**Design**

Contents

[Prototype methodology 1](#_Toc166173248)

[Platform selection 1](#_Toc166173249)

[Programming approach 2](#_Toc166173250)

[Diagrams 3](#_Toc166173251)

[Structural Diagrams 3](#_Toc166173252)

[Class Diagram 3](#_Toc166173253)

[Entity relationships 4](#_Toc166173254)

[Package diagram 5](#_Toc166173255)

[Behavioural Diagrams 6](#_Toc166173256)

[Use case diagram 6](#_Toc166173257)

[Ticket creation - Object diagram 7](#_Toc166173258)

[Ticket creation - Activity diagram 8](#_Toc166173259)

[Ticket management - Activity diagram 9](#_Toc166173260)

# Prototype methodology

Rapid prototyping will be used to allow for efficient iterative development in the early stages of the project. I feel the importance of this methodology is to ensure that the design and approach taken meets the needs and expectations of user. Following user centred design, I will be able to better meet criteria leveraging an AGILE framework to provide continuous feedback.

By breaking development down into small, manageable iterations I feel I will be able to better prioritise important features for the user. As well as identify and address issues early in the development process, resulting in a higher quality product at the end of development.

# Platform selection

I made the decision to develop the bug tracking app as a web application using .NET Core MVC within Visual studio 2022. This was due to factors such as compatibility, scalability, support, as well as time and cost effectiveness.

**Compatibility:** As a web application, the bug tracker can be used across various operating systems including Windows, mac, and Linux. This Is important as it allows for flexibility and accessibility as the app can be deployed in a wide range of environments. Regarding the game development studio, this means that aren’t limited in the hardware they chose.

**Scalability:** .NET Core MVC is highly scalable. This allows the app to handle increasing numbers of users, projects, and data without sacrificing performance. This ensures that the app can grow with the needs of the studio.

**Support:** For development I have access to a lot of resources and libraries which allows me to develop the application more effectively in a cost effective and timely manner. As I will encounter issues, I feel more confident that I have the resources available to mitigate the effects these issues might have on development.

**Time efficiency:** tools and features such as scaffolding, built in authentication and authorization, as week as integration with front-end frameworks like Bootstrap or React, help to reduce development time whilst also providing reliability in security and consistency in front end design. This allows me to better focus my attention on more important elements of the project, increasing the products quality.

**Cost effectiveness:** .NET Core MVC is open source meaning there are no fees to development. Hosting the application, however, will come with a cost but as I am able to integrate the application with Azure using Visual Studio this may be more cost effective than hosting with Digital Ocean for example.

