Deliverable #2 Template

SE 3A04: Software Design II – Large System Design

Tutorial Number: T03
Group Number: G07
Group Members:

- Farid Bastoros
- Neha Bhatla
- Omar Alam
- Luka Mahrt-Smith
- Aidan Lao

IMPORTANT NOTES

- Please document any non-standard notations that you may have used
  - *Rule of Thumb*: if you feel there is any doubt surrounding the meaning of your notations, document them
- Some diagrams may be difficult to fit into one page
  - Ensure that the text is readable when printed, or when viewed at 100% on a regular laptop-sized screen.
  - If you need to break a diagram onto multiple pages, please adopt a system of doing so and thoroughly explain how it can be reconnected from one page to the next; if you are unsure about this, please ask about it
- Please submit the latest version of Deliverable 1 with Deliverable 2
  - Indicate any changes you made.
- If you do NOT have a Division of Labour sheet, your deliverable will NOT be marked

1 Introduction

1.1 Purpose

This document provides a high-level overview of the Shroomify system architecture, outlining the core design principles, subsystems, and the rationale behind architectural choices. It is intended for internal stakeholders involved in the development of Shroomify, including software engineers, project managers, investors, domain experts, and system architects. Prior technical knowledge is beneficial but not required for understanding this document.
Please ensure that Shroomify Deliverable 1 is reviewed before Deliverable 2, as it provides essential context. Additionally, having technical knowledge may help in better understanding the contents of this document.

1.2 System Description

The Shroomify system follows a Model-View-Controller (MVC) architecture, separating data management, business logic, and user interactions. This type of architecture has Model components, View components and Controller components which increase its scalability and maintainability, allowing each component to be tested and updated independently.

Additionally, Shroomify integrates Blackboard architecture and Repository architecture for different subsystems based on their functionality. The repository architecture was used for subsystems that require efficient data storage and retrieval, allowing for concurrent access to shared data repositories. The Blackboard architecture was used for subsystems that need to combine different pieces of information to reach a final result, making it useful for tasks like decision-making and identification.

1.3 Overview

This is Deliverable 2 of the Shroomify project and provides a general description of the system architecture. It considers the requirements, use cases, and design decisions outlined in Deliverable 1. Section 2 has an Analysis Class Diagram that represents the major classes and how they relate to each other from our requirements analysis. This diagram establishes the foundation for the organization of the system internally and ensures that design is high in cohesion and low in coupling among its components. Section 3 presents a description of the Architectural Design of Shroomify. In this section, we define the chosen architectural pattern and discuss specific design decisions. Section 4 focused on Class Responsibility Collaboration (CRC) Cards. These cards capture the responsibilities and communications of each class within the system, detailing exactly how the individual components interact to satisfy the demands of the system.
Collectively, these sections provide a solid foundation for understanding the structure and functionality of the Shroomify app.

