

**Requirement & Design Specification**

**AutomateCrafter - Your Smart Email Customization and Automation Solution**

**Subject: SWD392**

**Version: 1.0**

– Danang, Jan 2025 –

**Record of Changes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date** | **A\* M, D** | **In charge** | **Change Description** |
| V1.0 | Jan 15 | A | HoaiNX |  |
|  |  |  |  |  |
|  |  |  |  |  |

\*A - Added M - Modified D - Deleted

Contents

[I. Requirement Specification 5](#_Toc194005353)

[I.1 Problem description 5](#_Toc194005354)

[Organizations face significant challenges managing email communications and automating routine processes. Handling multiple email accounts, creating personalized content, and implementing automation workflows requires specialized expertise and consumes valuable time. AutomateCrafter addresses these challenges by providing a unified platform that combines email management with Python-based process automation capabilities. 5](#_Toc194005355)

[I.2 Major Features 5](#_Toc194005356)

[**I.2.1 Features for customers are as follows** 5](#_Toc194005357)

[I.2.2 **Features for admin are as follows** 5](#_Toc194005366)

[ Admin Dashboard: Comprehensive system management, user administration, and reporting. 5](#_Toc194005367)

[ Bot Package Management: Create, test, and deploy automation packages to customer environments. 5](#_Toc194005368)

[ System Monitoring: Monitor performance metrics, resource usage, and job execution status. 5](#_Toc194005369)

[I.3 System context 5](#_Toc194005370)

[I.4 Nonfunctional Requirements 5](#_Toc194005371)

[I.5 Functional requirements 6](#_Toc194005372)

[**I.5.1 Use case diagrams** 7](#_Toc194005373)

[**I.5.2 Use case descriptions** 7](#_Toc194005374)

[**I.5.3 Activity diagram** 18](#_Toc194005375)

[I.6 Data Requirements 20](#_Toc194005376)

[II. Analysis models 20](#_Toc194005377)

[II.1 Interaction diagrams 20](#_Toc194005378)

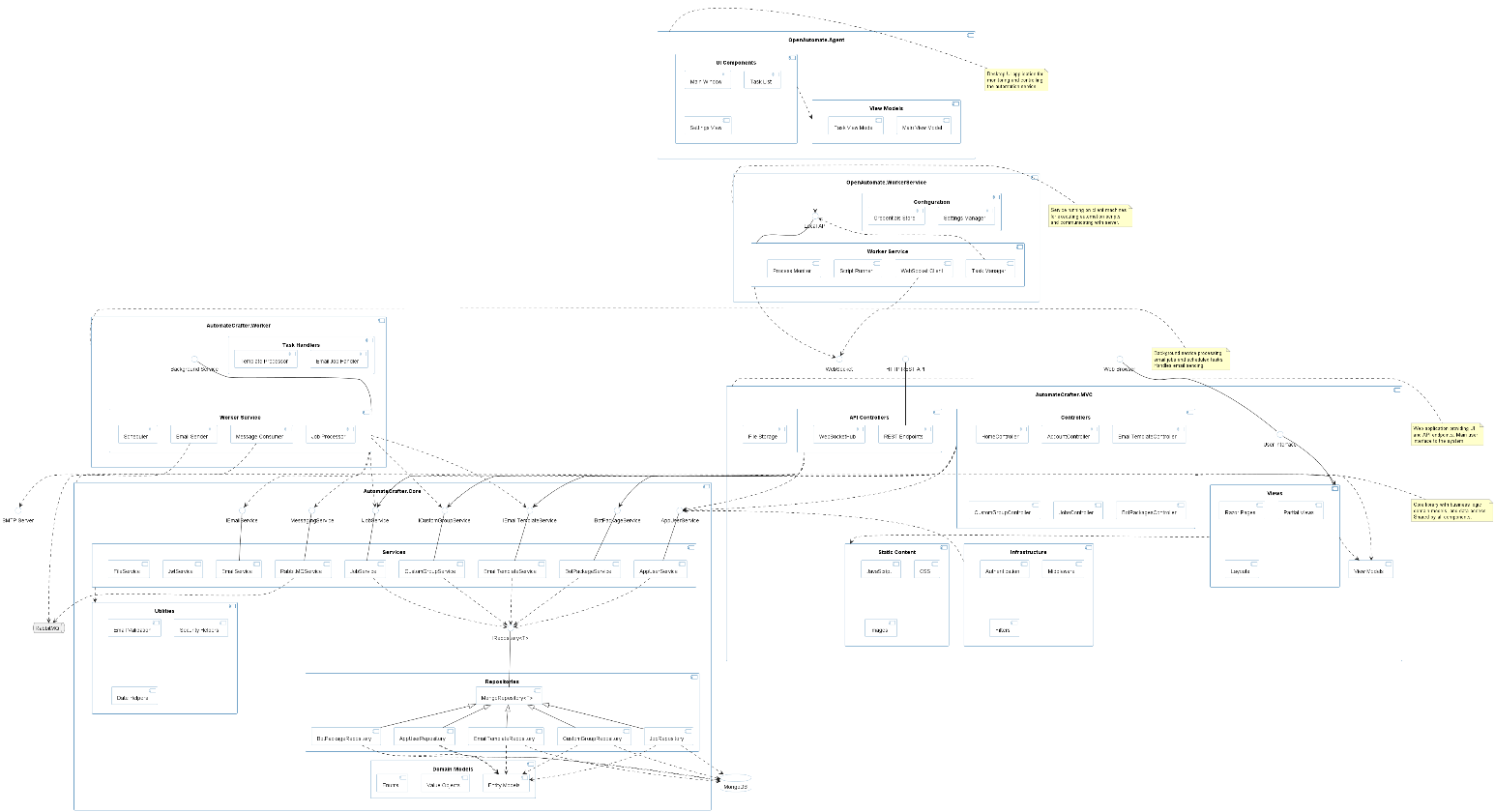
[II.2 State diagram 26](#_Toc194005379)

[III. Design specification 30](#_Toc194005380)

[III.1 Communication Diagrams 30](#_Toc194005381)

[III.2 System High-Level Design 31](#_Toc194005382)

[III.3 Component and Package Diagram 31](#_Toc194005383)

[ 32](#_Toc194005384)

[III.4 Detail Design 32](#_Toc194005385)

[III.5 Database Design 32](#_Toc194005386)

[IV. Implementation 33](#_Toc194005387)

[IV.1 Map architecture to the structure of the project 33](#_Toc194005388)

[V. Applying Design Pattern 34](#_Toc194005389)

# I. Requirement Specification

## I.1 Problem description

## Organizations face significant challenges managing email communications and automating routine processes. Handling multiple email accounts, creating personalized content, and implementing automation workflows requires specialized expertise and consumes valuable time. AutomateCrafter addresses these challenges by providing a unified platform that combines email management with Python-based process automation capabilities.

## I.2 Major Features

### **I.2.1 Features for customers are as follows**

### Multi-Email Integration: Centralized management of multiple email accounts in one dashboard.

### Customizable Email Templates: Create and save personalized templates with dynamic placeholders and conditional content.

### Custom Group Management: Organize recipients with tailored properties for personalized communication.

### Excel File Import: Upload and parse Excel files to create or update contact groups efficiently.

### Property-Based Personalization: Leverage recipient attributes for highly personalized email content.

### Recurring Email Campaigns: Schedule and automate regular email communications.

### Python-Based Automation: Create and deploy custom Python automation scripts to handle complex workflows.

### Robot Agent Management: Deploy and monitor automation bots on dedicated machines.

### I.2.2 **Features for admin are as follows**

## Admin Dashboard: Comprehensive system management, user administration, and reporting.

## Bot Package Management: Create, test, and deploy automation packages to customer environments.

## System Monitoring: Monitor performance metrics, resource usage, and job execution status.

## I.3 System context

AutomateCrafter functions as an integrated web application with distributed worker components. The core platform manages email services and provides the automation control center, while dedicated robot agents execute automation tasks on target machines. The platform integrates with email providers, databases, and various business applications through APIs and custom connectors.

## I.4 Nonfunctional Requirements

* **Performance**: Performance: Support processing of 10,000+ emails per hour and concurrent execution of 100+ automation tasks.
* **Scalability**: Horizontal scaling capabilities for both email processing and automation tasks.
* **Security**: End-to-end encryption, secure credential management, and role-based access controls.
* **Usability**: Intuitive interfaces for both email management and automation configuration.
* **Reliability**: 99.9% uptime for web services and robust error handling for automation tasks.
* **Cross-platform Support**: Robot agents compatible with Windows, Linux, and macOS environments.

