VAMK University of Applied Sciences

Agile Software Development

Supervisor

Dan Duong Dang

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

TenderPoint

e-Tendering System for X City

Table of Contents

[1. Executive Summary 2](#_Toc213460409)

[2. Introduction 4](#_Toc213460410)

[3. Project Management & Agile Process 5](#_Toc213460411)

[Project Timeline 5](#_Toc213460412)

[Requirements Engineering 6](#_Toc213460413)

[Major User Requirements 6](#_Toc213460414)

[Additional User Requests 6](#_Toc213460415)

[Operational / Performance / Security Requirements 7](#_Toc213460416)

[Agile Framework 8](#_Toc213460417)

[Sprint Planning 9](#_Toc213460418)

[Backlog Documentation 10](#_Toc213460419)

[Meetings & Communication 11](#_Toc213460420)

[Kanban Board 12](#_Toc213460421)

[Definition of Done (DoD) 13](#_Toc213460422)

[4. System Design & Architecture 14](#_Toc213460423)

[System Overview 14](#_Toc213460424)

[Database Design (ERD) 15](#_Toc213460425)

[Company and Category Structure 16](#_Toc213460426)

[Tender and Bidding Structure 17](#_Toc213460427)

[Notes on Implementation 18](#_Toc213460428)

[UML Class Diagram 19](#_Toc213460429)

[Sequence Diagrams 21](#_Toc213460430)

[Browsing and Searching Tenders 21](#_Toc213460431)

[Registering, Updating, and Deleting Tenders 22](#_Toc213460432)

[Bidding and Awarding a Tender 23](#_Toc213460433)

[5. Implementation 24](#_Toc213460434)

[Development Environment 24](#_Toc213460435)

[Project Structure 24](#_Toc213460436)

[Backend Implementation 25](#_Toc213460437)

[Frontend Implementation 27](#_Toc213460438)

[6. Testing & Quality Assurance 30](#_Toc213460439)

[Functional Test Cases 30](#_Toc213460440)

[Integration Testing 33](#_Toc213460441)

[7. Results & Evaluation 34](#_Toc213460442)

[8. Conclusion & Future Work 35](#_Toc213460443)

[9. Appendix 36](#_Toc213460444)

[10. References 37](#_Toc213460445)

# 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.

# 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.

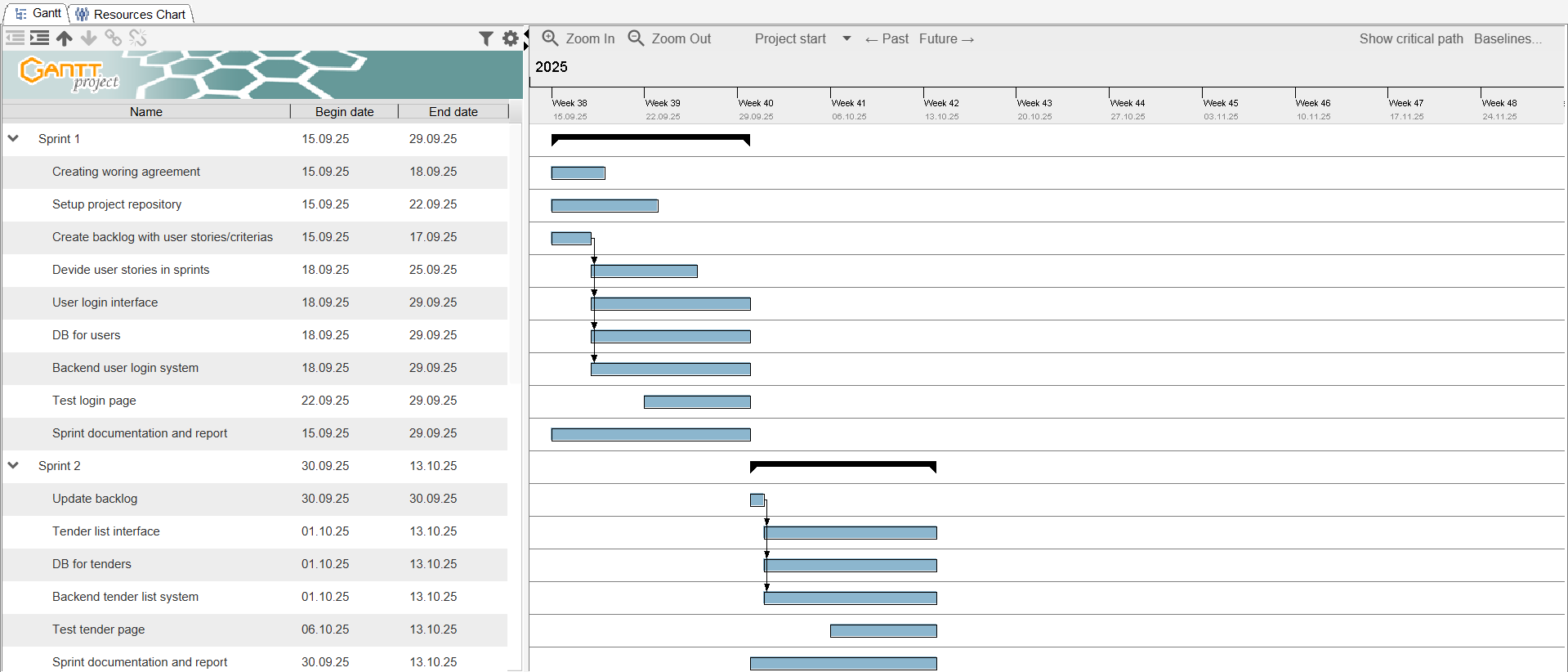
By the end of the project, TenderPoint turned into a fully functional e-tendering platform that meets the original requirements. It offers a simple and secure way to manage tenders digitally, increases transparency for citizens and makes it easier for companies to participate in city projects.