2 Analysis Class Diagram

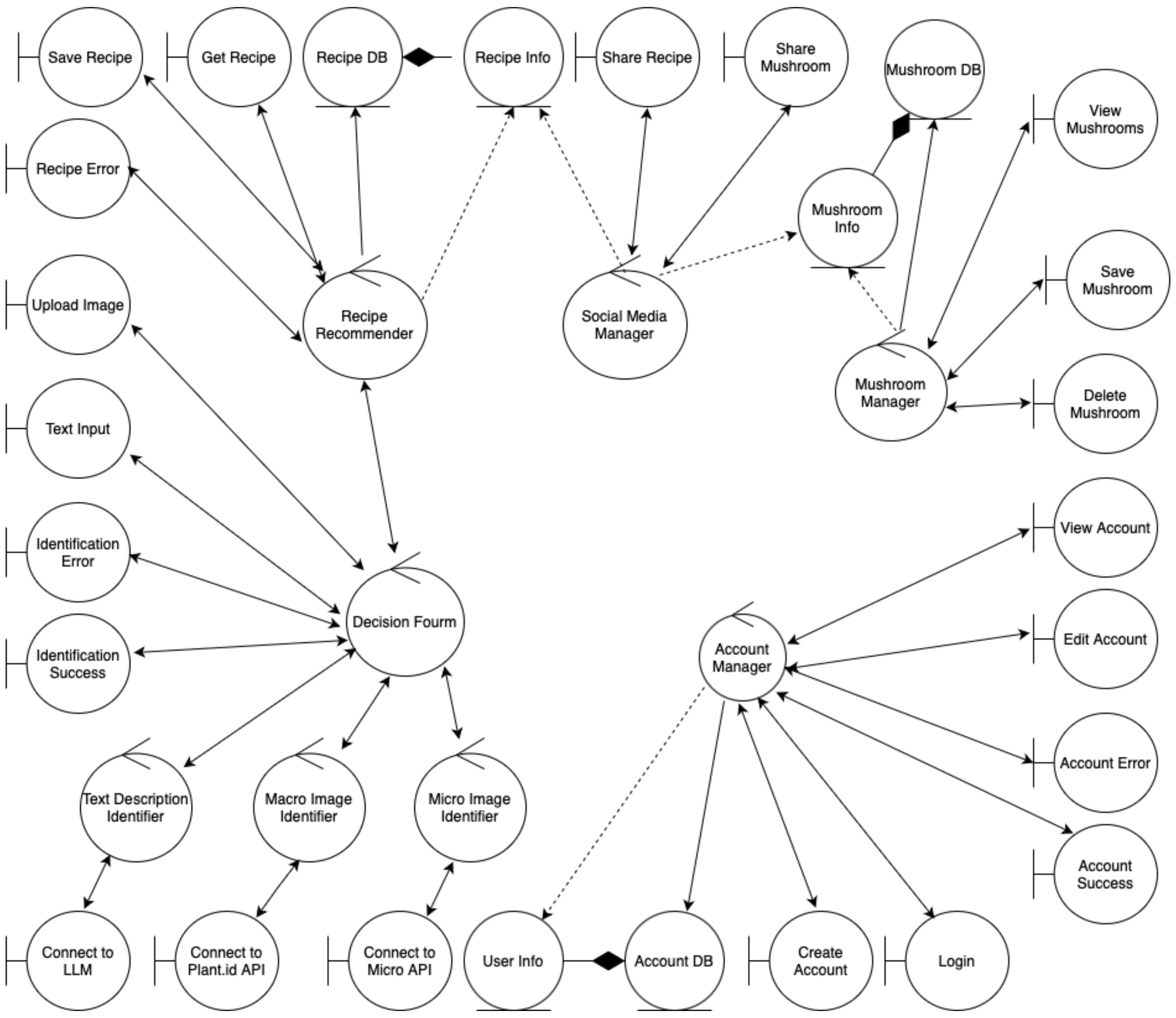

Figure 1: Analysis Class Diagram

3 Architectural Design

3.1 System Architecture

Shroomify uses the **Model-View-Controller (MVC)** architectural pattern as the main architecture of the system. Within the MVC pattern, the system is divided into three main component types: the **Model** components, the **View** components, and the **Controller** components. The **Model** components are responsible for managing data storage in the system. The **View** components are responsible for displaying the various user interface pages. The **Controller** components are responsible for handling user input and updating the models and views accordingly. The MVC pattern was chosen for Shroomify because it provides a clear separation of concerns between the different components of the system, making it easier to maintain and extend the system in the future. The MVC pattern also allows for the system to be easily tested, as each component can be tested independently of the others.

For each subsystem, we have identified different architectures that will be used:

Subsystem	Tasks	Architecture
Decision Forum	Take partial solutions from expert agents and produce final identification solution.	Blackboard Architecture
Mushroom Management	Store and modify information related to mushrooms identified by the user in a database.	Repository Architecture
Account Management	Manage account registration and modification.	Repository Architecture
Social Media Management	Handle information sharing with external social media platforms.	Model-View-Controller Architecture
Recipe Recommendation	Recommend recipes to user based on identified mushrooms.	Repository Architecture