## I.5 Functional requirements

**Email Management**

* Secure integration with multiple email providers (SMTP/IMAP/API-based)
* Automated token refresh and secure credential storage
* Email signature and template management
* Comprehensive email sending capabilities with attachments and formatting
* Email response tracking and analytics

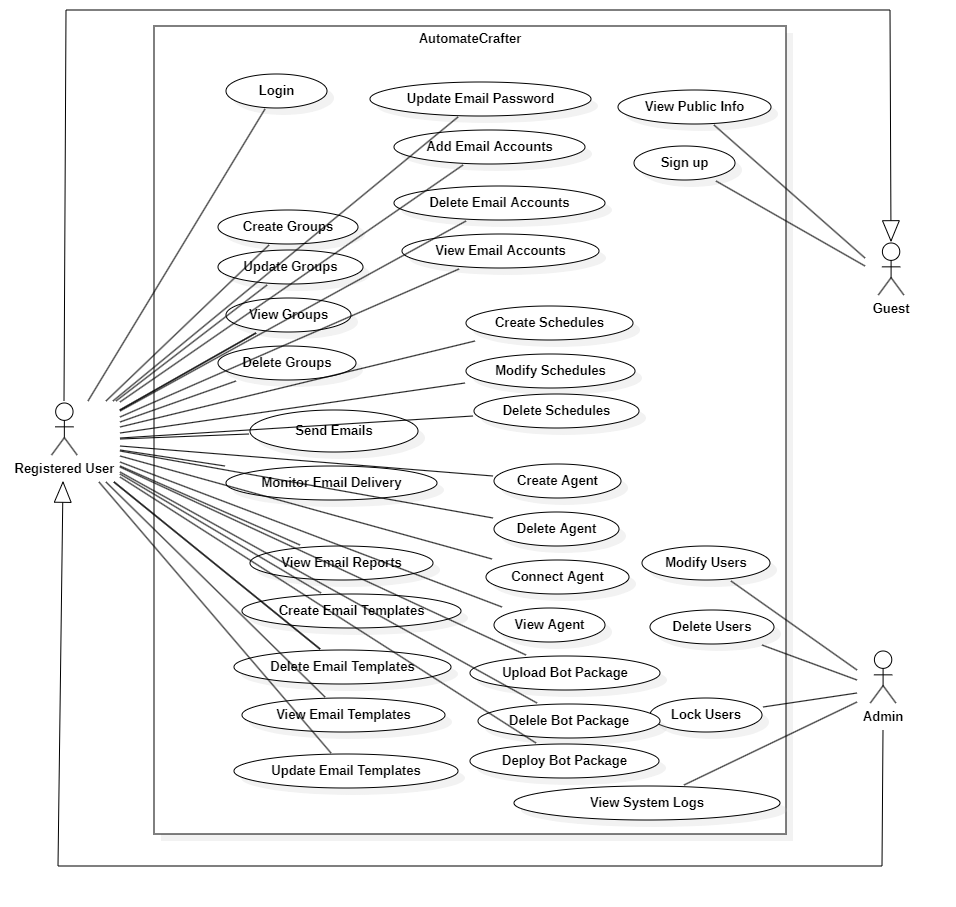
**Automation System**

* Python script execution environment
* Visual workflow designer with predefined components
* Scheduled and triggered automation execution
* Cross-application data transfer capabilities
* Error handling and automated recovery procedures
* Detailed execution logs and audit trails

**Agent Management**

* Remote deployment of automation packages
* Real-time monitoring of agent status
* Resource usage tracking and optimization
* Secure communication between central server and agents
* Agent auto-update mechanism

### **I.5.1 Use case diagrams**



### **I.5.2 Use case descriptions**

*Table I-1 Use case description for managing* **Sign up**

|  |  |  |  |
| --- | --- | --- | --- |
| ID and Name: | **UC-01 Sign up** | | |
| Created By: | HoaiNX | Date Created: | 18/Feb/2025 |
| Primary Actor: | Guest | Secondary Actors: |  |
| Description: | This use case allows guests to register and create a new account in the system. | | |
| Trigger: | The guest selects the "Sign up" option from the system menu. | | |
| Preconditions: | PRE-1. The guest has access to the system. | | |
| Postconditions: | POST-1. The guest account is successfully created and stored in the system. | | |
| Normal Flow: | 1. The guest selects the "Sign up" option. 2. The system displays the registration form. 3. The guest enters required details (e.g., name, email, password). 4. The guest submits the form. 5. The system validates the input. 6. If valid, the system creates an account and sends a confirmation email. 7. The system displays a success message. | | |
| Alternative Flows: | If the guest enters invalid details, the system displays an error message and prompts them to correct the input. | | |
| Exceptions: | If the email is already registered, the system notifies the guest and suggests logging in.  If the system encounters an error, an appropriate message is displayed. | | |
| Priority: | High | | |
| Business Rules: | A valid email and password must be provided.  Password must meet security requirements. | | |

#### Table I-2 Use case description for **Login**

|  |  |  |  |
| --- | --- | --- | --- |
| ID and Name: | **UC-02 Login** | | |
| Created By: | HoaiNX | Date Created: | 18/Feb/2025 |
| Primary Actor: | User | Secondary Actors: |  |
| Description: | This use case allows a registered user to log into the system using their credentials. | | |
| Trigger: | PRE-1. The user has a registered account in the system.  PRE-2. The user has access to the login page. | | |
| Preconditions: | PRE-1. Warehouse staff is logged into FSO.  PRE-2. Warehouse staff is authorized to add products. | | |
| Postconditions: | POST-1. The user is successfully authenticated and gains access to the system. | | |
| Normal Flow: | 1. The user selects the "Login" option. 2. The system displays the login form. 3. The user enters their email and password. 4. The user submits the form. 5. The system verifies the credentials. 6. If valid, the system grants access and redirects the user to the dashboard. | | |
| Alternative Flows: | If the user enters incorrect credentials, the system displays an error message and allows retries. | | |
| Exceptions: | If the user forgets their password, they can select "Forgot Password" to reset it.  If the account is locked due to multiple failed attempts, the system notifies the user and provides recovery options. | | |
|  | 1. The system logs the error and displays an error message to the WS. | | |
| Priority: | High | | |
| Business Rules: | The system must enforce strong authentication policies.  The system should log all login attempts for security monitoring. | | |

*Table I-3 Use case description for* **Manage Email Accounts**

|  |  |  |  |
| --- | --- | --- | --- |
| ID and Name: | **UC-03 Manage Email Accounts** | | |
| Created By: | HoaiNX | Date Created: | 18/Feb/2025 |
| Primary Actor: | User | Secondary Actors: |  |
| Description: | This use case allows users to manage their email accounts, including viewing, adding, and deleting them. | | |
| Trigger: | The user selects the "Manage Email Accounts" option from the system menu. | | |
| Preconditions: | PRE-1. The user is logged into the system. | | |
| Postconditions: | POST-1. Email accounts are successfully managed based on user actions. | | |
| Normal Flow: | 1. The user selects the "Manage Email Accounts" option. 2. The system retrieves the list of linked email accounts. 3. The system displays the "Manage Email Accounts" interface. 4. The user can perform the following actions: 5. View Email Accounts 6. Add Email Accounts 7. Delete Email Accounts 8. The system processes the user's action accordingly. | | |
| Alternative Flows: | If the user attempts to add an already linked email account, the system displays an error message. | | |
| Exceptions: | If the system encounters an issue retrieving email accounts, an error message is displayed. | | |
| Priority: | High | | |
| Business Rules: | Users can only manage email accounts they have linked to their profile.  The system must validate email formats when adding a new account. | | |

*Table I-4 Use case description for* **View Email Accounts**

|  |  |  |  |
| --- | --- | --- | --- |
| ID and Name: | **UC-04 View Email Accounts** | | |
| Created By: | HoaiNX | Date Created: | 18/Feb/2025 |
| Primary Actor: | User | Secondary Actors: |  |
| Description: | This use case allows users to view all linked email accounts in the system. | | |
| Trigger: | The user selects the "View Email Accounts" option from the system menu. | | |
| Preconditions: | PRE-1. The user is logged into the system. | | |
| Postconditions: | POST-1. The system displays all email accounts linked to the user's profile. | | |
| Normal Flow: | 1. The user selects the "View Email Accounts" option. 2. The system retrieves the list of linked email accounts. 3. The system displays the email accounts on the user interface. | | |
| Alternative Flows: | If the user has no linked email accounts, the system displays a message indicating no accounts are available. | | |
| Exceptions: | If the system fails to retrieve the email accounts, an error message is displayed. | | |
| Priority: | Medium | | |
| Business Rules: | Users can only view email accounts that are linked to their profile.  The system should ensure secure retrieval of email account information. | | |

*Table I-5 Use case description for Add Email Accounts*

|  |  |  |  |
| --- | --- | --- | --- |
| ID and Name: | **UC-05 Add Email Accounts** | | |
| Created By: | HoaiNX | Date Created: | 18/Feb/2025 |
| Primary Actor: | User | Secondary Actors: |  |
| Description: | This use case allows users to add new email accounts to the system. | | |
| Trigger: | The user selects the "Add Email Account" option from the system menu. | | |
| Preconditions: | PRE-1. The user is logged into the system. | | |
| Postconditions: | POST-1. A new email account is successfully linked to the user's profile. | | |
| Normal Flow: | 1. The user selects the "Add Email Account" option. 2. The system displays a form for entering the email account details. 3. The user enters the required information (e.g., email address, authentication details). 4. The user submits the form. 5. The system validates the input. 6. If valid, the system adds the email account and updates the user's profile. 7. The system displays a confirmation message. | | |
| Alternative Flows: | If the email account is already linked, the system notifies the user and prevents duplication. | | |
| Exceptions: | If the system encounters an issue while adding the email account, an error message is displayed. | | |
| Priority: | Medium | | |
| Business Rules: | The system must validate the email format before linking the account.  Users can only add email accounts they have access to. | | |