# 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.



Ein Bild, das Text, Screenshot, Zahl, Reihe enthält.

KI-generierte Inhalte können fehlerhaft sein.



Figure 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 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 |
| 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 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 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.

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

KI-generierte Inhalte können fehlerhaft sein.

Figure 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.

# 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.

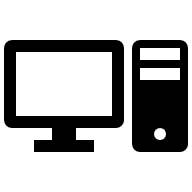
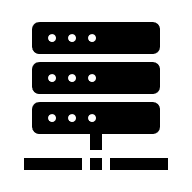


Figure Client-Server arschitecture

Request

Response

Server

Client

## 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:

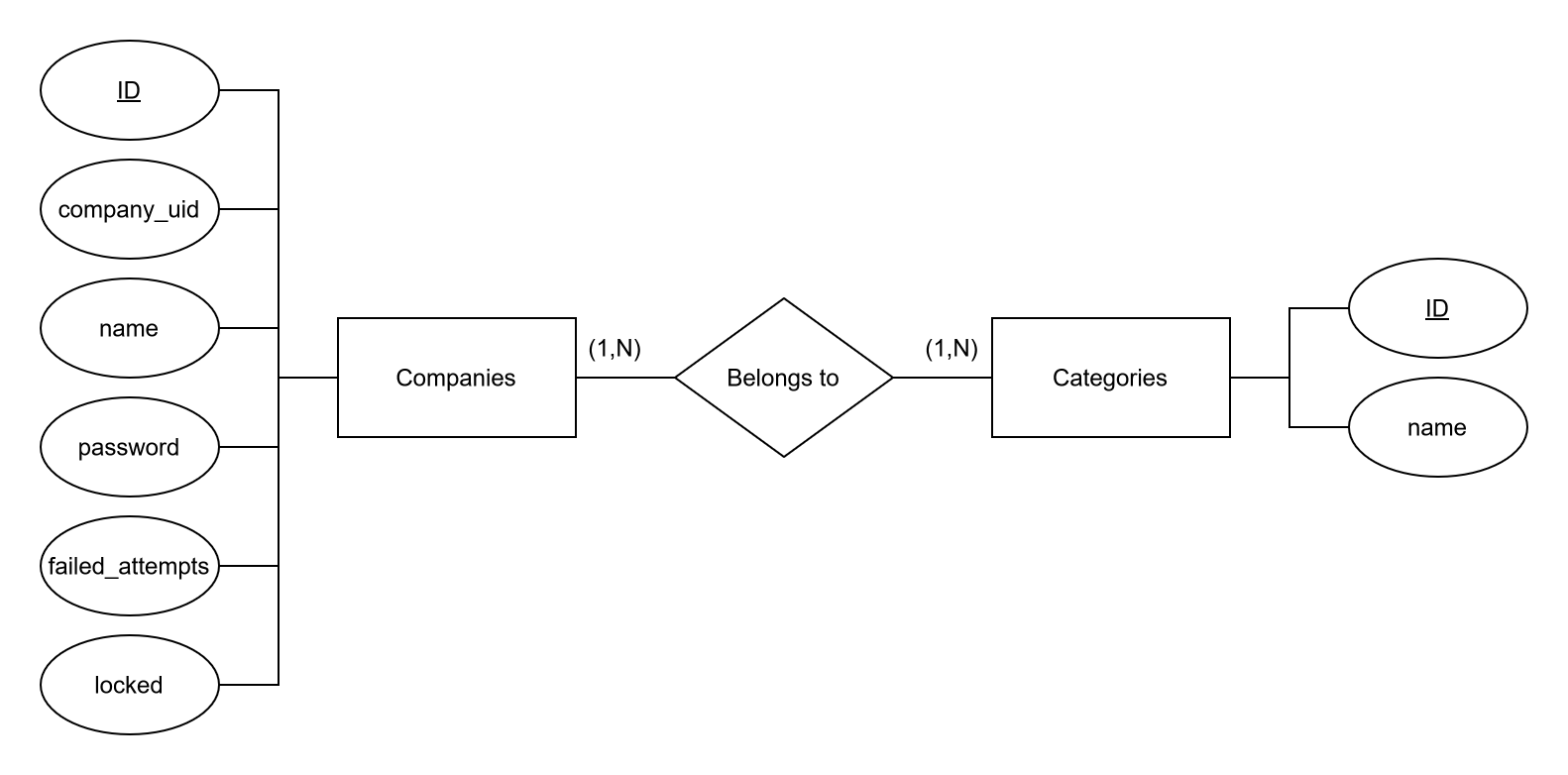


Figure Company-Category ERD

### 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:

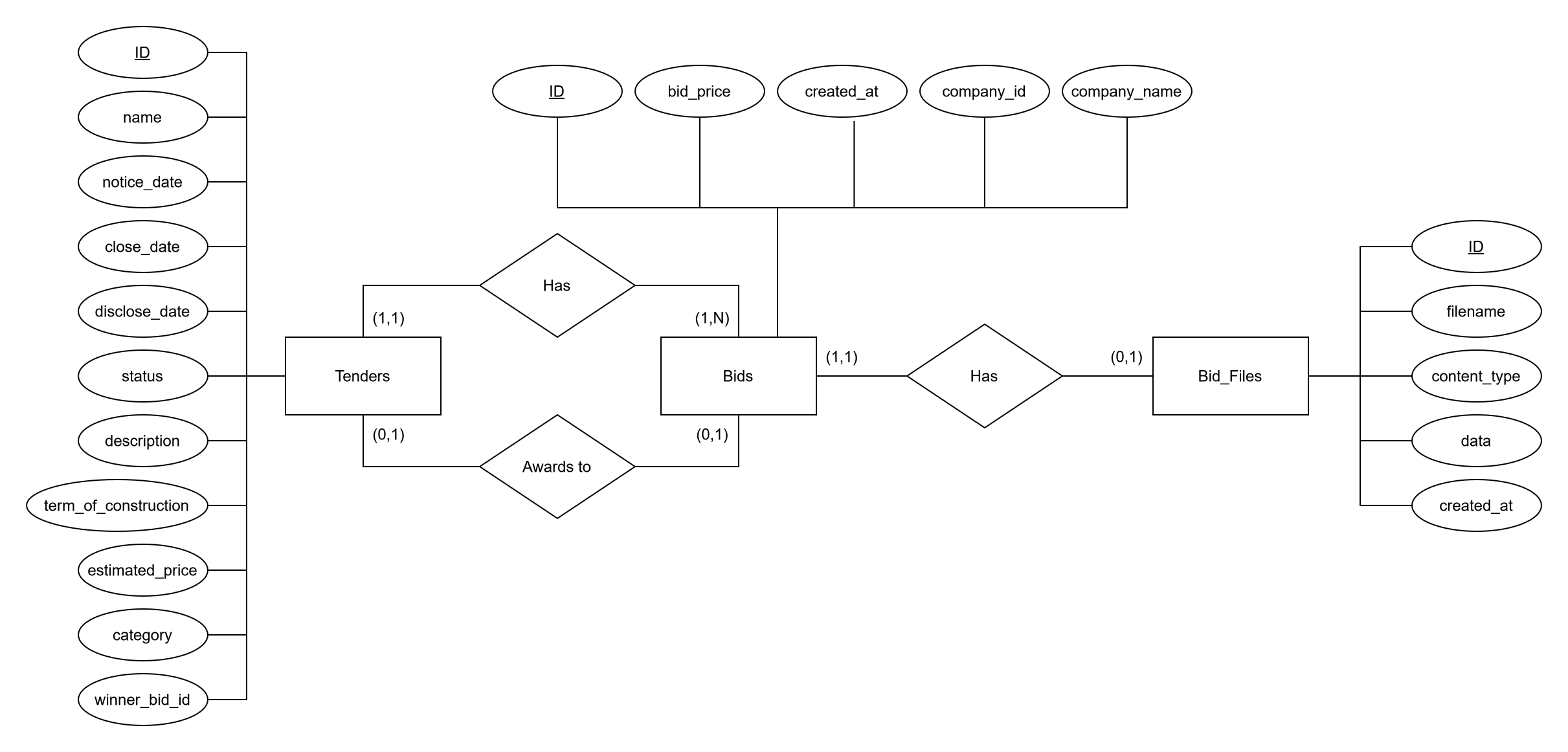


Figure Tender-Bidding ERD

### 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.

Ein Bild, das Text, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.This modular structure ensures a clear separation of concerns, improves maintainability and allows multiple developers to work on different layers simultaneously without conflicts.

Figure 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.