Table 1: Subsystem Architectures

3.1.1 Decision Forum Architecture

The blackboard architecture was chosen for the Decision Forum subsystem because it is well-suited to combine partial solutions from different expert software agents such as the **Text Description Identifier**, **Macro Image Identifier**, and **Micro Image Identifier**. The blackboard architecture allows these agents to work independently and asynchronously, and then combine their partial solutions to produce a final identification solution. This architecture is well-suited to the Decision Forum subsystem because it allows for the integration of multiple expert agents with different areas of expertise, and provides a flexible and extensible framework for combining their solutions. It also allows for us to easily modify each agent independently without affecting the overall system and also add more agents if needed in the future without any modification of pre-existing code. This architecture lets us apply the open-closed principle well which will help us with maintainability in the future. Another architecture that was considered for this subsystem was the Model-View-Controller architecture, but it was not chosen because the Decision Forum subsystem is not primarily focused on user interaction and does not require a user interface. MVC also does not provide a clear way to combine the partial solutions from different expert agents.

3.1.2 Mushroom Management Architecture

The repository architecture was chosen for the Mushroom Management subsystem because it is well-suited for the simple data storage and retrieval tasks required by this subsystem. The repository architecture provides a data store that can be accessed by software agents in different ways.

3.1.3 Account Management Architecture

The repository architecture was chosen for the Account Management subsystem for the same reasons as the Mushroom Management subsystem. The Account Management subsystem also requires simple data storage and retrieval tasks, and the repository architecture provides a flexible data store system.

3.1.4 Social Media Management Architecture

The Model-View-Controller architecture was chosen for the Social Media Management subsystem since a user interaction will be required to share information with external social media platforms. This entails having a user interface to interact with the system (View), an agent to handle user input (Controller), and a data store to manage the retrieve and store the information being shared (Model). The pipe and filter architecture was considered because of the nature of the subsystem sending streams of information to external platforms, and it may be contained within the controller of the architecture but is not well-suited to be the main architecture for the subsystem.

3.1.5 Recipe Recommendation Architecture

The repository architecture was chosen for the Recipe Recommendation subsystem since the Recommendation system is intended to be a simple search and compare algorithm from a database. The architecture provides a data store (database) which can be efficiently searched for recipes based on the identified mushrooms. The blackboard architecture was considered for this subsystem because it could potentially combine multiple recommendation algorithms to produce a final recommendation. However, the complexity of the recommendation system is not high enough to warrant the use of the blackboard architecture.

The following diagrams show the structural architecture of the system, illustrating the relationships between different controllers, databases and views.