*Table I-6 Use case description for Delete Email Accounts*

|  |  |  |  |
| --- | --- | --- | --- |
| ID and Name: | **UC-06 Delete Email Accounts** | | |
| Created By: | HoaiNX | Date Created: | 18/Feb/2025 |
| Primary Actor: | User | Secondary Actors: |  |
| Description: | This use case allows users to remove unnecessary email accounts from the system | | |
| Trigger: | The user selects the "Delete Email Account" option from the system menu. | | |
| Preconditions: | PRE-1. The user is logged into the system.  PRE-2. The user has at least one linked email account. | | |
| Postconditions: | POST-1. The selected email account is successfully removed from the system. | | |
| Normal Flow: | 1. The user selects the "Delete Email Account" option. 2. The system retrieves the list of linked email accounts. 3. The system displays the list of email accounts. 4. The user selects an email account to delete. 5. The system prompts the user for confirmation. 6. The user confirms the deletion. 7. The system removes the selected email account from the profile. 8. The system displays a success message. | | |
| Alternative Flows: | If the user cancels the deletion, no changes are made. | | |
| Exceptions: | If the user attempts to delete an email account that is required for system operations, the system prevents the deletion and notifies the user.  If the system encounters an error during deletion, an error message is displayed. | | |
| Priority: | Medium | | |
| Business Rules: | Users can only delete email accounts that belong to their profile.  The system must verify that at least one primary email remains linked to the account if required. | | |

*Table I-7 Use case description for* **Manage Email Templates**

|  |  |  |  |
| --- | --- | --- | --- |
| ID and Name: | **UC-07 Manage Email Templates** | | |
| Created By: | HoaiNX | Date Created: | 18/Feb/2025 |
| Primary Actor: | User | Secondary Actors: |  |
| Description: | This use case allows users to manage email templates by creating, editing, or deleting them. | | |
| Trigger: | The user selects the "Manage Email Templates" option from the system menu. | | |
| Preconditions: | PRE-1. The user is logged into the system. | | |
| Postconditions: | POST-1. Email templates are successfully created, edited, or deleted as requested. | | |
| Normal Flow: | 1. The user selects the "Manage Email Templates" option. 2. The system retrieves the list of existing email templates. 3. The system displays the "Manage Email Templates" interface. 4. The user can perform the following actions:  * View Email Templates * Add Email Templates * Edit Email Templates * Delete Email Templates  1. The system processes the user's action accordingly. | | |
| Alternative Flows: | If the user attempts to edit or delete a template that does not exist, the system notifies the user. | | |
| Exceptions: | If the system encounters an issue retrieving or modifying templates, an error message is displayed. | | |
| Priority: | Medium | | |
| Business Rules: | Users can only manage templates that they have created or have permission to edit.  The system must validate the template content before saving changes. | | |

*Table I-8 Use case description for* **View Email Templates**

|  |  |  |  |
| --- | --- | --- | --- |
| ID and Name: | **UC-08 View Email Templates** | | |
| Created By: | HoaiNX | Date Created: | 18/Feb/2025 |
| Primary Actor: | User | Secondary Actors: |  |
| Description: | This use case allows users to browse existing email templates. | | |
| Trigger: | The user selects the "View Email Templates" option from the system menu. | | |
| Preconditions: | PRE-1. The user is logged into the system. | | |
| Postconditions: | POST-1. Email templates are successfully created, edited, or deleted as requested. | | |
| Normal Flow: | 1. The user selects the "View Email Templates" option. 2. The system retrieves the list of available email templates. 3. The system displays the list of email templates to the user. | | |
| Alternative Flows: | If no email templates are available, the system displays a message indicating that no templates exist. | | |
| Exceptions: | If the system fails to retrieve email templates, an error message is displayed. | | |
| Priority: | Low | | |
| Business Rules: | Users can only view email templates they have access to.  The system should ensure secure retrieval of template data. | | |

