Architecture analysis   
GamifyWork

**­­­­­­­­­­­**



Student name: Rowan van der Weel

Student number: 4924576

Class: <class>

Teacher: <name>

Version: <last version>

Date: <last submission date>

Table of contents

[**Introduction** 3](#_Toc154257593)

[**Diagrams** 3](#_Toc154257594)

[C4 diagram 3](#_Toc154257595)

[Architectural design 3](#_Toc154257596)

[ERD 3](#_Toc154257597)

[**User stories** 3](#_Toc154257598)

[**Wireframes** 3](#_Toc154257599)

[**Version History** 3](#_Toc154257600)

# **Introduction**

In the realm of software development, an architectural analysis serves as a comprehensive documentation of the structural foundation and design decisions behind a project. Its primary purpose is to offer insights into the inner workings of the software, facilitating comprehension, maintenance, and potential future development by different individuals or teams.

For GamifyWork, undertaking an architectural analysis is a strategic move to safeguard the project's longevity and comprehensibility. If key contributors or project stakeholders are unavailable due to unforeseen circumstances, the architectural analysis acts as a reliable reference point. This documentation details the relationships between various components, the flow of data, and the integration of gamification elements with task management.

# **Diagrams**

## C4 diagram

The C4 (Context, Containers, Components, Code) model is an architectural framework that uses hierarchical diagrams to illustrate the structure of a software system. It provides a concise overview, from external interactions in the Context Diagram to internal components and code details. C4 diagrams enhance communication, aid decision-making, and simplify system maintenance.

**Context Diagram**

At the highest level, the Context Diagram provides a bird's-eye view of the system, showing its external actors (such as users, systems, or external services) and the high-level interactions between them. It serves to establish the boundaries and context within which the software system operates. For GamifyWork, it doesn’t make use of any external API’s nor users, so it’s a simple one.

Afbeelding met tekst, schermopname, Lettertype, Elektrisch blauw

Automatisch gegenereerde beschrijving

Figure 1 Context diagram

**Container Diagram**

The Container Diagram focuses on the internal structure of the system, illustrating the major containers (e.g., web servers, databases, desktop applications) and their interactions. It helps to identify the key components that make up the system and the relationships between them. In the case of this application, React is utilized for the frontend, ASP.NET Core serves as the backend, and MySQL functions as the database. Notably, Keycloak has been seamlessly integrated with both the frontend and backend, ensuring robust user protection measures.

Afbeelding met tekst, schermopname, Lettertype, logo

Automatisch gegenereerde beschrijving

Figure 2 Container diagram

**Component Diagram**

Moving deeper into the system, the Component Diagram breaks down each container into individual components or classes. It provides a detailed view of the internal workings of containers, showcasing the building blocks of the system and their interactions. This level of abstraction is useful for understanding how different components collaborate to achieve specific functionalities. For the API, I've organized the architecture into distinct layers, and the specific contents of each layer can be found [here](#_Architectural_design).  
  
Afbeelding met tekst, schermopname, diagram, Lettertype

Automatisch gegenereerde beschrijving

Figure 3 Component diagram

**Code (or Class) Diagram**

At the lowest level, the Code Diagram zooms in on the details of individual components, representing classes, interfaces, and their relationships. This level of granularity is particularly useful for developers and provides insights into the actual implementation of the software. For GamifyWork, no Code (or Class) Diagram has been made, primarily due to the solo development context. Working alone on this project, the decision was made to focus on the higher levels of the C4 model—Context, Containers, and Components.

## Architectural design

The image attached to this heading is a diagram representing the architecture of a software application, broken down into different layers. This is a common way to illustrate the separation of concerns in an application, showing how different parts of the codebase interact with one another. Here's a breakdown of each part:

**Presentation Layer**

This is the topmost layer, typically responsible for handling user interaction. It contains:

* Controllers: These are components that handle user requests and determine the response to send back to the user.
* Middleware: These are software components that are executed in the sequence of request processing and can perform various tasks, such as authentication, logging, etc.

