

Vysoká škola báňská - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

Vývoj informačních systémů

MiniGitHub

Jméno: Phat Tran Dai

Osobní číslo: TRA0163

Datum: 02. 12. 2025

Content

| | |
|--|-----------|
| 1. Vision | 3 |
| 1.1. Introduction | 3 |
| 1.2. Positioning | 3 |
| 1.2.1. Problem Statement | 3 |
| 1.2.2. Product Position Statement | 3 |
| 1.3. Stakeholder and User Descriptions | 4 |
| 1.3.1. Stakeholder Summary | 4 |
| 1.3.2. User Summary | 4 |
| 1.3.3. User Environment | 4 |
| 1.4. Product Overview | 4 |
| 1.4.1. Product Perspective | 4 |
| 1.4.2. Assumptions and Dependencies | 5 |
| 1.5. Product Features | 5 |
| 1.6. Other Product Requirements | 5 |
| 2. Funkční analýza - Use Case Model | 6 |
| 2.1. Hlavní aktéři | 6 |
| 2.2. Use-case diagram | 6 |
| 2.3. Use Case 1: Vytvoření repozitáře | 7 |
| 2.4. Use Case 2: Odeslání commitů (Commit změn) | 9 |
| 2.5. Use Case 3: Vytvoření Issue | 11 |
| 3. Analýza - technické požadavky | 14 |
| 3.1. Konceptuální model domény | 14 |
| 3.1.1. Hlavní entity: | 14 |
| 3.2. Odhad velikostí entit a jejich množství | 15 |
| 3.3. Odhad počtu současně pracujících uživatelů | 15 |
| 3.4. Typy interakcí uživatelů se systémem a jejich náročnost | 16 |
| 3.5. První představa o rozložení systému a volba platforem | 16 |
| 3.5.1. Architektura: | 16 |
| 3.5.2. Platformy: | 16 |
| 4. Skica | 17 |
| 4.1. Sign In/Sign Up/Sign Out | 17 |
| 4.2. Dashboard | 17 |
| 4.3. Search | 18 |
| 4.4. User Profile View | 18 |
| 4.5. Repository View | 18 |
| 4.6. Issue View | 19 |
| 4.7. Commit View | 19 |
| 4.8. Create Commit | 19 |
| 5. Domain Model | 20 |
| 5.1. Entities | 20 |
| 5.2. Mappers | 20 |
| 5.3. Repositories / Services | 20 |

| | |
|------------------------------|-----------|
| 5.4. Class Diagram | 21 |
| 5.5. Sequence Diagram | 22 |
| 6. Architecture | 23 |

1. Vision

1.1. Introduction

The purpose of this document is to define the high-level needs and features of MiniGitHub, a web-based code collaboration platform. MiniGitHub is designed to help developers and teams manage code repositories, track changes, and collaborate effectively on software projects.

This document provides an overview of the product's purpose, positioning in the market, stakeholders and users, and its key features. It will serve as a guide for aligning development efforts and ensuring the solution meets user expectations.

1.2. Positioning

1.2.1. Problem Statement

| | |
|--------------------------------|--|
| The problem of | fragmented and inefficient collaboration on code projects |
| affects | developers/student teams, open-source contributors, and organizations that rely on version control and collaboration tools |
| the impact of which is | code conflicts, slower development cycles, miscommunication, lack of version tracking, and difficulties in managing distributed teams |
| a successful solution would be | a simple and accessible platform that enables users to store, manage, and share code repositories, track changes through versioning, and collaborate effectively with features such as issue tracking and pull requests. |

Table 1: Problem statement

1.2.2. Product Position Statement

| | |
|-----------------------|---|
| For | developers, student teams, and organizations seeking a lightweight but structured collaboration platform |
| Who | need a reliable and accessible version control and project collaboration tool |
| The MiniGitHub system | is a repository hosting and collaboration platform |
| That | provides version control, issue tracking, and lightweight collaboration features in a user-friendly environment |
| Unlike | existing heavyweight platforms such as GitHub or GitLab, which may be overly complex for small teams |
| Our product | offers a streamlined, focused solution with only the essential collaboration tools. |

Table 2: Product Position Statement

1.3. Stakeholder and User Descriptions

1.3.1. Stakeholder Summary

| Name | Description | Responsibilities |
|----------------------|---|--|
| Project Sponsor | Provides funding or backing for development | Defines goals and approves overall direction |
| Development Team | Engineers building the system | Implements features, ensures maintainability |
| System Administrator | Manages hosting and deployment | Keeps system online and secure |

Table 3: Stakeholder Summary

1.3.2. User Summary

| Name | Descriptions | Responsibilities | Stakeholder |
|-------------|--|---|------------------|
| Developer | Individual writing and committing code | Creates repositories, commits changes, opens issues | Development Team |
| Maintainer | Oversees project repositories | Reviews contributions, manages issues and pull requests | Development Team |
| Contributor | External collaborator | Suggests changes, reports bugs, submits pull requests | Development Team |

Table 4: User Summary

1.3.3. User Environment

| | |
|-----------------------------|---|
| Number of users per project | Typically 2–10, may scale for open collaboration. |
| Task cycle | Continuous; developers frequently commit, push, and review code. |
| Environment constraints | Primarily desktop/laptop through a web browser; later expansion to mobile is possible. |
| Platforms in use | Modern web browsers for front-end, PostgreSQL/SQL Server for data, ASP.NET/Python-based backend. WPF desktop application for Windows. |

1.4. Product Overview

1.4.1. Product Perspective

MiniGitHub is an independent system designed as a lightweight GitHub alternative. It consists of three major layers:

- Presentation Layer: Web interface and desktop application for interaction.
- Domain Layer: Business logic for repository, commit, and issue management.
- Data Layer: Database with persistence services.

Interfaces:

- Desktop application (for end users)
- Web browser (for end users)
- REST API (for integration with Git clients)

1.4.2. Assumptions and Dependencies

System assumes internet-connected users with modern web browsers.

Database system (SQL Server or PostgreSQL) must be available.

Hosting environment supports .NET (if on Windows) or Python (if on Linux) runtime.

Full Git implementation is out of scope; repository management will be simplified.