### **I.5.3 Activity diagram**

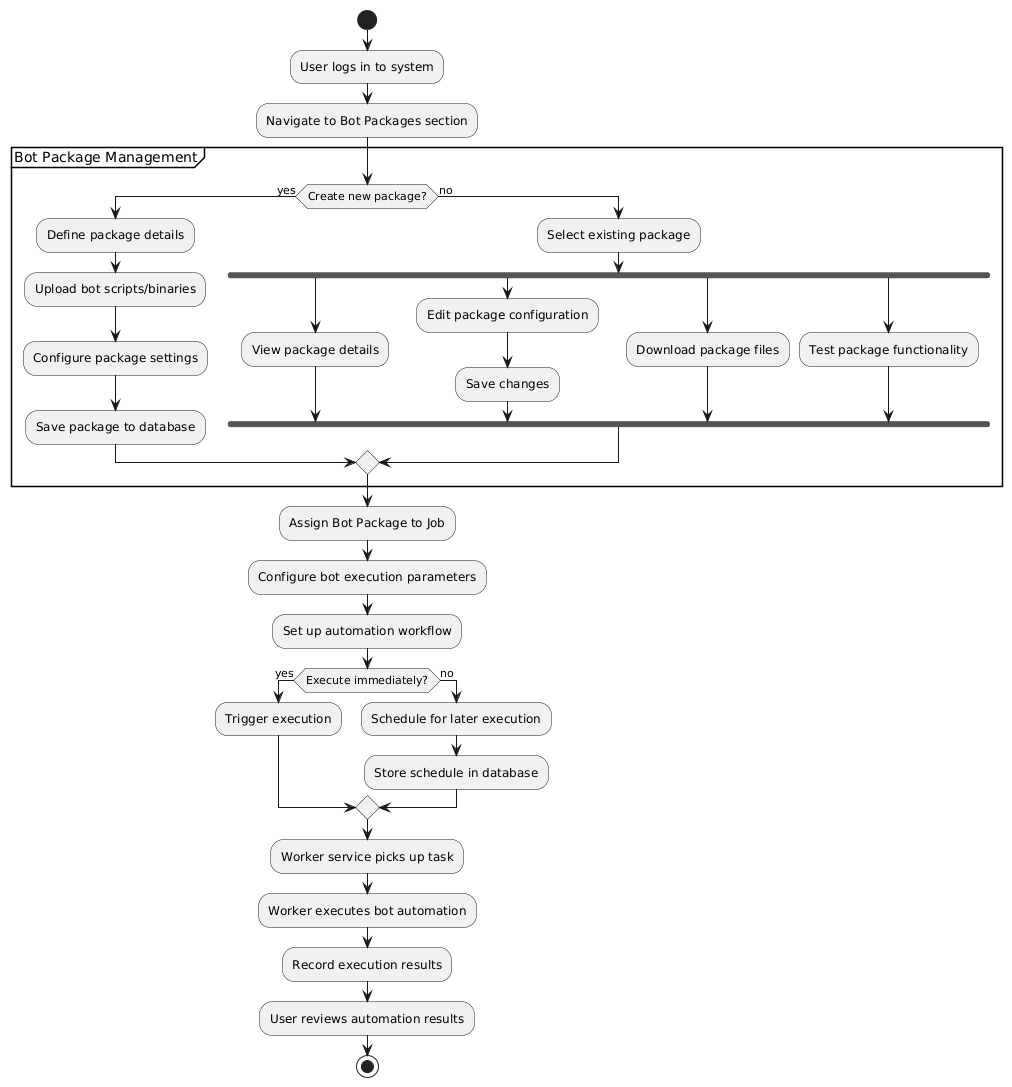


Figure . Bot Package Management Activity Diagram

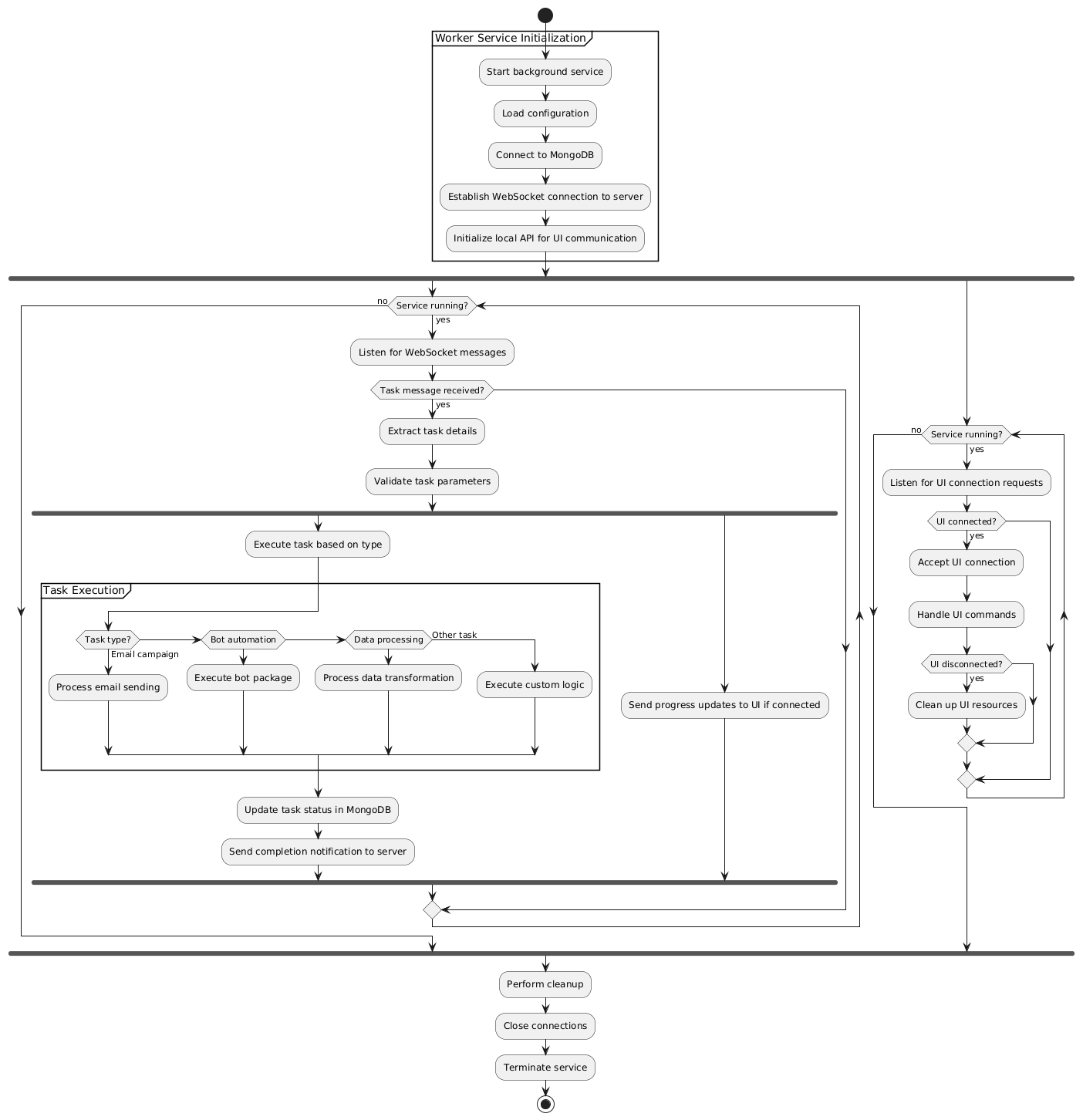


Figure . Worker Service Activity Diagram

## I.6 Data Requirements

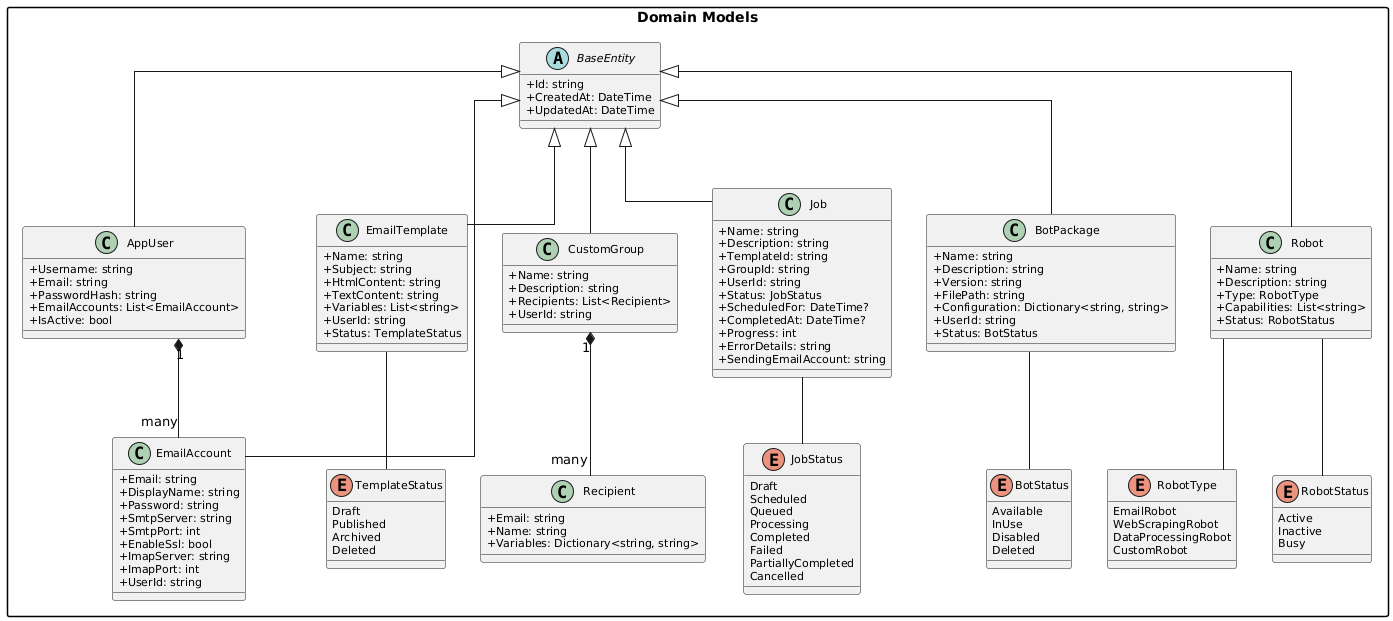


Figure . Domain Models Class Diagram