Ein Bild, das Text, Screenshot, Diagramm, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Figure 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.

Ein Bild, das Text, Screenshot, Diagramm, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Figure 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.

Ein Bild, das Text, Screenshot, Diagramm, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Figure Sequence Diagram - Bid and Open the Tender

# 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

Ein Bild, das Text, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.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 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.

Ein Bild, das Text, Screenshot, Schrift, Multimedia-Software enthält.

KI-generierte Inhalte können fehlerhaft sein.

Figure 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.

Ein Bild, das Text, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

Figure 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.

Ein Bild, das Text, Screenshot, Software, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Figure 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

Ein Bild, das Text, Software, Betriebssystem, Computersymbol enthält.

KI-generierte Inhalte können fehlerhaft sein.

Figure Frontend login.html

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

Ein Bild, das Text, Software, Webseite, Computersymbol enthält.

KI-generierte Inhalte können fehlerhaft sein.

Figure Frontend tender.html

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

Ein Bild, das Text, Software, Computersymbol, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein.

Figure Frontend tender-form.html

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

Ein Bild, das Text, Software, Zahl, Computersymbol enthält.

KI-generierte Inhalte können fehlerhaft sein.

Figure Frontend companies.html

* about.html: static informational pages

Ein Bild, das Text, Screenshot, Schrift, Software enthält.

KI-generierte Inhalte können fehlerhaft sein.

Figure Frontend about.html

* contact.html: static informational pages

Ein Bild, das Text, Screenshot, Software, Webseite enthält.

KI-generierte Inhalte können fehlerhaft sein.

Figure 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.

# 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 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 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 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.

Ein Bild, das Text, Schrift, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

Figure Test results with a completion of 98 integration tests

# Results & Evaluation

Present achieved functionalities, comparisons with expectations, and limitations.

# Conclusion & Future Work

Summarize achievements, lessons learned, and ideas for improvement.

# Appendix

Include meeting minutes, Kanban board snapshots, diagrams, and supporting materials.

# References

List all sources, books, and websites used.