1.5. Product Features

| | |
|--------------------------|---|
| User management | Registration, login, authentication. |
| Repository management | Create, list, delete repositories. |
| Commits | Track file version and changes. |
| Issue | Open, comment on, and close issues. |
| Pull requests (optional) | Propose and merge proposed changes. |
| Search and browse | Find repositories and issues. |
| Collaboration tools | Comments, activity feeds. |
| Basic analytics | Number of commits, contributors, and issues per repo. |

1.6. Other Product Requirements

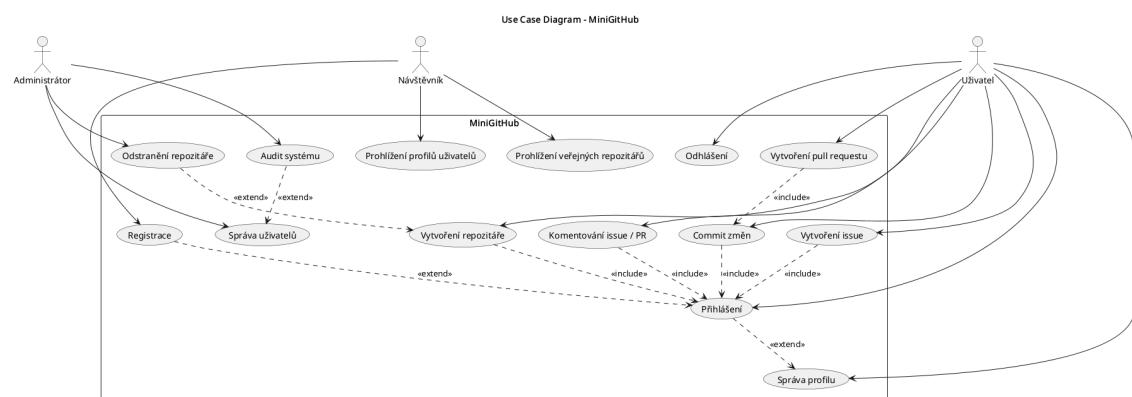
| | |
|-----------------------|--|
| Standards | Follows MVC architecture for the web front end, layered design (Data Access, Domain, Presentation), and selected patterns (Domain Model, Data Mapper, Unit of Work). |
| Performance | Must support 50–100 concurrent users with acceptable response time. |
| Robustness | Transactions must ensure consistent repository state. |
| Platform requirements | Runs on Linux/Windows server, database backend required. |
| Usability | Clean and intuitive UI, minimal learning curve. |
| Documentation | Basic user guide and technical developer documentation. |
| Constraints | Project scope limited to core GitHub-like features (full Git replication is out of scope). Only core repository and collaboration features are included. |

2. Funkční analýza - Use Case Model

2.1. Hlavní aktéři

| Aktér | Popis |
|---------------|--|
| Uživatel | Registrovaný uživatel systému, který může vytvářet repozitáře, provádět commity, spravovat issues a pull requesty. |
| Návštěvník | Nepřihlášený uživatel, který si může prohlížet veřejné repozitáře. |
| Administrátor | Správce systému, který dohlíží na chod aplikace, spravuje uživatele a řeší incidenty. |

2.2. Use-case diagram



2.3. Use Case 1: Vytvoření repozitáře

- **Název a identifikace:** Vytvoření repozitáře
- **Aktér:** Uživatel
- **Cíl:** Založit nový repozitář, ve kterém může uživatel ukládat a spravovat zdrojové kódy.
- **Prekondice:**
 - Uživatel je přihlášený do systému.
 - Název repozitáře není v účtu uživatele již použit.
- **Spouštěcí událost:** Uživatel klikne na tlačítko „Nový repozitář“.
- **Hlavní scénář:**
 - Systém zobrazí formulář pro vytvoření repozitáře.
 - Uživatel zadá název, volitelný popis a nastaví viditelnost (veřejný/soukromý).
 - Uživatel potvrdí vytvoření.
 - Systém ověří, že zadané údaje jsou platné.
 - Systém vytvoří nový repozitář, včetně počáteční větve (např. main).
 - Systém zobrazí stránku nově vytvořeného repozitáře.
- **Alternativní scénář:** Název je neplatný nebo již existuje - systém zobrazí chybové hlášení a umožní opravu.
- **Postkondice:**
 - V databázi vznikne nový záznam o repozitáři.
 - Uživatel se stává vlastníkem repozitáře.