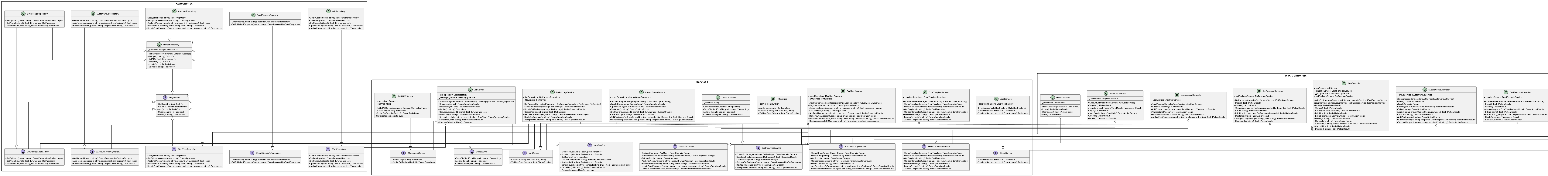
******

Figure . Repositories - Services - Controllers Class Diagram

# II. Analysis models

## II.1 Interaction diagrams

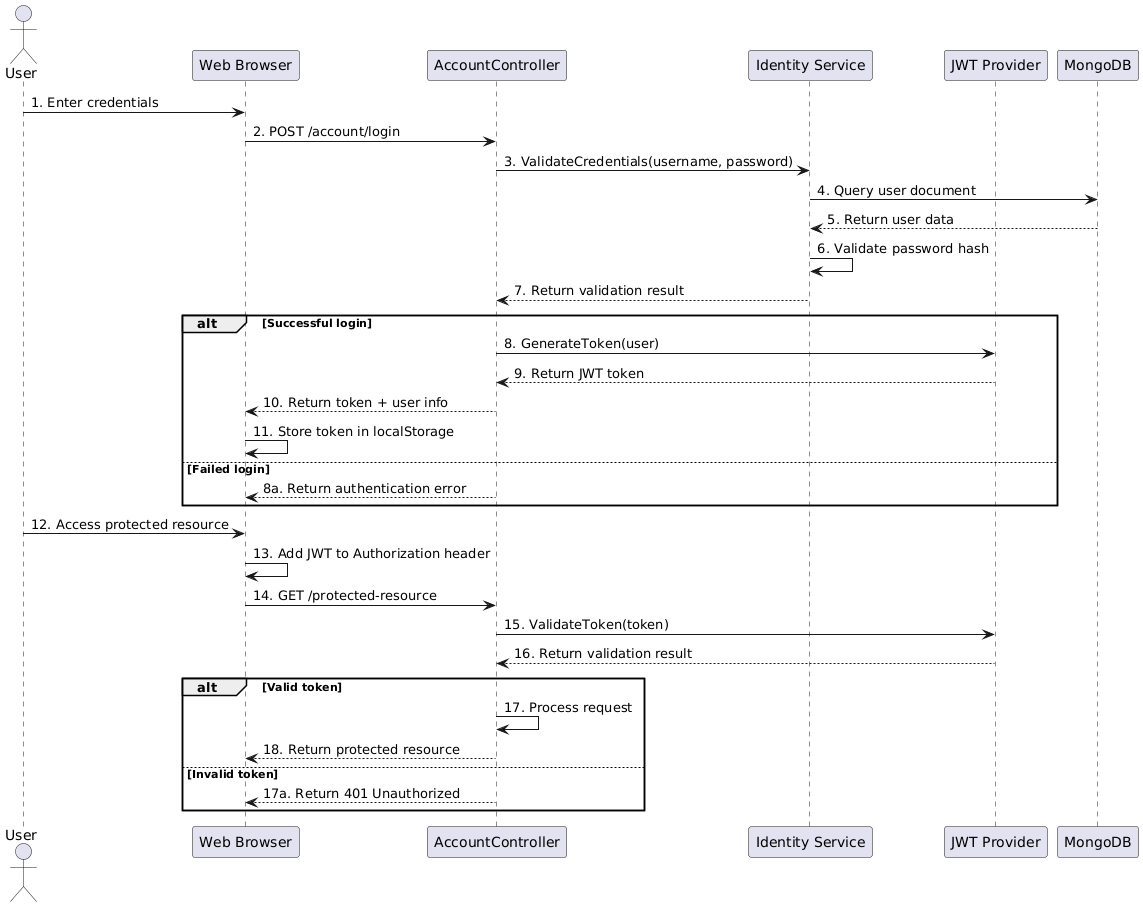


Figure . User Authentication Sequence Diagram

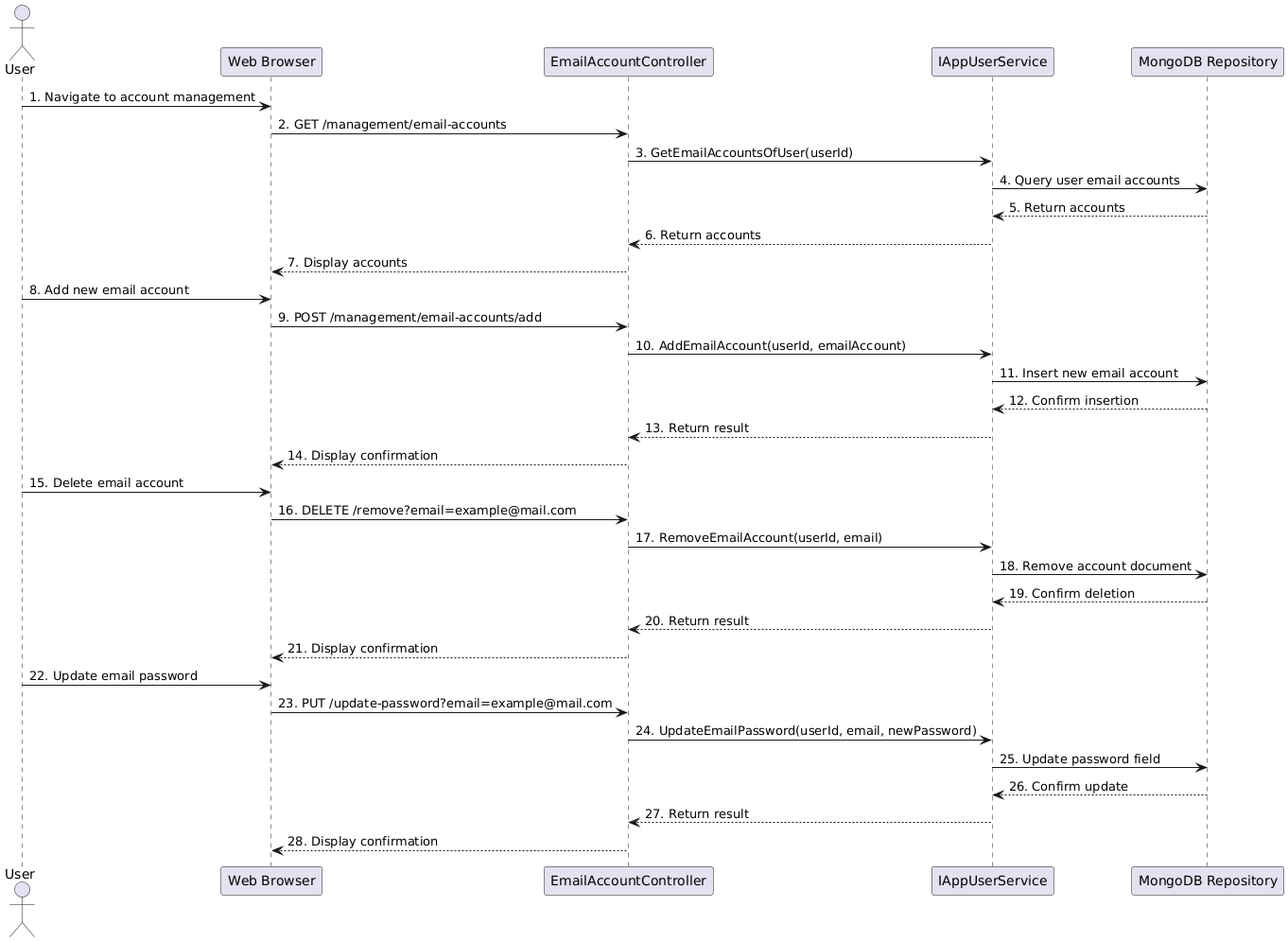


