       b.companyName = rs.getString("company_name");
       b.bidPrice = rs.getBigDecimal("bid_price");
       b.createdAt = rs.getTimestamp("created_at").toLocalDateTime();

       long aid = rs.getLong("attachment_id");
       b.attachmentId = rs.wasNull() ? null : aid;
       return b;
    }

    /** Lists bids for a tender, cheapest first, and includes
attachment_id if present. */
    public List<Bid> listFor(long tenderId) throws SQLException {
       try (Connection c = DB.get();
            PreparedStatement ps = c.prepareStatement("""
                select b.*,
                       (select id from bid_files bf where
bf.bid_id=b.id) as attachment_id
                from bids b
                where b.tender_id=?
                order by b.bid_price asc
              """)) {
          ps.setLong(1, tenderId);
          ResultSet rs = ps.executeQuery();
          List<Bid> out = new ArrayList<>();
          while (rs.next()) out.add(map(rs));
          return out;
       }
    }

    public long create(long tenderId, String companyId, String
companyName, BigDecimal price) throws SQLException {
       try (Connection c = DB.get();
            PreparedStatement ps = c.prepareStatement(
                  "insert into bids(tender_id, company_id, company_name,
bid_price) values(?,?,?,?)",
                  Statement.RETURN_GENERATED_KEYS)) {
          ps.setLong(1, tenderId);
          ps.setString(2, companyId);
          ps.setString(3, companyName);
```

```
            ps.setBigDecimal(4, price);
            ps.executeUpdate();
            ResultSet keys = ps.getGeneratedKeys();
            keys.next();
            return keys.getLong(1);
        }
    }

    public void delete(long id) throws SQLException {
        try (Connection c = DB.get();
             PreparedStatement ps = c.prepareStatement("delete from bids
where id=?")) {
            ps.setLong(1, id);
            ps.executeUpdate();
        }
    }

    /**
     * Save (or replace) the attachment for a bid.
     * Table schema guarantees a single file per bid (unique on
bid_id).
     */
    public void saveFile(long bidId, String filename, String
contentType, byte[] data) throws SQLException {
        try (Connection c = DB.get()) {
            c.setAutoCommit(false);
            try {
                // Ensure only one file per bid (unique in schema).
Replace if exists.
                try (PreparedStatement del = c.prepareStatement("delete
from bid_files where bid_id=?")) {
                    del.setLong(1, bidId);
                    del.executeUpdate();
                }
                try (PreparedStatement ins = c.prepareStatement("""
                        insert into bid_files(bid_id, filename,
content_type, data)
                        values(?,?,?,?)
                    """)) {
                    ins.setLong(1, bidId);
                    ins.setString(2, filename);
                    ins.setString(3, contentType);
                    ins.setBytes(4, data);
                    ins.executeUpdate();
                }
                c.commit();
            } catch (SQLException e) {
                c.rollback();
                throw e;
            } finally {
                c.setAutoCommit(true);
            }
        }
    }
}
```

LoginServlet.java : Handles user authentication for companies and city staff:

```
package org.todo.controller;

import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.todo.model.company.Company;
import org.todo.model.company.CompanyDao;
import org.todo.view.TemplateEngine;

import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;

@WebServlet("/login")
public class LoginServlet extends HttpServlet {

    private final CompanyDao companyDao = new CompanyDao();

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse
resp) throws IOException {
        // Already logged in as company? go home.
        if (req.getSession(false) != null &&
req.getSession(false).getAttribute("username") != null) {
            resp.sendRedirect(req.getContextPath() + "/");
            return;
        }
        // Show login page (message placeholder supported)
        if (req.getAttribute("message") == null)
req.setAttribute("message", "");
        TemplateEngine.process("login.html", req, resp);
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse
resp) throws IOException {
        String action = s(req.getParameter("action"));
        if (!"login".equalsIgnoreCase(action)) { doGet(req, resp);
return; }

        String companyId = s(req.getParameter("companyId"));
        String password  = s(req.getParameter("password"));

        if (companyId.isBlank() || password.isBlank()) {
            req.setAttribute("message", "Please enter company-id and
password.");
            TemplateEngine.process("login.html", req, resp);
            return;
        }

        try {
            Company c = companyDao.findByUid(companyId);
            // Unknown id → generic error (don't reveal)
            if (c == null) {
```

```
            req.setAttribute("message", "Invalid company-id or
password.");
            TemplateEngine.process("login.html", req, resp);
            return;
         }

         // Locked?
         if (c.locked) {
            req.setAttribute("message", "Account is locked due to too
many failed attempts. Please contact city staff.");
            TemplateEngine.process("login.html", req, resp);
            return;
         }

         boolean ok = hash(password).equalsIgnoreCase(c.passwordHash);
         if (!ok) {
            companyDao.recordFailedLogin(companyId);
            // Refresh to compute attempts left
            Company after = companyDao.findByUid(companyId);
            int left = Math.max(0, 5 - (after == null ? 0 :
after.failedAttempts));
            String msg = (after != null && after.locked)
                  ? "Account locked due to too many failed attempts."
                  : ("Invalid company-id or password." + (left > 0 ? "
Attempts left: " + left + "." : ""));
            req.setAttribute("message", msg);
            TemplateEngine.process("login.html", req, resp);
            return;
         }

         // Success → reset attempts, create session
         companyDao.resetLoginFailures(companyId);

         var ses = req.getSession(true);
         ses.setAttribute("username", c.companyUid);   // used by
BidsApiServlet
         ses.setAttribute("companyName", c.name);
         ses.setAttribute("companyDbId", c.id);
         ses.removeAttribute("staffId");
         ses.removeAttribute("staffEmail");

         resp.sendRedirect(req.getContextPath() + "/");
      } catch (Exception e) {
         req.setAttribute("message", "Login failed: " +
e.getMessage());
         TemplateEngine.process("login.html", req, resp);
      }
   }
```