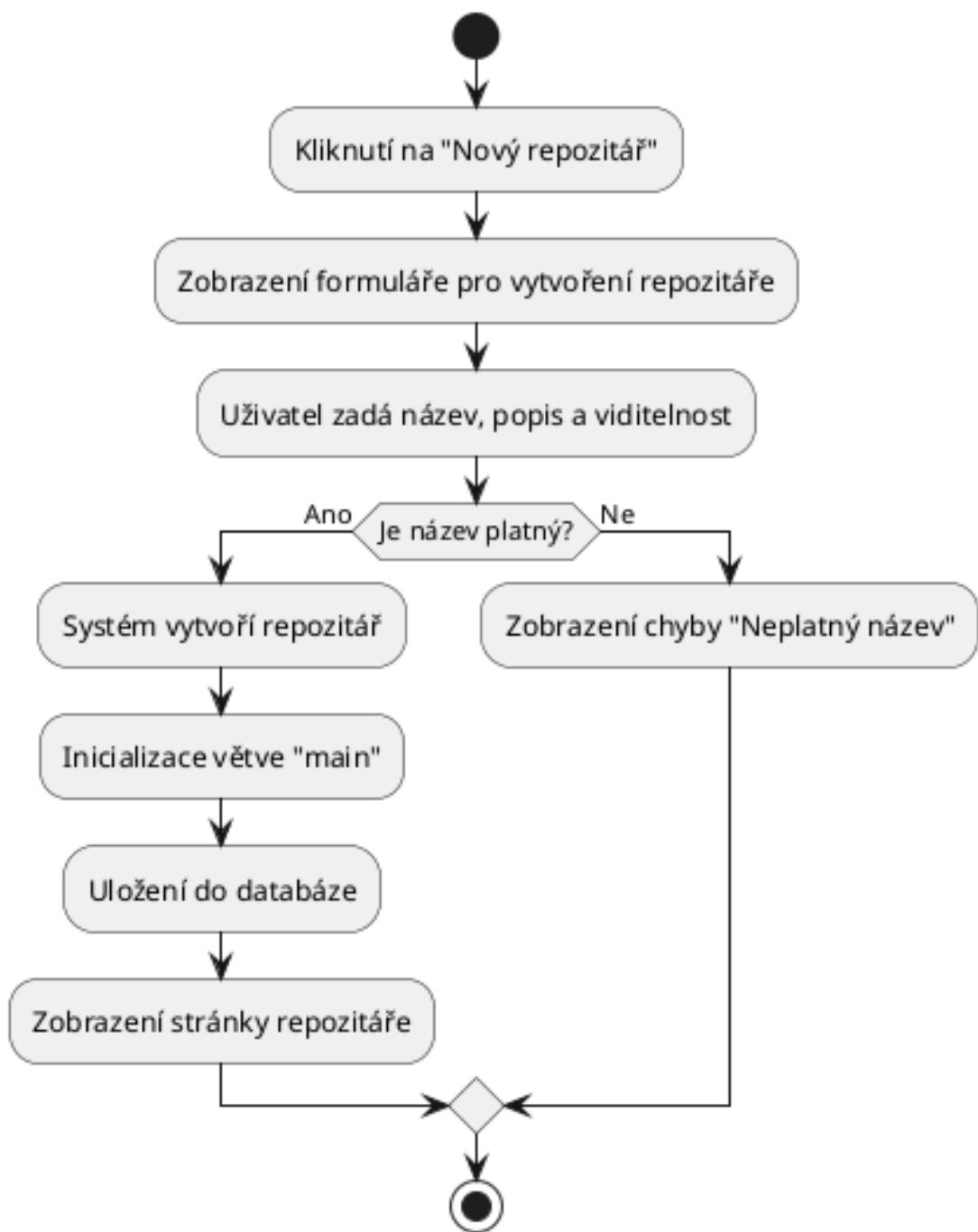


Figure 1: Use-case diagram - Vytvoření repozitáře

2.4. Use Case 2: Odeslání commitů (Commit změn)

- **Název a identifikace:** Odeslání commitů do repozitáře
- **Aktér:** Uživatel
- **Cíl:** Uložit změny ve zdrojových souborech do historie verzí repozitáře.
- **Prekondice:**
 - Uživatel má právo zapisovat do repozitáře.
 - Repozitář existuje a obsahuje alespoň jednu větev.
- **Spouštěcí událost:** Uživatel provede změny v lokální kopii repozitáře a chce je uložit.
- **Hlavní scénář:**
 - Uživatel vybere soubory, které chce commitnout.
 - Zadá popis změny (commit message).
 - Potvrdí odeslání.
 - Systém ověří přístupová práva uživatele.
 - Systém vytvoří nový commit a propojí ho s předchozím.
 - Systém aktualizuje historii verzí dané větve.
 - Systém zobrazí potvrzení o úspěšném commitu.
- **Alternativní scénáře:**
 - Uživatel nemá oprávnění k zápisu → systém zobrazí chybu „Access denied“.
 - Došlo ke konfliktu s aktuální verzí → systém požádá o vyřešení konfliktu.
- **Postkondice:**
 - Nový commit je uložen v databázi.
 - Historie repozitáře je rozšířena o nový stav.