# Programming approach

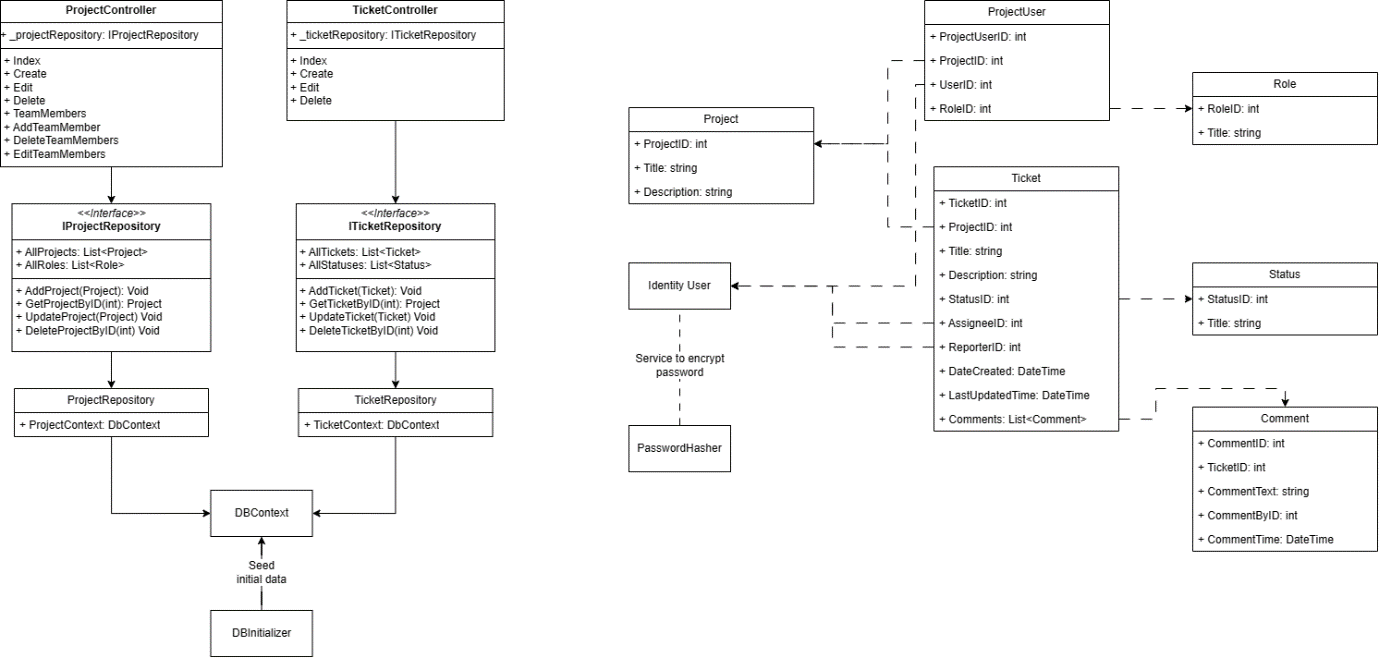
I intend on making use of Test Driven Development (TDD) to ensure any issues are promptly resolved throughout development. As the project grows issues that go unnoticed could cause lots of issues towards the end of development which may result in the deadline not being reached. This would have a negative effect on cost. The product is needed to streamline the studios game development projects and so without it, the time in which they spend developing may also increase costing the studio money.

Practices such a complying with coding standards and version control should be used to allow for better maintenance and refactoring. Maintenance is crucial for long term success and so complying with coding standards and the use of TDD allows for a maintainable, testable, and readable codebase that developers can understand. Refactoring can be seen as investment as it reduces the risk of code becoming outdated or unmaintainable over time. This comes with the cost of time during development, but this outweighs the potential cost of system failures, for example, which could prevent the game studio from completing work. Costing the studio money.

# Diagrams

## Structural Diagrams

### Class Diagram

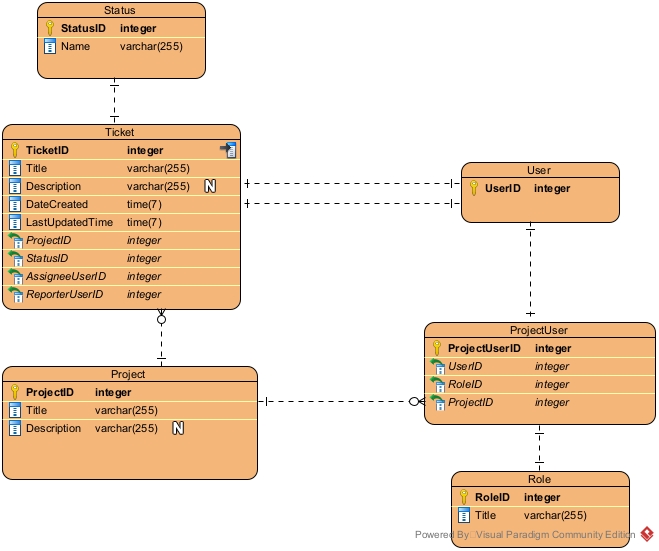


The image here can be found in its full resolution in the GitHub repository documentation/diagrams folder.

This class diagram represents the model view controller pattern that will be used to structure and create the project. I have included the controllers with their views listed within. As well as this I have demonstrated how a repository pattern will be utilized to separate out methods into their given repositories for more maintainable, reusable, and testable code. These repositories will communicate with the database context to store data using the entity framework.

The models can be seen to the right of the controllers, demonstrating an overview of the models and how they relate to one another. This is explored further below in the entity relationship diagram.

### Entity relationships



This diagram demonstrates how the entity framework relation database should be created. The user entity is to make use of a scaffolded Identity framework user and so details are not provided for this. The identity framework is chosen as it should speed up the process of creating and managing users, allowing development to be focused on the bug tracking system.