**Business Layer**

This layer implements the core functionality of the application, its business logic.

* Service Interface: Defines the contracts for the services that will be provided.
* Services: Contains the implementation of the business logic.
* Exceptions: Custom exceptions for the business logic.
* Models: Data structures used by the business layer.
* Mapper Interface: Defines the contract for mapping between different data models, typically between the database and business models.

**Contract Layer**

This layer defines the contracts/interfaces and data transfer objects (DTOs) that are shared between layers.

* Interfaces: Abstract definitions of functionality, without implementation.
* DTOs (Data Transfer Objects): Lightweight objects for transferring data between layers.

**Data Access Layer**

This layer provides access to data stored in a persistent storage, like a database.

* Repository Interfaces: Abstractions for accessing data sources that define CRUD operations.
* Repositories: Concrete implementations of the repository interfaces.
* Entities: Domain-specific objects that are mapped to the database tables.
* Data: Represents the data access implementations.

**Data Source**

The actual storage system (database).

**Mapper Layer**

This is often used to map between different data models, particularly between the database models (Entities) and the DTOs/business objects.

**Relations**

The arrows labelled "Implements" and "References" indicate relationships between the layers, showing how the upper layers implement the interfaces defined in the lower layers and reference (use) the lower layers' components. This separation helps in isolating the user interface from business logic and data access code, which makes the application easier to maintain and scale.

Afbeelding met tekst, schermopname, nummer, software

Automatisch gegenereerde beschrijving

Figure 4 Architectural design

## ERD

An Entity-Relationship Diagram (ERD) is a visual representation that illustrates the relationships between entities within a database system. Entities are depicted as tables, and relationships between them are represented by connecting lines. ERDs are fundamental tools in database design, providing a clear and concise overview of the structure and connections between various entities. By visualizing the entities and their relationships, stakeholders gain insights into the organization of data, helping to ensure the effective design and maintenance of a relational database system.

**User:** Represents the users of the system.

**Group:** Represents groups that users can belong to.

**Task:** Represents tasks that users can create or be assigned to.

**Label:** Represents labels that can be assigned to tasks for categorization.

**TaskLabel:** A junction table that implements the many-to-many relationship between tasks and labels.

**Reward:** Represents rewards that users can earn, likely by completing tasks.

Each table has a primary key, which is a unique identifier for its records. For example, User\_ID is the primary key for the User table, which means each user in the system has a unique User\_ID.

Foreign keys are keys that refer to the primary key in another table, creating a link between the two tables. For example, the User table has a Group\_ID foreign key, which links it to the Group table, indicating which group the user belongs to.

Each table has attributes, which are the data fields stored within it. For instance, the User table includes Username, Email, and Profile Picture, among others.

**Relationships**

One-to-Many (1: N): Represented by a line that ends with a crow's foot symbol. For instance, one Group can have many User records associated with it, but a User belongs to only one Group.

Many-to-Many (N:M): Represented by using a junction table. Task and Label have a many-to-many relationship via the TaskLabel junction table.

Afbeelding met diagram, Parallel, Rechthoek, tekst

Automatisch gegenereerde beschrijving

Figure 5 Entity-relationship diagram

# **User stories**

|  |  |  |
| --- | --- | --- |
| **Title:** View tasks in homepage | **Priority:** EXTREME | **Estimate: 13** |
| **User Story:**  As a user, I want to view all my tasks and reward-tasks in the home screen, so that I immediately know what to do and what I’ve completed. | | |
| **Acceptance Criteria:**  The home screen should display a list of all tasks, categorizing them as to-dos and recurring tasks and reward-tasks.  The user could filter on “completed”, “active” and “all”. When the task is not in editing mode (just normal in the homepage) only the title and description are visible. | | |