Figure . Email Account Management Sequence Diagram

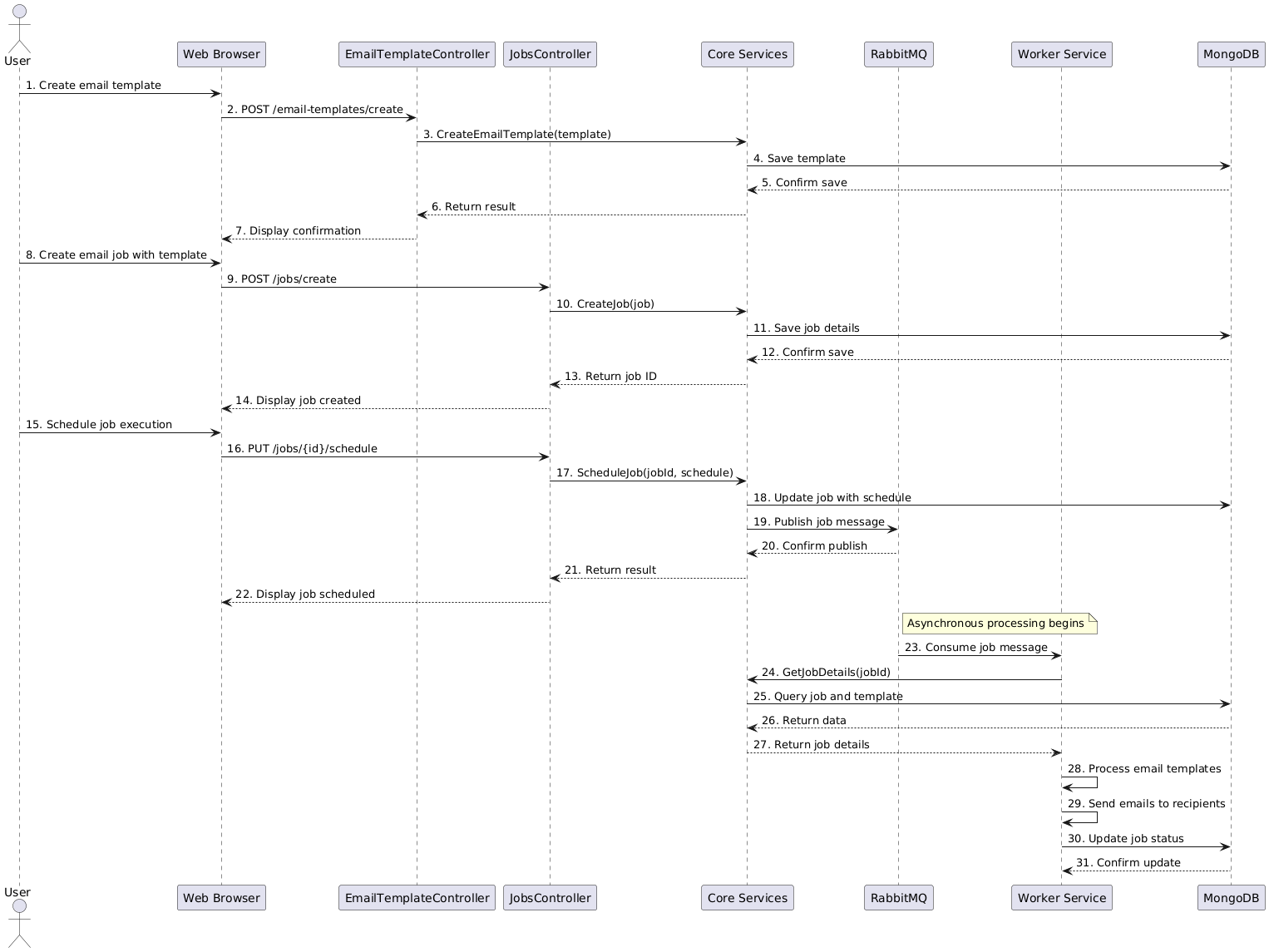


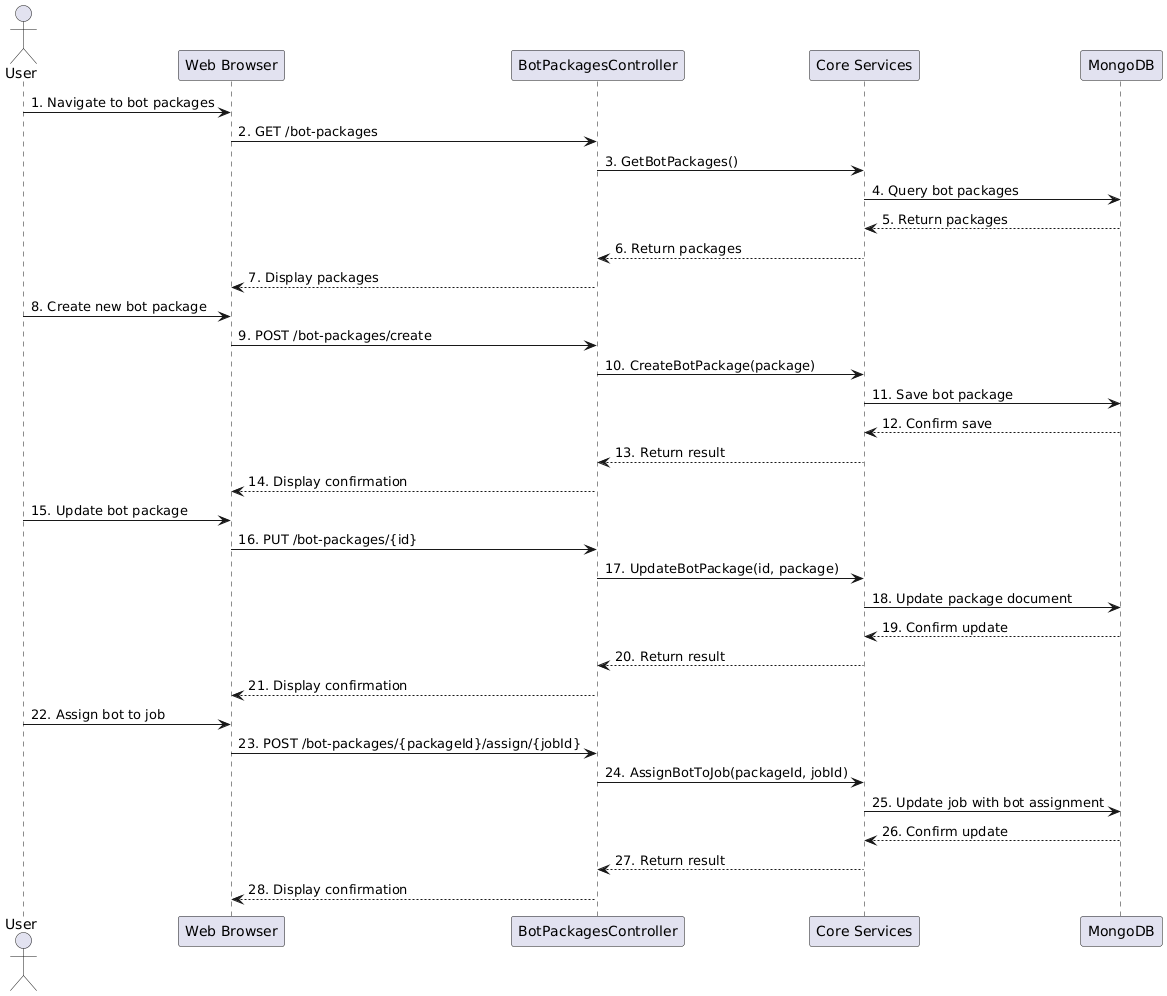
Figure . Email Campaign Creation and Scheduling Sequence Diagram

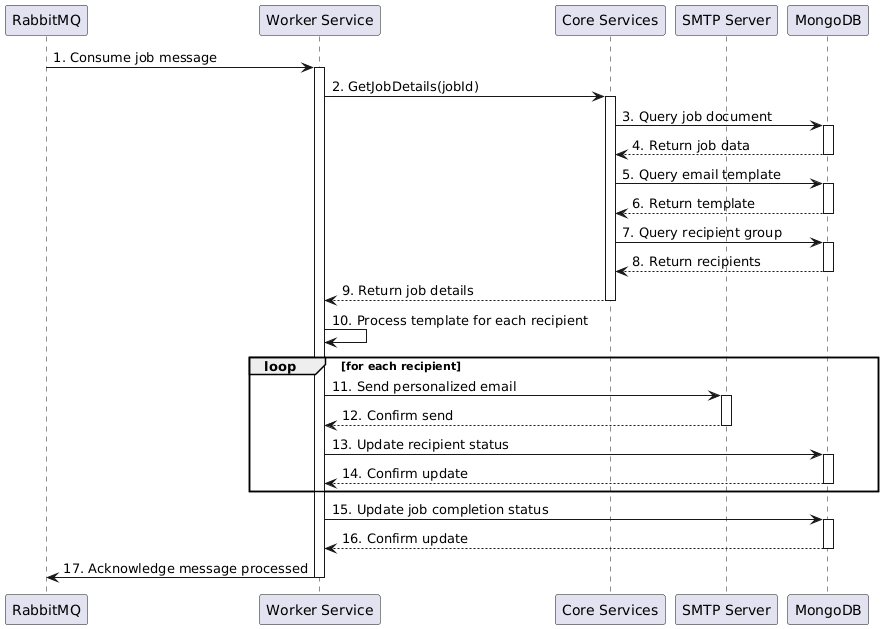
Figure . Bot Package Management Sequence Diagram