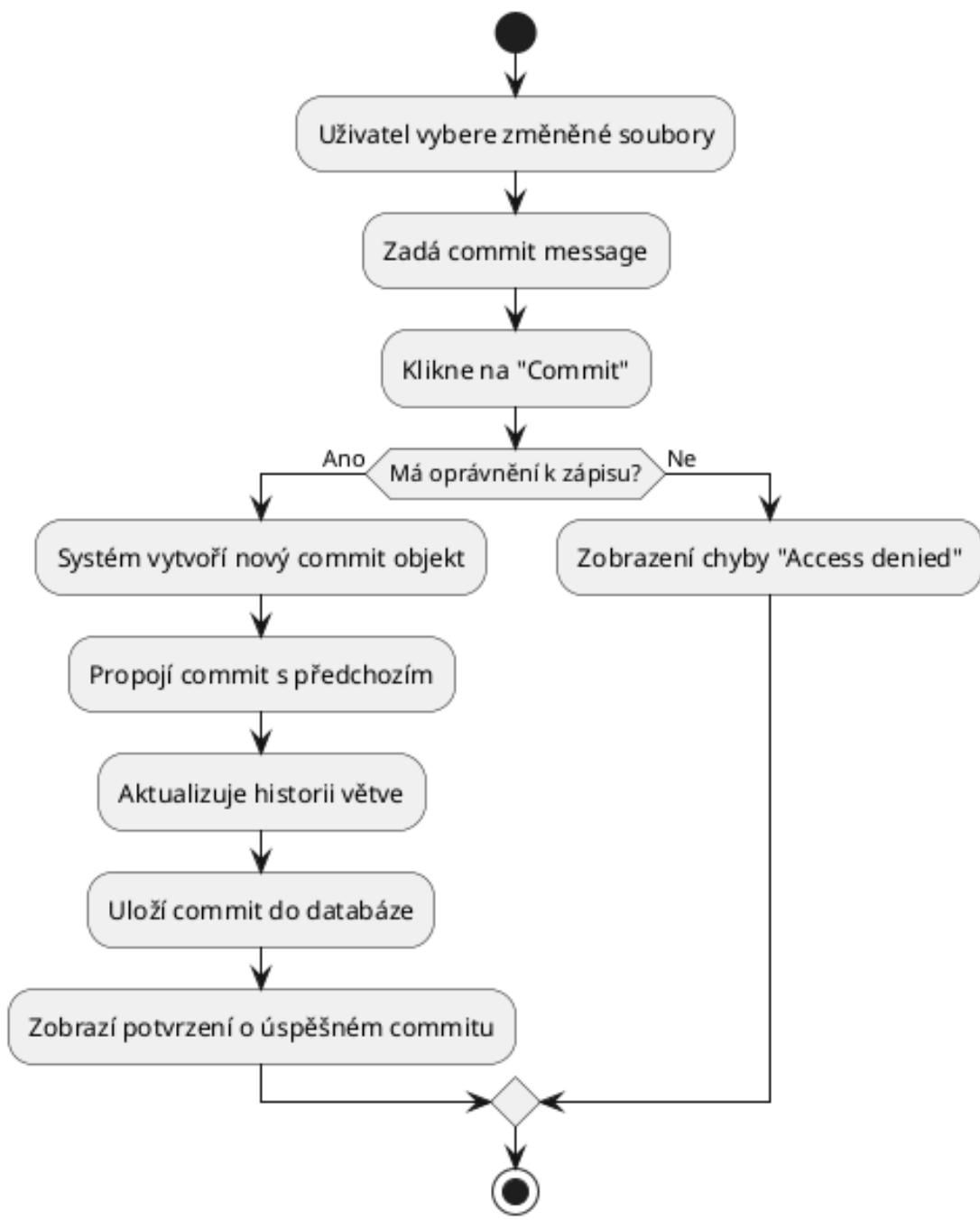


Figure 2: Use-case diagram - Odeslání commitů do repozitáře

2.5. Use Case 3: Vytvoření Issue

- **Název a identifikace:** Vytvoření issue
- **Aktér:** Uživatel (Contributor nebo Maintainer)
- **Cíl:** Nahlásit problém, chybu nebo návrh na vylepšení v repozitáři, aby mohl být sledován a vyřešen.
- **Prekondice:**
 - Repozitář existuje.
 - Uživatel má oprávnění vytvářet issue v daném repozitáři.
- **Spouštěcí událost:** Uživatel zjistí chybu nebo má návrh na vylepšení a klikne na „New Issue“.
- **Hlavní scénář:**
 - Uživatel otevře stránku repozitáře.
 - Klikne na „New Issue“.
 - Systém zobrazí formulář pro vytvoření issue.
 - Uživatel vyplní název, popis a případně přidá štítky (labels) nebo přiřadí řešitele (assignee).
 - Uživatel potvrdí vytvoření issue.
 - Systém ověří vstupy (např. nepovolené znaky, délku názvu).
 - Systém uloží issue do databáze s výchozím stavem Open.
 - Systém zobrazí nově vytvořenou issue v seznamu.
- **Alternativní scénáře:**
 - Zadání obsahuje neplatné nebo prázdné hodnoty → systém zobrazí chybové hlášení a umožní opravu.
 - Došlo k chybě při zápisu do databáze → systém zobrazí chybové hlášení „Nepodařilo se uložit issue“.
- **Postkondice:**
 - Nové issue je viditelné v seznamu všech issues.
 - Uživatelé mohou komentovat nebo měnit její stav (např. Closed).

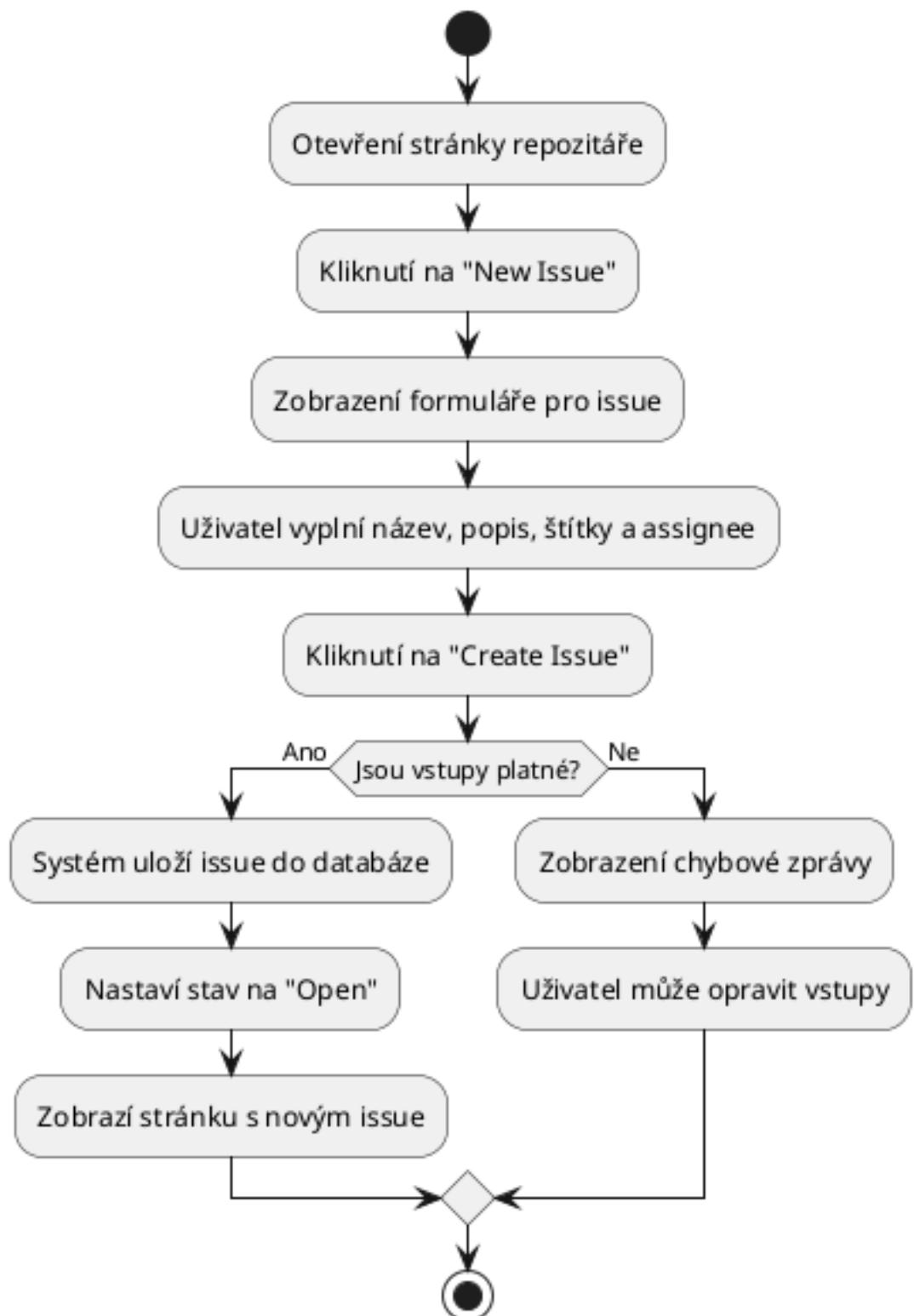


Figure 3: Use-case diagram - Vytvoření issue

3. Analýza - technické požadavky

3.1. Konceptuální model domény

3.1.1. Hlavní entity:

- **User** - reprezentuje uživatele systému (vlastní účet, přihlašovací údaje, role).
- **Repository** - projektový prostor obsahující commity a issue.
- **Commit** - záznam o změně zdrojového kódu, propojený s autorem.
- **Issue** - položka pro sledování problémů, chyb nebo návrhů.
- **Comment** - komentář k issue nebo commitu.
- **File** - soubor uložený v repozitáři, s historií verzí.