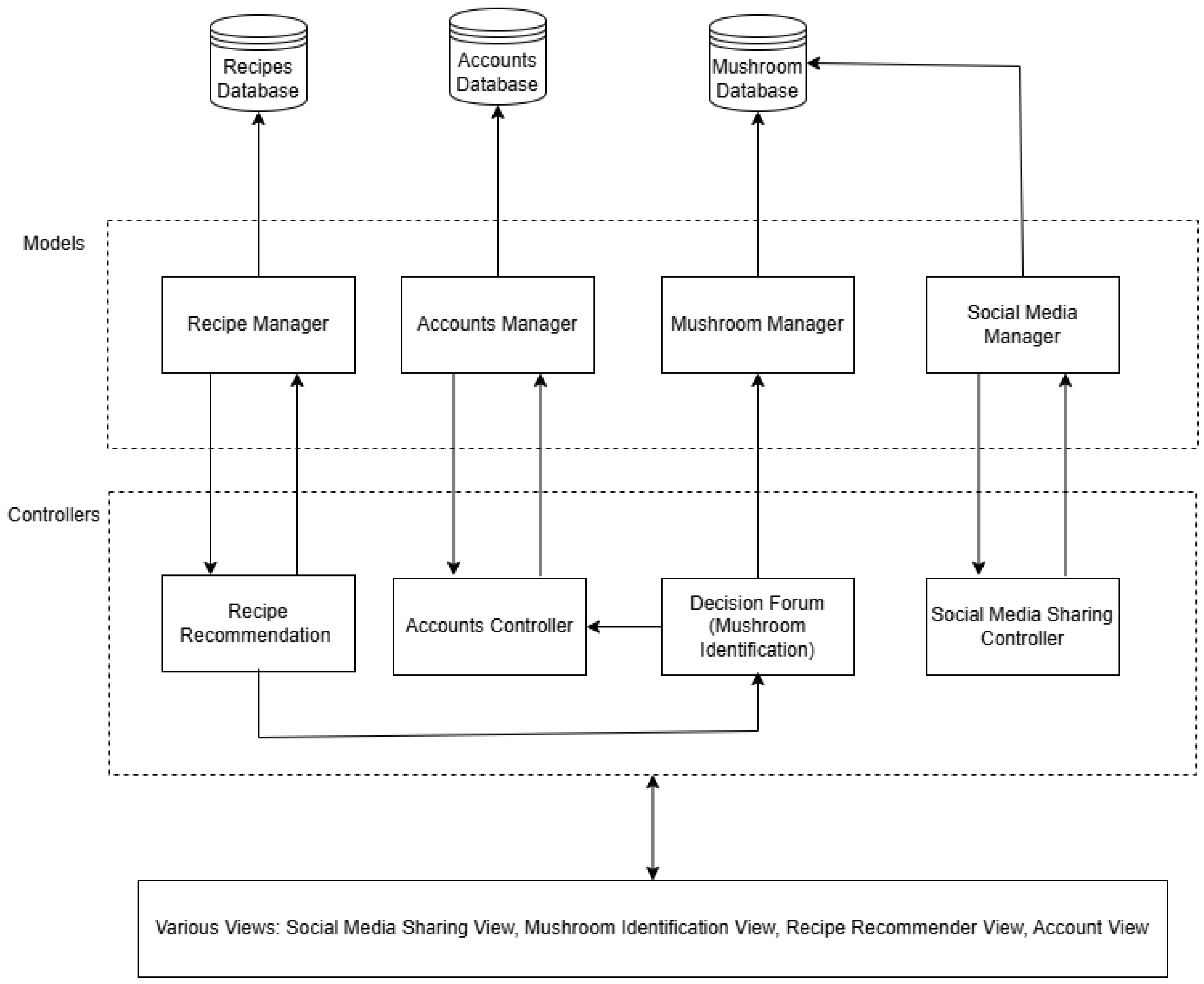
Recipes Database
Accounts Database
Mushroom Database
Models
Recipe Manager
Accounts Manager
Mushroom Manager
Social Media Manager
Controllers
Recipe Recommendation
Accounts Controller
Decision Forum (Mushroom Identification)
Social Media Sharing Controller
Various Views: Social Media Sharing View, Mushroom Identification View, Recipe Recommender View, Account View

Figure 2: Model-View-Controller Architecture

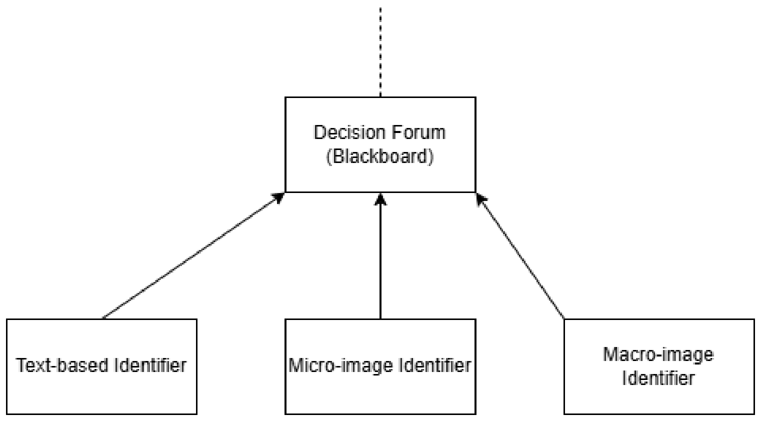

Figure 3: Blackboard Architecture for Decision Forum

3.2 Subsystems

Our system is divided into five main subsystems: **Decision Forum, Mushroom Management, Account Management, Social Media Management, and Recipe Recommendation**. These subsystems interact to provide users with a seamless experience in identifying mushrooms, managing their findings, and sharing information.

3.2.1 Decision Forum

The **Decision Forum** subsystem is responsible for identifying mushrooms. It contains the **machine learning algorithm** that classifies mushrooms based on user-provided descriptions and **computer vision technology** that analyzes images to determine species. If the identification confidence is low, users can also post images and descriptions to receive verification from the community.