\* Check for the wireframe to get an idea

|  |  |  |
| --- | --- | --- |
| **Title:** Create tasks | **Priority:** Critical | **Estimate: 3** |
| **User Story:**  As a user, I want to create to-does and recurring tasks, for my personal planning. | | |
| **Acceptance Criteria:**  Users should be able to create tasks first with their title. The title should be max 50 characters.  The user should create the task in the correct boxes: “daily tasks” and “to-do”. If the user creates a task in the first box, there should be an option for the task to recurring itself (even after its marked as completed). | | |

|  |  |  |
| --- | --- | --- |
| **Title:** Mark tasks | **Priority:** Critical | **Estimate: 5** |
| **User Story:**  As a user, I want to mark tasks when I completed them, so I can see the progress I’ve made and prevent for doing things twice. | | |
| **Acceptance Criteria:**  User can click on a task to mark it as completed.  There should be a visible action, like a checkbox or button, associated with each task.  Clicking this action should change the status of the task to "completed".  If a task is already marked and you press it to mark, it will be unmarked.  So, the status from “completed” will change to “active”. | | |

|  |  |  |
| --- | --- | --- |
| **Title:** Edit tasks | **Priority:** High | **Estimate: 5** |
| **User Story:**  As a user, I want to edit my tasks, so I can fix it if I have accidentally made some errors. | | |
| **Acceptance Criteria:**  Users should have the option to edit task titles, descriptions, and other details (perhaps a check-off list or end date).  Changes made to tasks should be reflected immediately. | | |

|  |  |  |
| --- | --- | --- |
| **Title:** Delete tasks | **Priority:** High | **Estimate: 2** |
| **User Story:**  As a user, I want to delete tasks, so I won’t have to do them. | | |
| **Acceptance Criteria:**  Given the user has made a task, when they click on the task and scroll down, there should be a button called “delete task”.  If the user presses that button there should be a warning with “Are you sure you want to delete the task?”.  After the user deletes it, the deleted tasks should no longer appear on the task list. | | |

|  |  |  |
| --- | --- | --- |
| **Title:** Recurring tasks | **Priority:** High | **Estimate: 8** |
| **User Story:**  As a user, I want the option to set up recurring tasks, so I won’t have to create them again every time. | | |
| **Acceptance Criteria:**  Users should be able to customize their tasks, including frequency (daily, weekly, monthly) and how many time a week/month/year. | | |

|  |  |  |
| --- | --- | --- |
| **Title:** Earn points | **Priority:** High | **Estimate: 5** |
| **User Story:**  As a user, I want to earn points for completing tasks, so I’ll stay motivated to do more tasks. | | |
| **Acceptance Criteria:**  Users should receive points upon completing tasks, and the points should be added to their account.  The points should be visible in the home-screen. | | |

|  |  |  |
| --- | --- | --- |
| **Title:** Spend points | **Priority:** High | **Estimate: 5** |
| **User Story:**  As a user, I want to be able to spend points, so it makes me eager to earn more points and do more tasks. | | |
| **Acceptance Criteria:**  Users should have options to spend their points on rewards either in a virtual shop, or in real life (for example watch tv).  In your profile you should be an able to see the items you bought. | | |

|  |  |  |
| --- | --- | --- |
| **Title:** Create rewards | **Priority:** Medium | **Estimate: 3** |
| **User Story:**  As a user, I want to create reward-tasks, so I can have fun as well. | | |
| **Acceptance Criteria:**  Users should be able to create reward-tasks first with their title. The title should be max 50 characters.  After it created a user can edit it, add a description and a fee. | | |

|  |  |  |
| --- | --- | --- |
| **Title:** Label tasks | **Priority:** Medium | **Estimate: 8** |
| **User Story:**  As a user, I want to label my tasks, so I have a better view of my tasks. | | |
| **Acceptance Criteria:**  Users should be able to assign labels and delete labels.  Tasks should be filterable or searchable by these labels. | | |

|  |  |  |
| --- | --- | --- |
| **Title:** Add description | **Priority:** Medium | **Estimate: 3** |
| **User Story:**  As a user, I want the ability to add notes/descriptions (perhaps a check-off list or end date) to my tasks, so that I have a better understanding of it. | | |
| **Acceptance Criteria:**  Users should be able to add detailed notes or descriptions to their tasks with a max of 255 characters.  The notes/descriptions should be accessible when viewing the task details. | | |