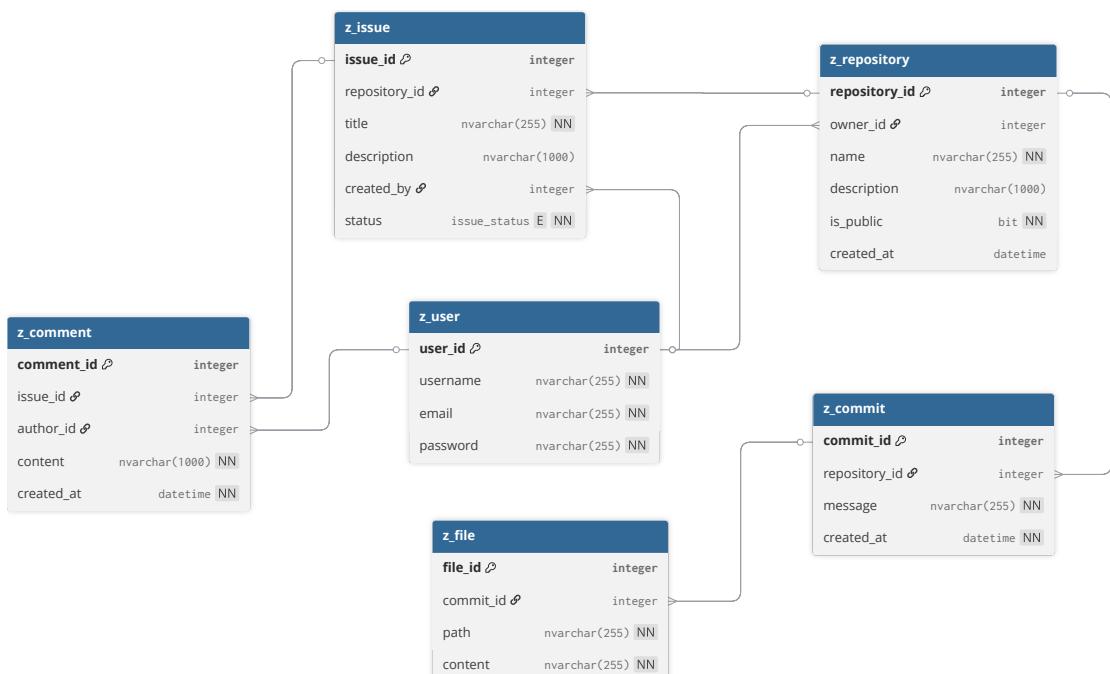


Figure 4: Konceptuální model (je to entitní model)

3.2. Odhad velikostí entit a jejich množství

| Entita | Odhadovaný počet záznamů | Velikost jednoho záznamu | Poznámka |
|------------|--------------------------|--------------------------|------------------------------------|
| User | 5000 | ~2 KB | základní profil + hash hesla |
| Repository | 15000 | ~3 KB | metadata + nastavení přístupu |
| Commit | 500000 | 5 KB | hash, message, reference na předka |
| Issue | 50000 | 3 KB | popis problému, status, priorita |
| Comment | 200000 | 1 KB | text, autor, vazba na issue/commit |
| File | 1000000 | 4 KB | metadata, nikoliv binární obsah |

- **Celkový odhad datové velikosti:** 6-8 GB v produkčním provozu (včetně indexů a metadat).

3.3. Odhad počtu současně pracujících uživatelů

- Systém cílí na menší vývojové týmy a jednotlivce, proto:
 - Běžná zátěž 50-100 současně aktivních uživatelů
 - Špičkové vytížení až 300 uživatelů (např. open-source komunita)
 - Neaktivní účty: až 10x více než aktivních uživatelů
- Vzhledem k asynchronní povaze práce s repozitářem (většina operací je krátká) je tento rozsah realistický pro běžný serverový deployment.

3.4. Typy interakcí uživatelů se systémem a jejich náročnost

| Typ interakce | Popis | Odhad náročnosti | Frekvence |
|-----------------------------|---|------------------|------------|
| Zobrazení repozitáře | čtení metadat, seznam commitů a větví | nízká | častá |
| Commit změn | zápis nové verze do DB, kontrola konfliktů | střední | střední |
| Vytvoření issue | zápis záznamu, notifikace | nízká | střední |
| Komentování | zápis textu, aktualizace | nízká | častá |
| Vytvoření repozitáře | kontrola duplicity, inicializace DB záznamů | střední | méně častá |
| Prohlížení historie commitů | agregace dat, načtení diffů | vysoká | střední |
| Vyhledávání | fulltext v repozitářích a issue | střední | častá |

Table 6: Typy interakcí uživatelů

3.5. První představa o rozložení systému a volba platforem

3.5.1. Architektura:

- Vícevrstvá architektura s oddělením zodpovědností:
 - ▶ Data Layer:
 - Implementace pomocí SQLite (vývoj) nebo PostgreSQL (produkce).
 - Využití patternů: Data Mapper, Identity Map, Unit of Work.
 - ▶ Domain Layer:
 - Logika systému (repositorie, commity, issue) v čistém C#.
 - Pattern: Domain Model + Transaction Script.
 - ▶ Presentation Layer:
 - WebUI: ASP.NET Core MVC
 - Desktop: Avalonia UI

3.5.2. Platformy:

| Vrstva | Technologie | Platforma |
|---------|-----------------------|-----------------|
| Data | SQLite / PostgreSQL | Linux / Docker |
| Domain | .NET C# Class Library | cross-platform |
| Web UI | ASP.NET Core MVC | Browser |
| Desktop | UI Avalonia | Linux / Windows |