Purpose Automates mushroom identification using AI and provides a space for community-based verification.

Relationship to Other Subsystems

- Works with **Mushroom Management** to store confirmed identifications.
- Interfaces with **Account Management** to associate identifications with users.
- Can interact with **Social Media Management** for users to share identification discussions.

3.2.2 Mushroom Management

The **Mushroom Management** subsystem serves as a **personal dictionary** of previously identified mushrooms. It allows users to view mushrooms they've found in the past, track their quantity, and access relevant information, including images and descriptions.

Purpose Stores and displays user-identified mushrooms but does **not** perform identification itself.

Relationship to Other Subsystems

- Retrieves identification results from **Decision Forum** and saves them.
- Integrates with **Account Management** to store mushroom data within user profiles.
- Interfaces with **Recipe Recommendation** to provide relevant recipes for identified mushrooms.

3.2.3 Account Management

The **Account Management** subsystem handles user authentication, profile settings, and data storage. It maintains a personalized record of identified mushrooms and user activity.

Purpose Manages user profiles and ensures that each user's mushroom data is securely stored.

Relationship to Other Subsystems

- Stores user-identified mushrooms from **Mushroom Management**.
- Maintains user participation history in **Decision Forum**.
- Interfaces with **Social Media Management** to track and share posts.

3.2.4 Social Media Management

The **Social Media Management** subsystem allows users to share their mushroom identifications, forum discussions, and recipes on external platforms such as Facebook, Instagram, and Twitter.

Purpose Enhances community engagement by enabling content sharing.

Relationship to Other Subsystems

- Works with **Account Management** to post from user profiles.
- Allows users to share discoveries from **Mushroom Management** and **Decision Forum**.
- Enables recipe-sharing from **Recipe Recommendation**.

3.2.5 Recipe Recommendation

The **Recipe Recommendation** subsystem suggests safe and relevant recipes based on identified mushrooms. It retrieves information from a database and presents culinary suggestions to users.

Purpose Provides users with potential culinary uses for their identified mushrooms.

Relationship to Other Subsystems

- Works with **Mushroom Management** to access user-identified mushrooms.
- Allows users to share recipes through **Social Media Management**.

4 Class Responsibility Collaboration (CRC) Cards

This section should contain all of your CRC cards.

- Provide a CRC Card for each identified class

Class Name: Recipe Recommender (Controller)	
Responsibility:	**Collaborators:**
- Retrieves and recommends recipes based on user preferences and available ingredients.	Get Recipe
- Saves user-selected recipes to the database for future access.	Save Recipe
- Handles errors related to recipe retrieval and recommendation failures.	Recipe Error
- Communicates with the recipe database to fetch and store recipe information.	Recipe DB
- Integrates with the Decision Forum to refine recommendations based on expert analysis.	Decision Forum
- Manages and processes recipe-related data for accurate recommendations.	Recipe Info

Class Name: Recipe Info (Entity)	
Responsibility:	**Collaborators:**
Knows Recipe Recommender	Recipe Recommender
Knows Recipe DB	Recipe DB
Knows Social Media Manager	Social Media Manager

Class Name: Recipe DB (Entity)	
Responsibility:	**Collaborators:**
Knows Recipe Recommender Knows Recipe Info	Recipe Recommender Recipe Info

Class Name: Get Recipe (Boundary)	
Responsibility:	**Collaborators:**
Knows Recipe Recommender Handles click event of “Get Recipe” button	Recipe Recommender

Class Name: Save Recipe (Boundary)	
Responsibility:	**Collaborators:**
Knows Recipe Recommender Handles click event of “Save Recipe” button	Recipe Recommender

Class Name: Recipe Error (Boundary)	
Responsibility:	**Collaborators:**
Knows Recipe Recommender Handles Get Recipe error events Handles Save Recipe error events	Recipe Recommender

