System Handbook

Subtitle

Prepared by:

<name> | <email>

Compiled and Completed:

<date>

# Team

|  |  |  |
| --- | --- | --- |
| **Role** | **Member** | **Responsibilities** |
|  |  |  |

# 

# System Rationale

## Description

## 

## Objectives

## 

## Significance

## Scope

# Business Analysis

## Use Case Diagram

A Use Case Diagram is a graphical representation of the interactions between a system and its users, known as actors. It is used to capture the functional requirements of a system and to model the user's interactions with that system.

A use case diagram consists of three main elements:

* Actors: Represent the users of the system, including external entities such as people, organizations, or other systems.
* Use Cases: Represent the functions or services provided by the system to its actors. Each use case is a sequence of actions that the system performs to achieve a specific goal for the actor.
* Relationships: Show the relationships between actors and use cases, including generalization (an actor can perform multiple use cases), include (one use case includes the functionality of another), and extend (a use case extends the functionality of another).

There are three main relationships in a use case diagram:

* Generalization: This relationship indicates that one actor can perform multiple use cases.
* Include: This relationship indicates that one use case includes the functionality of another use case.
* Extend: This relationship indicates that a use case extends the functionality of another use case.

These relationships help to organize and simplify the use cases by showing how they are related and dependent on each other. Understanding these relationships is important for identifying potential problems and for making design decisions.

Use case diagrams are useful for defining the scope of a system, understanding the requirements of the users, and for communication between stakeholders. They provide a high-level view of the system and are useful for identifying potential problems early in the development process.

## Data Flow Diagram

A Data Flow Diagram (DFD) is a graphical representation of the flow of data through a system. It is a modeling tool used in systems analysis and design to illustrate how data is processed within a system, from its source to its final destination.

DFDs consist of four basic components:

* Processes: Represent the operations performed on data, such as calculation, data manipulation, and data storage.
* Data Flows: Represent the movement of data between processes.
* Data Stores: Represent storage locations for data, such as databases, files, and buffers.
* External Entities: Represent the sources or destinations of data outside of the system, such as people, organizations, or other systems.

DFDs are used to capture the data requirements of a system, to understand how data is processed, and to identify opportunities for process improvement and data optimization. They provide a high-level view of the system, making it easier to identify areas that need further attention and design. Providing this, it is achieved which physical process forms are transferred per process in the current system.

## Entity Relationship Diagram

An Entity Relationship Diagram (ERD) is a graphical representation of entities and their relationships to each other, used in database design. It provides a high-level view of the structure of a database, including entities and the relationships between them.

An ERD consists of three main components:

* Entities: Represent objects or concepts in the real world, such as customers, orders, or products.
* Attributes: Represent characteristics or properties of an entity, such as name, address, or date of birth.
* Relationships: Represent the relationships between entities, such as a customer placing an order or a product being part of an order. Relationships can be one-to-one, one-to-many, or many-to-many.

ERDs are used to define the structure of a database and to communicate the relationships between entities to stakeholders. They are an important tool in database design, helping to ensure that the database is properly structured to support the needs of the system. ERDs can also be used to identify potential data redundancies, improve data integrity, and simplify database maintenance.

## CRUD Matrix

A CRUD Matrix, also known as a Create-Read-Update-Delete (CRUD) Matrix, is a table used in software development to identify and document the operations that can be performed on data within a system. It provides a high-level view of the functionality of the system and helps to ensure that all necessary operations are accounted for in the design.

The matrix consists of four basic operations:

* Create (C): Refers to the ability to add new data to the system.
* Read (R): Refers to the ability to retrieve or view existing data in the system.
* Update (U): Refers to the ability to modify existing data in the system.
* Delete (D): Refers to the ability to remove data from the system.

Each row in the matrix represents a data item or an object, and each column represents a specific operation. The intersection of a row and a column represents the specific operation that can be performed on the data item.

The CRUD matrix is a useful tool in ensuring that all necessary operations are accounted for in the design of the system and in identifying potential issues early in the development process.

## Sequence Diagram

A Sequence Diagram represents the interactions between objects or components in a system. It shows the sequences of messages exchanged between objects over time, and provides a visual representation of the interactions between objects and the order in which these interactions occur.

A sequence diagram consists of several elements, including:

* Objects or components: Represent the objects or components in the system, such as classes, functions, or modules.
* Messages: Represent the interactions between objects or components. Each message is represented by an arrow pointing from the sender object to the receiver object.

Sequence diagrams are useful for capturing the interactions between objects or components in a system, and for visualizing the flow of control in the system. They ensure that the interactions between objects or components are understood and properly designed.

## Activity Diagram

An Activity Diagram represents the flow of activities in a system. It provides a visual representation of the steps involved in a process or activity, and the flow of control between these steps.

An activity diagram consists of several elements, including:

* Activities: Represent the steps or tasks involved in a process or activity. They are represented as rounded rectangles.
* Transitions: Represent the flow of control between activities. They are represented as arrows connecting activities.
* Decision nodes: Represent decisions or branches in the flow of control. They are represented as diamonds.
* Initial and final nodes: Represent the starting and ending points of a process or activity. The initial node is represented as a filled circle, and the final node is represented as a bull's eye.
* Swimming lanes: Represent the partitioning of activities into pools, which represent different participants or roles in the process or activity.

Activity diagrams are useful for modeling the flow of control in a system and for understanding the relationships between activities.

Technical Architecture

# Tech Stack

A tech stack is the combination of technologies a company uses to build and run an application or project. Sometimes called a “solutions stack,” a tech stack typically consists of programming languages, frameworks, a database, front-end tools, back-end tools, and applications connected via APIs.

# Project File Structure

# Class Diagram

A Class Diagram represents the classes, objects, and their relationships in a system. It provides a blueprint of the classes and their attributes, operations, and relationships with other classes.

A class diagram consists of several elements, including:

* Classes: Represent the objects and entities in a system, and are represented as rectangles.
* Attributes: Represent the data or properties of a class, and are represented as a list of variables within the class rectangle.
* Operations: Represent the behavior or actions of a class, and are represented as a list of methods within the class rectangle.
* Relationships: Represent the relationships between classes, such as inheritance, aggregation, composition, and association. They are represented as arrows connecting classes.

Class diagrams are useful for understanding the structure of a system and for illustrating the relationships between classes.

# Deployment Diagram

A Deployment Diagram represent the physical deployment of software components and hardware components in a system. It provides a visual representation of the mapping of software components to hardware components, such as servers, databases, and clients.

A deployment diagram consists of several elements, including:

* Nodes: Represent the hardware components in a system, such as servers, clients, or databases.
* Components: Represent the software components in a system, such as modules, libraries, or executables.
* Communication Paths: Represent the communication between nodes, such as network connections or communication protocols.
* Artifacts: Represent the physical files that are generated or deployed as part of the system, such as executables or configuration files.

Deployment diagrams are useful for understanding the physical deployment of software components and hardware components in a system. They can be used in a variety of domains, including software development, system design, and architecture.

References

Links to process forms, etc