4. Skica

4.1. Sign In/Sign Up/Sign Out

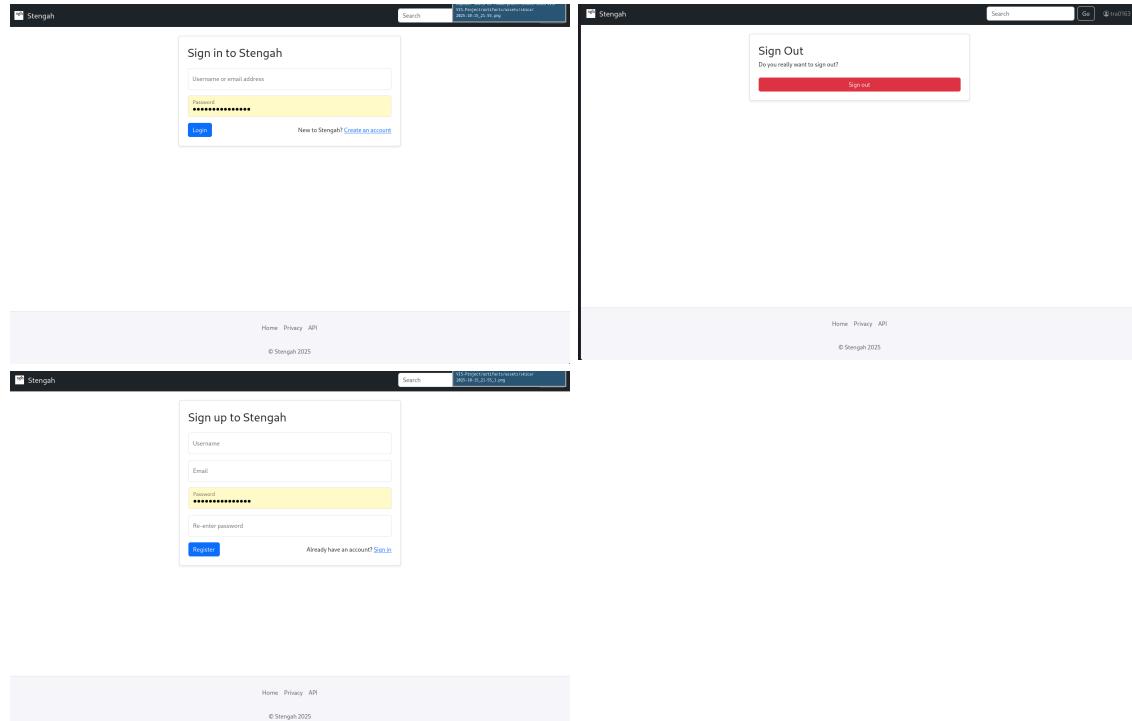


Figure 5: Skica - Sign In/Sign Up/Sign Out

4.2. Dashboard

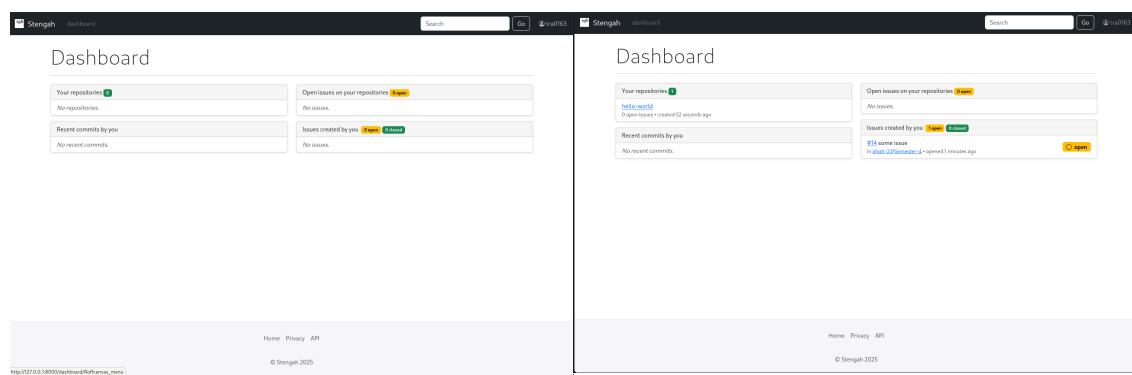


Figure 6: Skica - Dashboard

4.3. Search

The figure consists of two side-by-side screenshots of the Skica search interface. Both screenshots show a search bar at the top with the placeholder "Search for users and repositories". Below the search bar are sections for "Users" and "Repositories". In the "Users" section, it says "No users were found.". In the "Repositories" section, there are two items: "Semester-4" and "fdff". The "Semester-4" item has a subtitle "phatt-23" and "All semester 4 stuff". The "fdff" item has a subtitle "phatt-23" and "stjks". At the bottom of each screenshot, there are links for "Home", "Privacy", and "API", and a copyright notice "© Stengah 2025".

Figure 7: Skica - Search

4.4. User Profile View

The figure consists of two side-by-side screenshots of the Skica user profile view for the user "tra0163". The left screenshot shows the user's profile page. It features a placeholder profile picture, a bio placeholder ("This user hasn't provided a bio."), and a repository list containing "hello-world". The right screenshot shows the "Update Profile" form. It includes fields for "Choose Avatar" (with a placeholder "Browse... No file selected. Photo or PDF. Max size: 2MB"), "Email" (with the value "tra0163@vub.cz"), and "Bio" (with the placeholder "Tell others something about yourself"). At the bottom are "Save Changes" and "Cancel" buttons. Both screenshots include links for "Home", "Privacy", and "API", and a copyright notice "© Stengah 2025".

Figure 8: Skica - User Profile

4.5. Repository View

The figure consists of three screenshots of the Skica repository view for the repository "Semester-4". The left screenshot shows a list of files: "SourceMainWindow.cpp", "SourceMinibar.cpp", "SourceSideBar.cpp", "SourceMenuBar.cpp", "SourceMenuBar.h", "SourceMainWindow.h", "SourceMinibar.h", "SourceEmailListView.cpp", "SourceEmailListView.h", "SourceEmailListFrame.cpp", "SourceEmailListFrame.h", "SourceEmailCard.cpp", "SourceEmailCard.h", "SourceEmailListBar.cpp", "SourceEmailListBar.h", "SourceEmailComponent.cpp", "SourceEmailListFrameView.cpp", "SourceEmailListFrameView.h", "SourceEmailCardView.cpp", "SourceEmailCardView.h", and "SourceEmailRegexp.cpp". The middle screenshot shows the content of the "SourceMainWindow.cpp" file, which includes code for creating a window and initializing children. The right screenshot shows two forms: "Edit repository" (with fields for repository name "hello-world" and description "Bla bla bla") and "Create a new repository" (with fields for repository name "hello-world" and description "Bla bla bla", and checkboxes for "Public" and "Private"). Both screenshots include links for "Home", "Privacy", and "API", and a copyright notice "© Stengah 2025".