Class Name: Decision Fourm (Controller)	
Responsibility:	**Collaborators:**
- Receives image and text input for identification requests.	Upload Image
- Forwards text or image input to the relevant identification modules.	Text Input
- Manages identification errors and provides feedback to users.	Identification Error
- Handles successful identification results and forwards them for further processing.	Identification Success
- Coordinates with the Text Description Identifier for text-based identification.	Text Description Identifier
- Coordinates with the Macro Image Identifier for macro image-based identification.	Macro Image Identifier
- Coordinates with the Micro Image Identifier for micro image-based identification.	Micro Image Identifier
- Interfaces with the Recipe Recommender to suggest recipes based on the identification results.	Recipe Recommender

Class Name: Text Description Identifier (Controller)	
Responsibility:	**Collaborators:**
- Connects to the Language Model (LLM) for text-based identification.	Connect to LLM
- Forwards text input to the Decision Forum for further processing.	Decision Forum

Class Name: Macro Image Identifier (Controller)	
Responsibility:	**Collaborators:**
- Connects to the Plant.id API for macro image-based identification. - Forwards macro image input to the Decision Forum for further analysis.	Connect to Plant.id API Decision Forum

Class Name: Micro Image Identifier (Controller)	
Responsibility:	**Collaborators:**
- Connects to the Micro API for micro image-based identification. - Forwards micro image input to the Decision Forum for further analysis.	Connect to Micro API Decision Forum

Class Name: Upload Image (Boundary)	
Responsibility:	**Collaborators:**
Knows Decision Fourm Handles click event of "Upload Image" button	Decision Fourm

Class Name: Text Input (Boundary)	
Responsibility:	**Collaborators:**
Knows Decision Fourm Handles click event of "Text Input" button	Decision Fourm

Class Name: Identification Error (Boundary)	
Responsibility:	**Collaborators:**
Knows Decision Fourm Handles Upload Image error events Handles Text Input error events	Decision Fourm

Class Name: Identification Success (Boundary)	
Responsibility:	**Collaborators:**
Knows Decision Fourm Handles Successful events of Upload Image Handles Successful events of Text Input	Decision Fourm

Class Name: Connect to LLM (Boundary)	
Responsibility:	**Collaborators:**
Knows Text Description Identifier Handles click event of "Identify Mushroom" button	Text Description Identifier

Class Name: Connect to Plant.id API (Boundary)	
Responsibility:	**Collaborators:**
Knows Text Description Identifier Handles click event of "Identify Mushroom" button	Text Description Identifier

Class Name: Connect to Micro API (Boundary)	
Responsibility:	**Collaborators:**
Knows Text Description Identifier Handles click event of "Identify Mushroom" button	Text Description Identifier

Class Name: Account Manager (Controller)	
Responsibility:	**Collaborators:**
- Handles user account creation, authentication, and management. - Retrieves and updates user account information from the database. - Processes login requests and verifies user credentials. - Manages account-related success and error handling. - Provides access to account details and allows modifications when needed. - Ensures data consistency and security during account operations.	Account DB User Info Create Account Edit Account View Account Login Account Success Account Error

Class Name: Account DB (Entity)	
Responsibility:	**Collaborators:**
Knows Account Manager Knows User Info	Account Manager User Info

Class Name: User Info (Entity)	
Responsibility:	**Collaborators:**
Knows Account Manager Knows Account DB	Account Manager Account DB

Class Name: Create Account (Boundary)	
Responsibility:	**Collaborators:**
Knows Account Manager Handle user interactions for creating an account	Account Manager

Class Name: Edit Account (Boundary)	
Responsibility:	**Collaborators:**
Knows Account Manager Handles click-event of ”Edit Account” button	Account Manager

Class Name: View Account (Boundary)	
Responsibility:	**Collaborators:**
Knows Account Manager Handles click-event of ”View Account” button	Account Manager

Class Name: Login (Boundary)	
Responsibility:	**Collaborators:**
Knows Account Manager Handles click-event of ”Login” button	Account Manager

Class Name: Account Success (Boundary)	
Responsibility:	**Collaborators:**
Knows Account Manager Handles successful events for account creation/editing	Account Manager

Class Name: Account Error (Boundary)	
Responsibility:	**Collaborators:**
Knows Account Manager Handles Login error events Handles Edit Account error events Handles View Account error events	Account Manager

Class Name: Mushroom Manager (Controller)	
Responsibility:	**Collaborators:**
- Receives mushroom identification requests (text/image) from boundary classes.	Mushroom DB
- Forwards input to the appropriate identification modules for analysis.	Mushroom Info
- Aggregates and resolves identification results from expert modules.	View Mushrooms
- Sends the final identification result to the requesting boundary class.	Save Mushroom
- Manages saving, updating, and deleting mushroom records in the database.	Delete Mushroom
- Handles failed identification attempts and provides appropriate feedback.	