Figure . Worker Service Job Processing Sequence Diagram

## II.2 State diagram

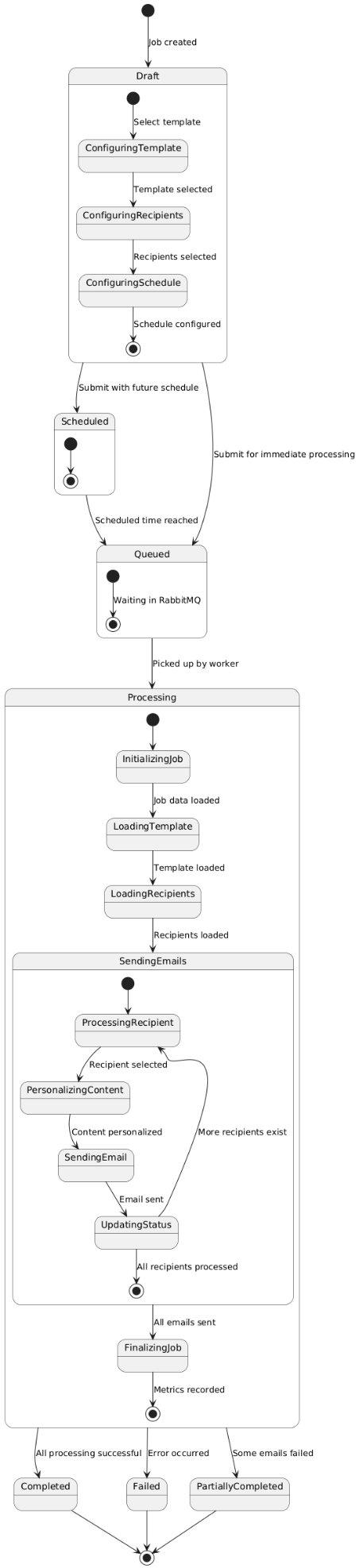
**

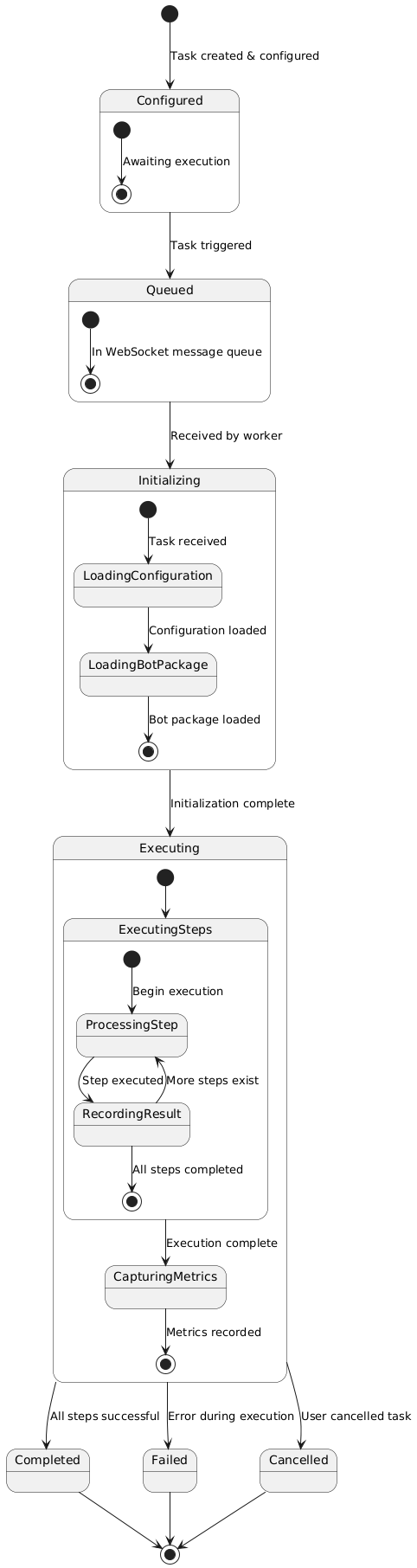
Figure . Email Job State Diagram

Figure . Bot Task State Diagram

# III. Design specification

## III.1 Communication Diagrams

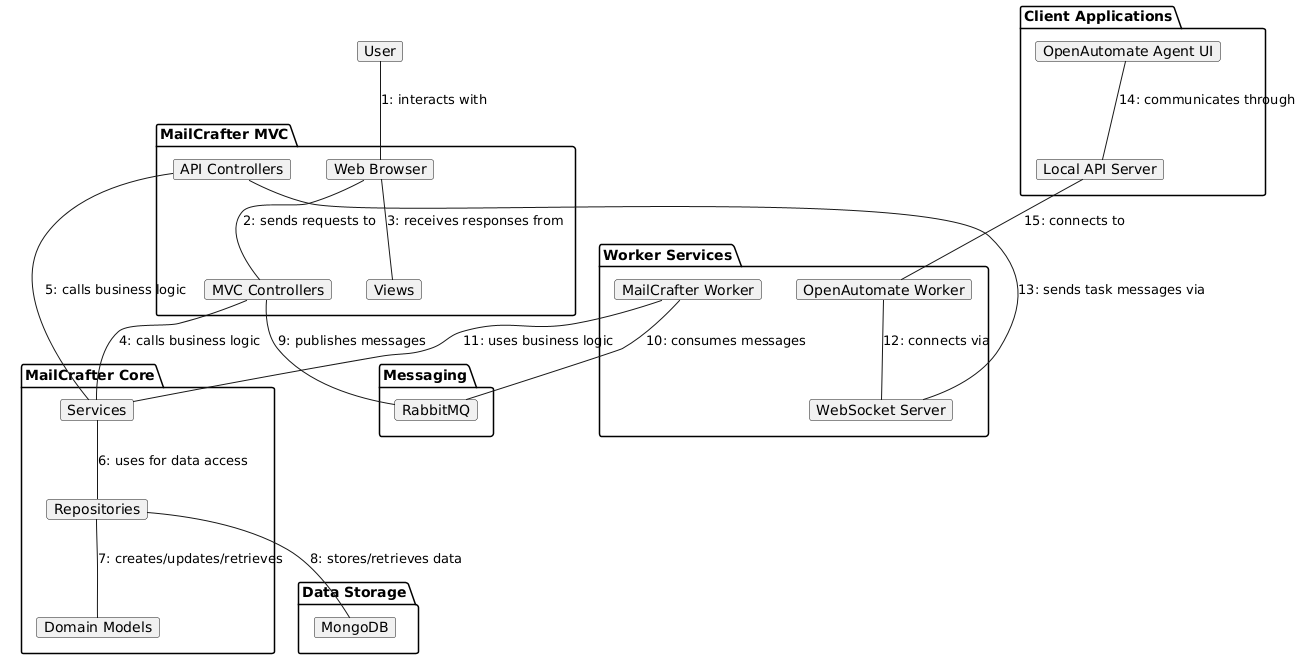


Figure . Core Communication Diagram

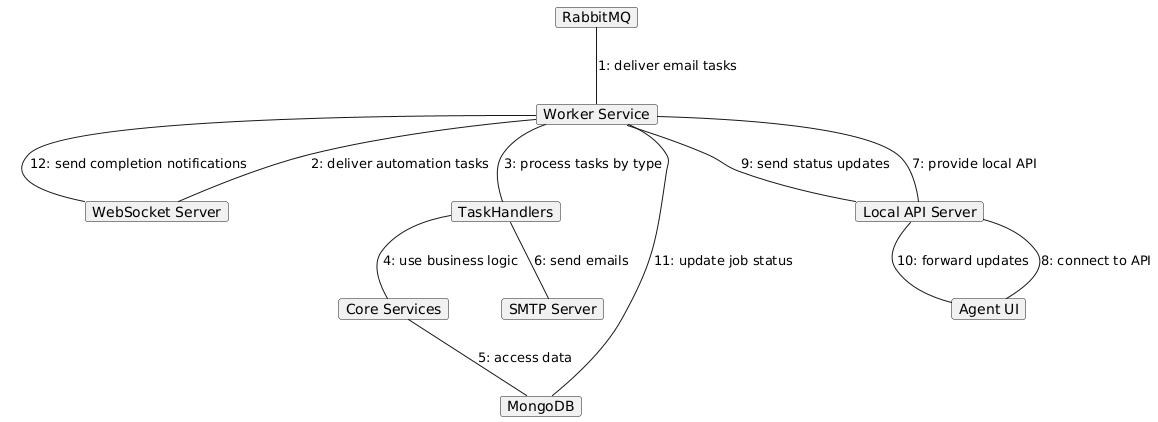


Figure . Worker Service Communication Diagram.