Figure 9: Skica - Repository

4.6. Issue View

The screenshot shows the Skica interface for viewing issues. On the left, there is a list of issues with a selected one, '#14 some issue'. This issue has a single comment from 'trah03' and a body containing 'bla bla bla'. Below this is a comment from 'trah03' saying 'hello, world'. A modal window titled 'Open a new issue for phatt-23/Semester-4' is open, prompting for a title and description, with 'Create issue' and 'Cancel' buttons.

Figure 10: Skica - Issue

4.7. Commit View

The screenshot shows the Skica interface for committing changes. It displays two commits for the 'hello-world' repository. The first commit, 'Commit for hello-world', was committed on Oct. 15, 2025, at 7:52 p.m. with the message 'Initial'. It contains three files: 594.py (0.45 KB), 10189.py (1.20 KB), and 10196.py (2.79 KB). The second commit, 'Commit for hello-world', was also committed on Oct. 15, 2025, at 7:52 p.m. with the message 'Initial'. It contains the same three files. Both commits have a file size of 0.45 KB and a type of 'diff'.

Figure 11: Skica - Commit

4.8. Create Commit

The screenshot shows the Skica interface for creating a new commit. It displays two side-by-side forms for creating a commit for the 'hello-world' repository. Both forms include fields for 'Commit message' (describing changes) and 'Select files' (with options to 'Browse...' or 'Select a folder'). The right-hand form shows a expanded section for 'Hide staged files (3)' containing the files 594.py, 10189.py, and 10196.py, each with 'Preview' and 'Remove' buttons. The bottom part of the right-hand form shows the commit message and file selection fields again. The bottom of both forms has 'Commit changes' and 'Cancel' buttons.

Figure 12: Skica - Create Commit

5. Domain Model

There are 3 main groups of classes:

- Entities
- Mappers
- Repositories

5.1. Entities

Entities represent the model classes in the domain layer. They are mostly the same with the models in the data layer, but they also include some other methods and have compound attributes. For example a Commit model also contains Files.

5.2. Mappers

They provide seamless mapping between models

- from the data layer to the domain layer
- from the domain layer to the data layer

They are used for abstracting the models internal workings. The client is only ever exposed to the model from the domain layer when working in the domain layer.

The opposite is true. The client is only exposed to the data models when working in the data layer.

5.3. Repositories / Services

These classes provide actions that we can do basic CRUD operations upon our models. These CRUD operations are translated into database calls, that are not necessarily only trivial operations. Some actions that these services provide deal with DB transactions and insert multiple records at once. Again the client doesn't have to worry about the inner workings of the DB, they just call upon the action and it either succeeds or it fails and throws an error.

5.4. Class Diagram

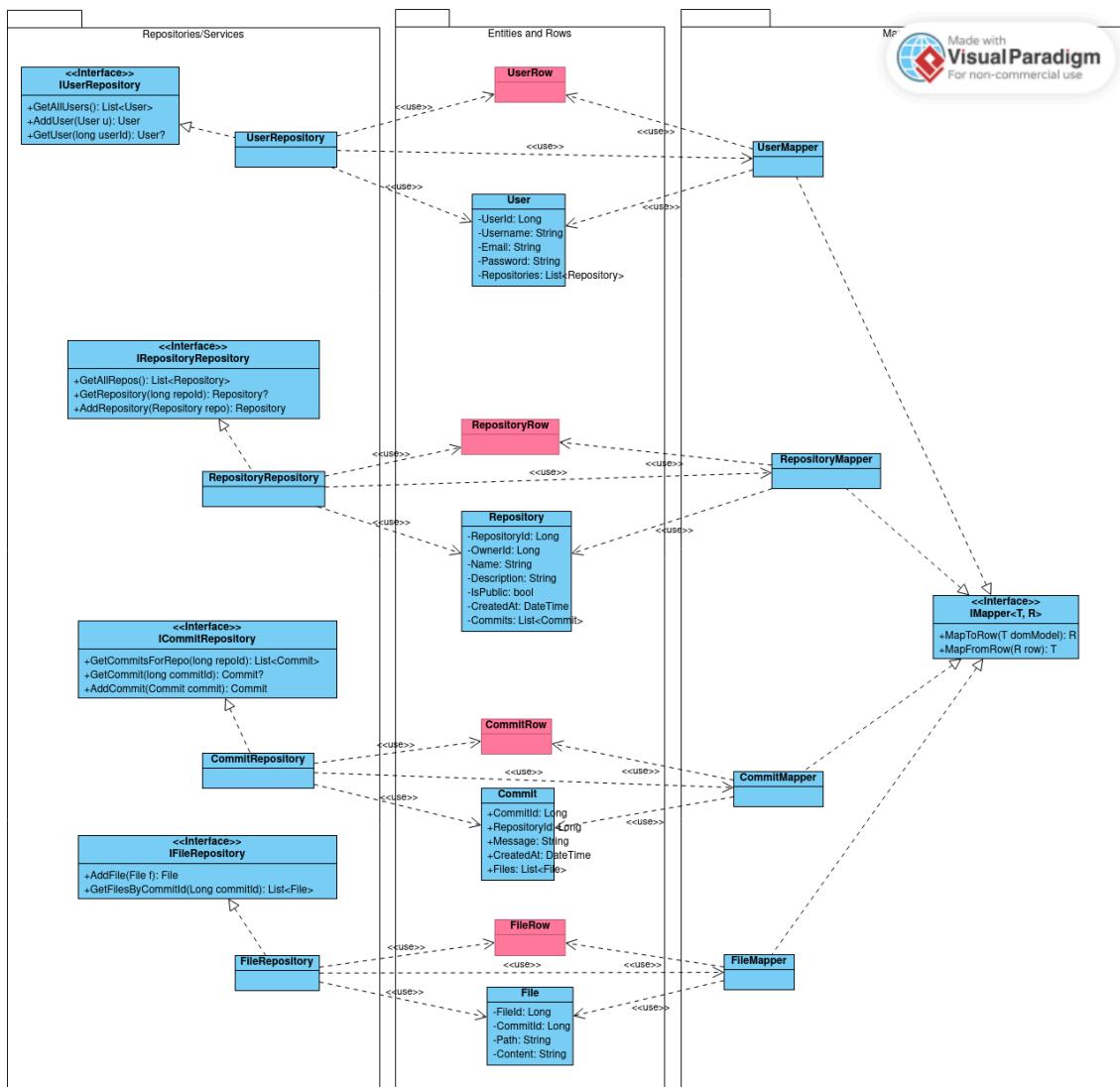


Figure 13: Domain Model Class Diagram

The classes filled red belong to the data layer.