|  |  |  |
| --- | --- | --- |
| **Title:** Create account | **Priority:** Medium | **Estimate: 8** |
| **User Story:**  As a user, I want to be able to create an account, so I can start using GamifyWork. | | |
| **Acceptance Criteria:**  The registration process should include fields for username, email, and password.  Users should receive a confirmation email upon registration for account verification.  The password should be at least 8 characters including a special character. | | |

|  |  |  |
| --- | --- | --- |
| **Title:** Groups | **Priority:** Medium | **Estimate: 13** |
| **User Story:**  As a user, I want to join and create groups, so I can see the ranking and it creates a sense of community and competition. | | |
| **Acceptance Criteria:**  Users should be able to create and join groups or communities.  In the group there should be a leaderboard based on users' task completion and points.  (Competition between al the users in the group.) | | |

|  |  |  |
| --- | --- | --- |
| **Title:** Add friends | **Priority:** Low | **Estimate: 13** |
| **User Story:**  As a user, I want to add friends with other users, so I can compete and talk to them. | | |
| **Acceptance Criteria:**  Users should have the ability to send and accept friend requests.  Friends should be listed in a user's profile.  The user should be able to chat with the other users. | | |

|  |  |  |
| --- | --- | --- |
| **Title:** Welcome guide | **Priority:** Low | **Estimate: 5** |
| **User Story:**  As a user, I want a welcome guide, so that I have an idea how the application works. | | |
| **Acceptance Criteria:**  Upon logging in for the first time, users should be presented with an interactive or informative welcome guide.  The guide should introduce users to the key features and functionality of GamifyWork.  Users should have the option to skip the guide or revisit it later if needed. | | |

|  |  |  |
| --- | --- | --- |
| **Title:** Profile picture | **Priority:** Low | **Estimate: 5** |
| **User Story:**  As a user, I want to create a profile picture, so people can recognize me. | | |
| **Acceptance Criteria:**  Users should be able to upload or choose a profile picture during or after registration.  The profile picture should be displayed in the user's profile. | | |

|  |  |  |
| --- | --- | --- |
| **Title:** Google auth | **Priority:** Low | **Estimate: 13** |
| **User Story:**  As a user, I want the option to sign in with Google, so I won’t have to enter a mail and do the verification. | | |
| **Acceptance Criteria:**  Users should see a "Sign up with Google" and a "Sign up with Google" option on the login screen.  Clicking the "Sign in with Google" option should authenticate the user with their Google account and log them into GamifyWork.  Clicking the "Sign up with Google" option should authenticate the user with their Google account and validate them as a valid user. | | |

# **Wireframes**

A wireframe is a basic visual representation of a web page, application, or product interface. It serves as a skeletal outline that outlines the structure and layout of key elements without incorporating design details such as colours or graphics. Wireframes are essential in the early stages of the design process as they provide a clear and simplified blueprint for the user interface. Their importance lies in facilitating communication among team members, including designers, developers, and stakeholders, by offering a visual reference for the intended layout and functionality. By focusing on the structure and placement of elements, wireframes streamline the design process, allowing for early feedback and iteration before investing time and resources in detailed design elements. They act as a valuable tool for aligning the vision of the project, refining user experience, and ultimately ensuring the efficient development of a user-friendly and visually coherent interface. These are the wireframes for GamifyWork.

Afbeelding met tekst, schermopname, software, Lettertype

Automatisch gegenereerde beschrijving

Figure 6 Homepage

Afbeelding met tekst, schermopname, software, Computerpictogram

Automatisch gegenereerde beschrijving

Figure 7 Friend page

# **Version History**

|  |  |
| --- | --- |
| **When?** | **What?** |
| 23/12/2023 | First initialization, Introduction finished. |
| 24/12/2023 | Diagram finished |
| 30/12/2023 | User stories and wireframe finished |