### Package diagram

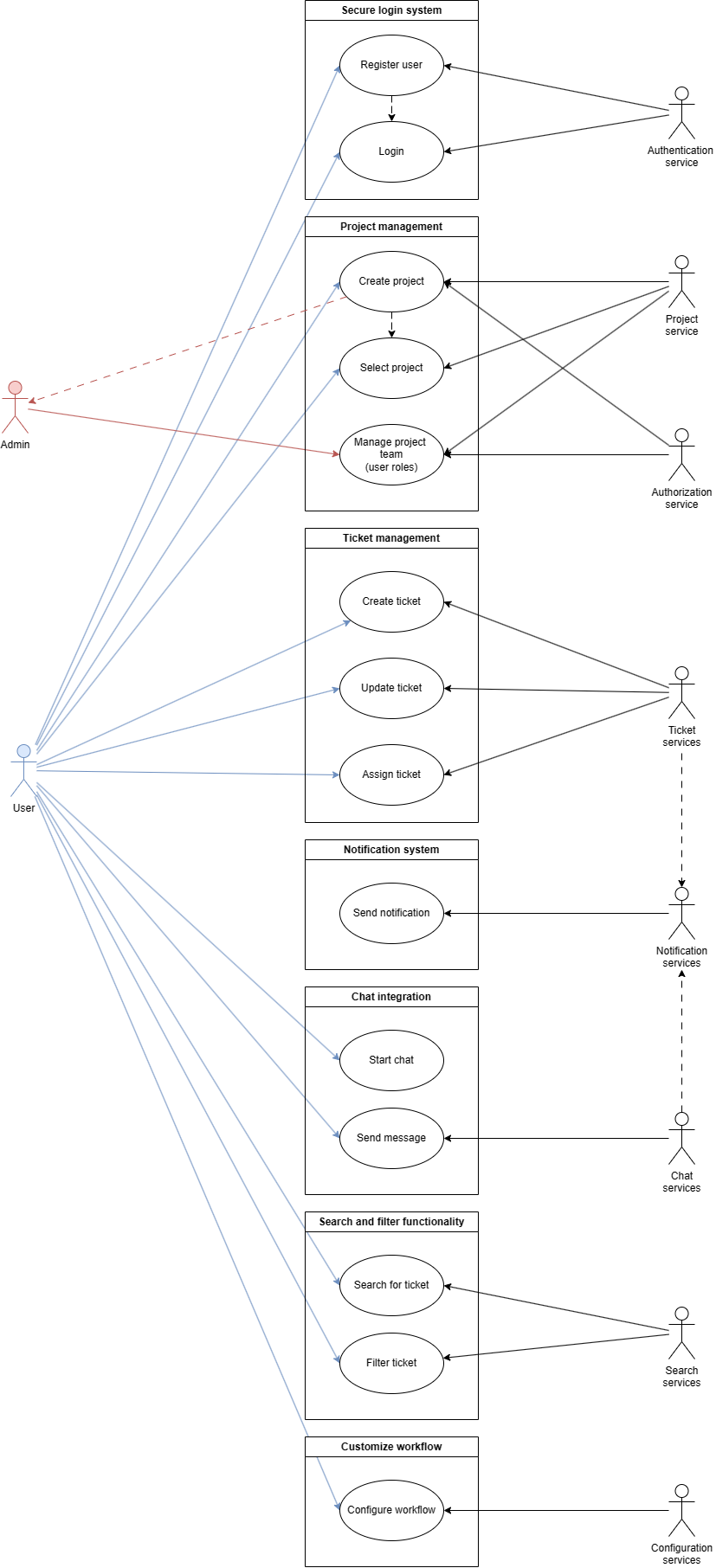
A diagram of a software system

Description automatically generated

Chat integration and notifications are included in the following diagrams but are not required as part of the initial system setup. This is to provide a better understanding of the entire system and expandability is something that should be prepared for and a requirement of the studio for this system. This is also present below in the activity diagrams.

## Behavioural Diagrams

### Use case diagram



### Ticket creation - Object diagram

A diagram of a website

Description automatically generated

### Ticket creation - Activity diagram

A diagram of a flowchart

Description automatically generated

### Ticket management - Activity diagram

A diagram of a flowchart

Description automatically generated