5.5. Sequence Diagram

Here's a sequence diagram for adding a commit.

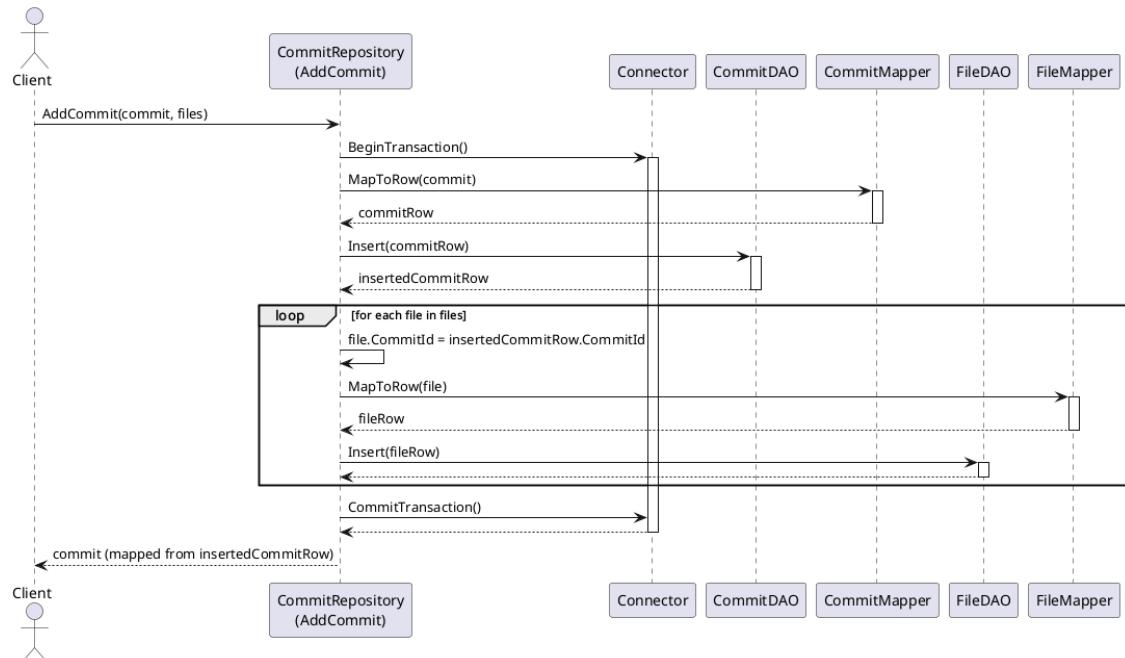


Figure 14: Sequence Diagram - Add Commit

```

Commit CommitRepository::AddCommit(Commit commit, List<File> files) {
    try {
        _connector.BeginTransaction();

        var insertedRow = _commitDao.Insert(_commitMapper.MapToRow(commit));
        foreach (var file in files)
        {
            file.CommitId = insertedRow.CommitId;
            _fileDao.Insert(_fileMapper.MapToRow(file));
        }

        _connector.CommitTransaction();
        return _commitMapper.MapFromRow(insertedRow);
    }
    catch (Exception e) {
        _connector.RollbackTransaction();
        throw;
    }
}
  
```

Listing 1: Code Implementation - Add Commit

6. Architecture

This project follows a very standard architecture. It has 3 main components - Data, Domain and Presentation layers.

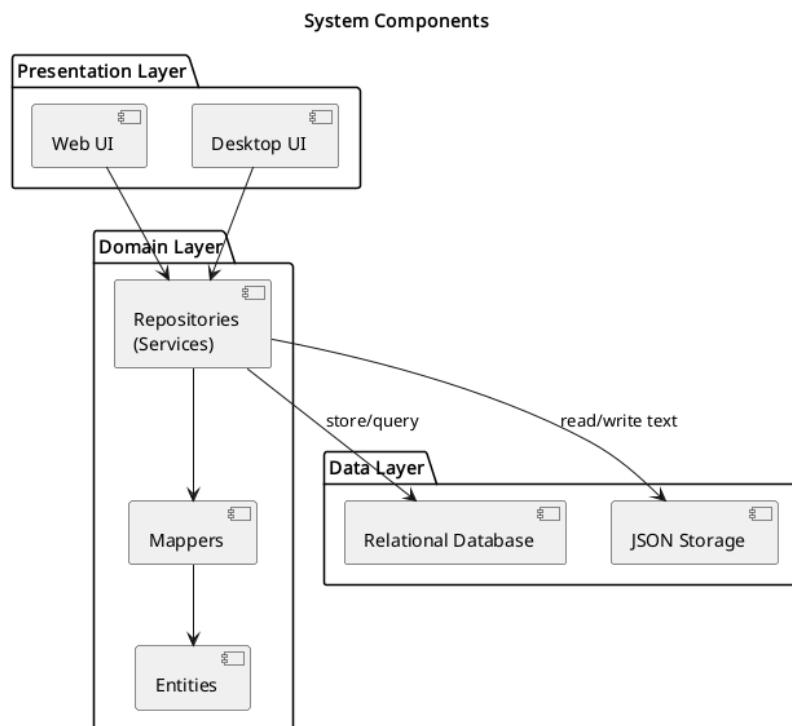


Figure 15: Component Diagram

It will be deployed on some server running the instance of the application. Another server will host the databases. Client devices are connected to the server running the application.

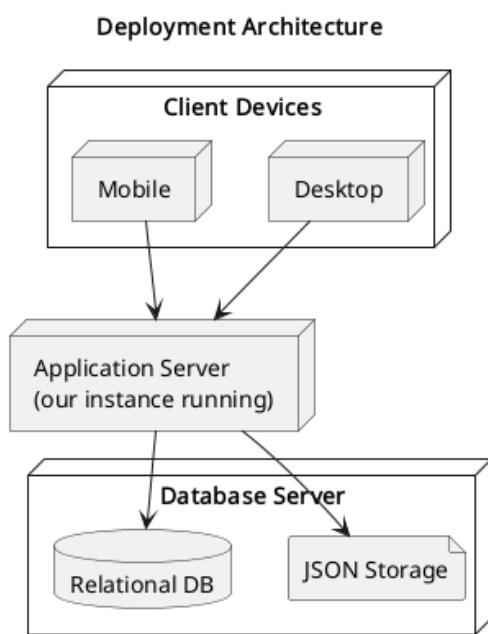


Figure 16: Deployment Diagram