## III.2 System High-Level Design

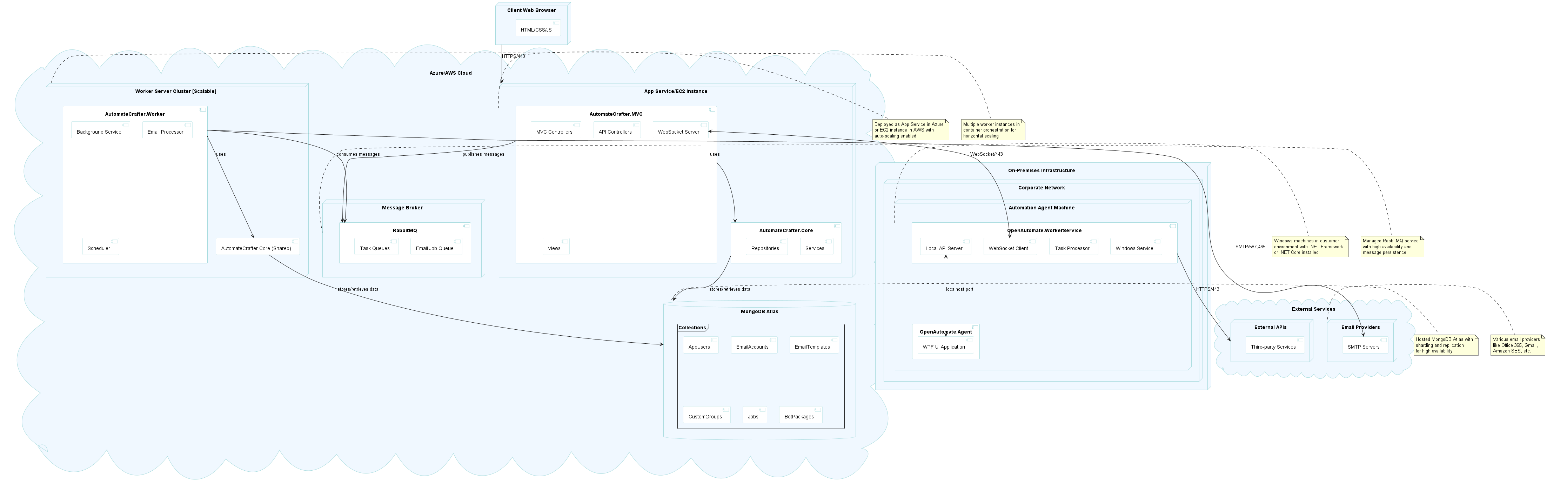


Figure . Deployment Diagram

## III.3 Component and Package Diagram

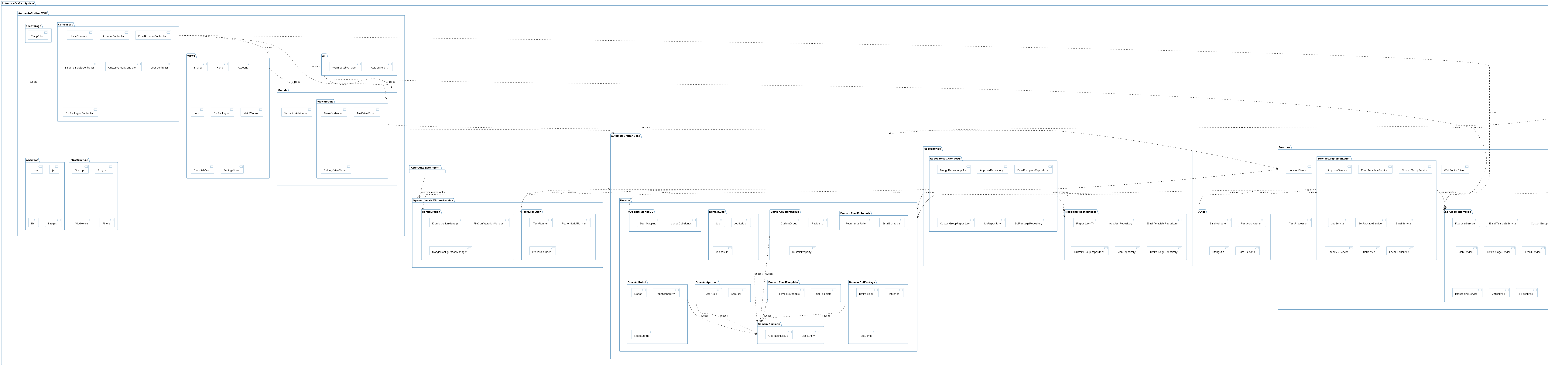


Figure . Package Diagram

## 

Figure . Component Diagram

## III.4 Detail Design

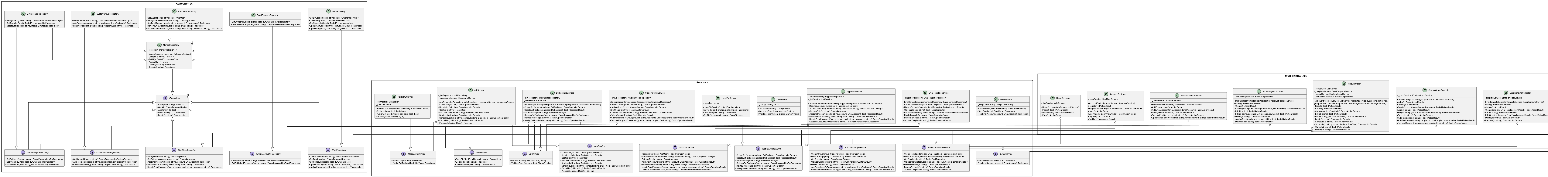


Figure . Repositories - Controllers - Services Class Diagram

## III.5 Database Design

# IV. Implementation

## IV.1 Map architecture to the structure of the project

A screenshot of a computer program

AI-generated content may be incorrect.

Figure . The structure of the project

# V. Applying Design Pattern

#### The AutomateCrafter system employs several design patterns to ensure maintainability, extensibility, and separation of concerns. Here's an analysis of the key design patterns used throughout the project:

#### **1. Architectural Patterns**

#### **Repository Pattern**

#### **Implementation**: In MailCrafter.Core/Repositories

#### Generic IRepository<T> interface defines standard CRUD operations

#### MongoRepository<T> provides implementation for MongoDB

#### Specialized repositories like EmailTemplateRepository extend the base repository

#### **Benefits**:

#### Abstracts data access logic from business services

#### Enables swapping database implementations without affecting business logic

#### Centralizes data access code and query logic

#### Facilitates unit testing through mock repositories

#### **Dependency Injection Pattern**

#### **Implementation**: Used throughout the system

#### Services registered in Startup.cs/Program.cs with appropriate lifetimes

#### Controllers, services, and workers receive dependencies via constructor injection

#### Interfaces used to decouple concrete implementations

#### **Benefits**:

#### Loose coupling between components

#### Easier unit testing through mock dependencies

#### Centralized configuration of service lifetimes

#### Simplified management of cross-cutting concerns

#### **MVC Pattern**

#### **Implementation**: Primary pattern in MailCrafter.MVC

#### Controllers handle HTTP requests and application flow

#### Models represent data structures and validation

#### Views render the UI representation

#### **Benefits**:

#### Clear separation of concerns

#### Organized code structure

#### Easier maintenance and testing

#### Follows ASP.NET Core's built-in conventions

#### **2. Creational Patterns**

#### **Factory Method Pattern**

#### **Implementation**: Used in service layers

#### SmtpClientFactory for creating SMTP clients

#### BotTaskFactory for creating different types of automation tasks

#### **Benefits**:

#### Encapsulates object creation logic

#### Allows for runtime decision about which concrete class to instantiate

#### Centralizes instantiation code

#### **Builder Pattern**

#### **Implementation**: Used for complex object construction

#### EmailBuilder for constructing email messages

#### BotConfigurationBuilder for building automation configs

#### **Benefits**:

#### Step-by-step construction of complex objects

#### Fluent interface for improved readability

#### Separation of construction from representation

#### Enables different representations of the same object

#### **3. Structural Patterns**

#### **Adapter Pattern**

#### **Implementation**: Used for integrating external systems

#### SmtpClientAdapter adapts different email providers to a common interface

#### ExternalApiAdapter provides uniform access to various external APIs

#### **Benefits**:

#### Makes incompatible interfaces work together

#### Enables integration with third-party systems without modifying their code

#### Provides a common interface for similar functionality

#### **Decorator Pattern**

#### **Implementation**: Used for adding functionality to existing components

#### LoggingJobProcessor adds logging to job processing

#### RetryEmailSender adds retry logic to email sending

#### **Benefits**:

#### Dynamically adds responsibilities to objects

#### Provides an alternative to subclassing for extending functionality

#### Follows single responsibility principle

#### Enables combining multiple behaviors

#### **Facade Pattern**

#### **Implementation**: Simplifies complex subsystems

#### EmailService provides a simplified interface to email operations

#### BotService abstracts complex automation operations

#### **Benefits**:

#### Provides a unified interface to a set of interfaces in a subsystem

#### Hides complexity of subsystems from clients

#### Reduces dependencies between clients and subsystems

#### **4. Behavioral Patterns**

#### **Observer Pattern**

#### **Implementation**: Used for event handling and notifications

#### JobProgressTracker notifies observers of job progress

#### WebSocket notifications for real-time updates

#### **Benefits**:

#### Supports one-to-many dependency between objects

#### Notifies dependent objects when state changes

#### Enables loose coupling between components

#### **Strategy Pattern**

#### **Implementation**: Used for algorithm selection

#### EmailPersonalizationStrategy for different template processing approaches

#### AuthenticationStrategy for different auth methods

#### **Benefits**:

#### Defines a family of algorithms, encapsulates each one, and makes them interchangeable

#### Allows algorithm selection at runtime

#### Eliminates conditional statements in client code

#### **Command Pattern**

#### **Implementation**: Used for operation encapsulation

#### SendEmailCommand encapsulates email sending operation

#### BotTaskCommand represents executable automation tasks

#### **Benefits**:

#### Encapsulates a request as an object

#### Allows parameterization of clients with different requests

#### Supports undoable operations

#### Enables queueing of requests

#### **State Pattern**

#### **Implementation**: Used for job and task status management

#### JobState and derived states manage job lifecycle

#### TaskState for automation task states

#### **Benefits**:

#### Allows an object to alter its behavior when its internal state changes

#### Encapsulates state-specific behavior in separate classes

#### Simplifies state transition logic

#### **5. Concurrency Patterns**

#### **Producer-Consumer Pattern**

#### **Implementation**: Used in worker services

#### RabbitMQ consumers process messages from queues

#### Background services process tasks asynchronously

#### **Benefits**:

#### Decouples producers and consumers

#### Manages workload distribution

#### Handles backpressure through the queue

#### **Task-based Asynchronous Pattern (TAP)**

#### **Implementation**: Used throughout the codebase

#### Async/await for non-blocking operations

#### Task-based API design

#### **Benefits**:

#### Non-blocking operations for improved scalability

#### Simplified asynchronous code

#### Better resource utilization

#### **6. Domain-Driven Design Patterns**

#### **Aggregate Root Pattern**

#### **Implementation**: Used in domain models

#### AppUser as aggregate root containing EmailAccount entities

#### CustomGroup containing Recipient entities

#### **Benefits**:

#### Ensures data consistency within aggregate boundaries

#### Simplifies transaction management

#### Clarifies ownership and relationships

#### **Value Object Pattern**

#### **Implementation**: Used for immutable data structures

#### EmailAddress as a value object with validation

#### RecurrencePattern for scheduling information

#### **Benefits**:

#### Represents concepts with no identity

#### Ensures immutability

#### Encapsulates validation logic

#### **7. Integration Patterns**

#### **Message Queue Pattern**

#### **Implementation**: Used for asynchronous communication

#### RabbitMQ for job distribution

#### Queue-based processing for email campaigns

#### **Benefits**:

#### Decouples components

#### Enables asynchronous processing

#### Provides load balancing and buffer capacity

#### Ensures reliable message delivery

#### **Circuit Breaker Pattern**

#### **Implementation**: Used for external service calls

#### Handles transient failures when calling external APIs

#### Prevents cascading failures

#### **Benefits**:

#### Prevents repeated calls to failing services

#### Enables graceful degradation

#### Supports self-healing systems