Class Name: Mushroom DB (Entity)	
Responsibility:	**Collaborators:**
Knows Mushroom Manager	Mushroom Manager
Knows Mushroom Info	Mushroom Info

Class Name: Mushroom Info (Entity)	
Responsibility:	**Collaborators:**
Knows Mushroom Manager	Mushroom Manager
Knows Social Media Manager	Social Media Manager

Class Name: View Mushrooms (Boundary)	
Responsibility:	**Collaborators:**
Knows Mushroom Manager Handles click-event of ”View Mushrooms” button	Mushroom Manager

Class Name: Save Mushroom (Boundary)	
Responsibility:	**Collaborators:**
Knows Mushroom Manager Handles click-event of ”Save Mushroom” button	Mushroom Manager

Class Name: Delete Mushroom (Boundary)	
Responsibility:	**Collaborators:**
Knows Mushroom Manager Handles click-event of ”Delete Mushroom” button	Mushroom Manager

Class Name: Social Media Manager (Controller)	
Responsibility:	**Collaborators:**
- Receives requests to share mushroom or recipe information from boundary classes.	Mushroom Info
- Retrieves relevant mushroom or recipe details from the database.	Recipe Info
- Formats and forwards content for sharing via supported social media platforms.	Share Recipe
- Manages permissions and restrictions for sharing mushroom and recipe data.	Share Mushroom
- Handles errors and ensures successful sharing operations.	

Class Name: Share Recipe (Boundary)	
Responsibility:	**Collaborators:**
Knows Social Media Manager Handles click-event of "Share Recipe" button	Social Media Manager

Class Name: Share Mushroom (Boundary)	
Responsibility:	**Collaborators:**
Knows Social Media Manager Handles click-event of "Share Mushroom" button	Social Media Manager

A Division of Labour

- Include a Division of Labour sheet which indicates the contributions of each team member. This sheet must be signed by all team members.
- Farid Bastoros:
  - Updated CRC Cards Including:
    - Account Manager
    - Mushroom Manager
    - Social Media Manager
    - Recipe Recommender
    - Decision Fourm
    - Text Description Identifier
    - Macro Image Identifier
    - Micro Image Identifier
  - Section 1 - Reviewed and updated 1.1
  - Section 1 - Reviewed and updated 1.3
  - Section 2 - Completed half of the classes
  - Section 2 - Formatted analysis class diagram in LaTeX
  - Section 4 - Completed Half of the CRC Cards
  - Modified Deliverable 1 with Fixes including:
    - Section 1.3 - Definitions, Acronyms, and Abbreviations
    - Section 1.4 - References
    - Section 4 - Highlight of Functional Requirements BE1
    - Section 5.1 - Look and Feel Requirements
    - Section 5.2 - Usability and Humanity Requirements
    - Section 5.3 - Performance Requirements
    - Section 5.4 - Operational and Environmental Requirements
    - Section 5.5 - Maintainability and Support Requirements
    - Section 5.6 - Security Requirements
    - Section 5.7 - Cultural and Political Requirements
    - Section 5.8 - Legal Requirements
    - Section 6 - Innovative Feature

[illegible]

- Neha Bhatla:
  - Section 1.1 - Completed and reviewed Purpose
  - Section 1.2 - Completed and reviewed System Requirements
  - Section 4 - Comepleted half of the CRC Cards

[illegible]

- Omar Alam:
  - Add BE6 to D1.
  - Section 3.1 - Define architecture and subsystems
  - Section 3.1 - Add Diagrams for the architecture

[illegible]

- Luka Mahrt-Smith:
  - Section 1.3 - Overview
  - Section 2 - Analysis Class Diagram

[illegible]

- Aidan Lao:
  - Section 3.2 - Subsystems

[illegible]