Software Component Cataloguing Software
Documentation

Table of Contents

Problem Statement

Software Component Cataloguing Software

The software component cataloguing software consists of a software components catalogue and various functions defined on this components catalogue. The software components catalogue should hold details of the components which are potentially reusable. The reusable components can be either design or code. The design might have been constructed using different design notations such as UML, ERD, structured design, etc. Similarly, the code might have been written using different programming languages. A cataloguer may enter components in the catalogue, may delete components from the catalogue, and may reuse information with a catalogue component. In order to help manage the component catalogue, the cataloguing software should maintain the information such as how many times a component has been used, and how many times the component has come up in a query but not used. Since the number of components usually tend to be very high, it is desirable to be able to classify the different types of components hierarchically. A user should be able to browse the components in each category.

Software Requirement Specification (SRS) Document

Table of Contents of the SRS Document

1. Introduction

1.1 Purpose

The purpose of the Software Component Cataloguing Software is to provide an organized, efficient system for managing and reusing software components. This software aims to create a centralized catalogue that stores both design and code components, allowing users to search, update, and manage components effectively. By promoting component reuse, the software aims to reduce redundancy, accelerate development cycles, and improve collaboration among teams.

1.2 Scope

This software covers the management of reusable software components, including both design elements (e.g., UML diagrams, ERDs) and code (written in various programming languages). It supports functionalities such as component addition, deletion, updating, and categorization, along with advanced features for searching components using keywords. The software also includes tracking of component usage and query statistics to optimize the catalogue and ensure the most relevant components are readily available for reuse. Additionally, it features role-based access too, to ensure appropriate permissions for different users.

1.3 Definitions, Acronyms, and Abbreviations

- **Software Component**: A modular unit of software, which may include both design elements (such as UML diagrams or ERDs) and executable code.
- **Catalogue**: A centralized collection or directory of reusable software components.
- **Component Reuse**: The practice of using pre-existing software components in new applications or projects.
- **Keywords**: Descriptive terms associated with components that help in searching and identifying their functionality.
- **Versioning**: The management of different iterations or versions of a component.
- **Query**: A search request made by the user to find specific components based on keywords or other criteria.
- **Purge**: The process of removing unused or outdated components from the catalogue.
- **Admin**: A user role with permissions to manage the catalogue, including adding, deleting, and updating components, as well as configuring system settings.

1.4 References

- "Software Reuse: Architecture, Process and Organization for Software Reuse" by Ian Sommerville
- IEEE Standard 610.12-1990, "IEEE Standard Glossary of Software Engineering Terminology"
- UML Specification, Object Management Group (OMG), Version 2.5.1
- "Effective Java" by Joshua Bloch (for programming language reference)

1.5 Overview

The Software Component Cataloguing Software consists of several key features aimed at managing reusable components throughout their lifecycle. These include the ability to add, update, and delete components, organize them into categories, and associate them with keywords for easy searching. The software also tracks component usage and query statistics to aid in component optimization and purging. Users can access the catalogue based on their role and permissions, ensuring a streamlined experience for both administrators and developers. With this software, organizations can reduce development time, foster better collaboration, and promote efficient reuse of software components.

2. Overall Description

The **Software Component Cataloguing Software** is a comprehensive system designed to organize, manage, and facilitate the reuse of software components. These components can range from **design elements** (such as UML diagrams, Entity-Relationship Diagrams (ERD), and other design notations) to **code** written in various programming languages. The system is built to support software development teams by providing a centralized repository where reusable components can be catalogued, searched, and updated with ease.

The core functionality of the software revolves around efficient management of components. Users, ranging from developers to administrators, can add new components to the catalogue, update existing ones, delete outdated or unused components, and organize components into categories for easy navigation. Each component can be tagged with **keywords** that describe its functionality, allowing for quick and accurate searching. Additionally, the software tracks usage statistics, such as how often a component has been used in projects and how often it appears in search queries but is not used, helping to identify which components are most valuable.

The system supports **hierarchical classification**, allowing components to be organized into categories and subcategories, thus improving the structure of the catalogue and simplifying component discovery. This structure makes it easier for users to find relevant components quickly, minimizing the time spent searching and ensuring components are reused efficiently.

In addition to these features, the software includes **role-based access control** to ensure that users only have access to the functionality appropriate for their role. For example, an administrator may have permissions to manage the entire catalogue, including adding or deleting components and configuring system settings, while a regular user may only have access to view and search components.

To keep the catalogue relevant and efficient, the system also includes a **purging mechanism**, which removes unused components after a set period or when they no longer meet certain criteria (such as low usage frequency). This ensures that the catalogue remains lean and focused on the components that are most beneficial to users.

Overall, the **Software Component Cataloguing Software** enhances collaboration, streamlines development processes, and maximizes the reuse of software components, ultimately reducing duplication of effort and speeding up software development cycles.

3. Specific Requirements

The specific requirements are:

3.1 Functionality

This subsection details the functional requirements for the **Software Component Cataloguing Software** (SCCS). These requirements are aligned with the core features discussed in the vision document. Features from the vision document are translated into use case diagrams and sequence diagrams to clearly capture the functional needs of the system. All functional requirements are traceable via a traceability matrix.

3.1.1 Cataloguing and Configuration of Components

- **3.1.1.1** The system shall display all available software components that can be added to the catalog.
- **3.1.1.2** The system shall allow the user to select a component to configure based on type (e.g., algorithm, database design, code snippets).
- **3.1.1.3** The system shall display all available attributes (e.g., input parameters, output, dependencies) for each component.
- **3.1.1.4** The system shall allow the user to configure one or more attributes for each selected component.
- **3.1.1.5** The system shall notify the user when there are configuration conflicts (e.g., incompatible component versions).
- **3.1.1.6** The system shall allow the user to update the configuration to resolve any conflicts.
- **3.1.1.7** The system shall enable the user to confirm and finalize the configuration once completed.

3.1.2 Component Details

- **3.1.2.1** The system shall provide comprehensive details for each component, including metadata such as name, description, version, author, and any associated documentation.
- **3.1.2.2** The system shall allow users to browse the component catalog with detailed search filters (e.g., type, category, tags).

3.1.3 Component Categorization

- **3.1.3.1** The system shall organize components into well-defined categories (e.g., algorithm, code snippets, design patterns) for easy browsing.
- **3.1.3.2** The system shall allow users to navigate through hierarchical categories of components.

3.1.4 Search Functionality

- **3.1.4.1** The system shall enable the user to enter search queries to find components within the catalog.
- **3.1.4.2** The system shall support multiple filtering options for refining search results (e.g., tags, types, categories).

- **3.1.4.3** The system shall display all components matching the search query, limited to 10 results per page.
- **3.1.4.4** The system shall notify the user if no components match the search query.

3.1.5 User Profile Management

- **3.1.5.1** The system shall allow the user to create a profile with credentials (e.g., username, password).
- **3.1.5.2** The system shall authenticate the user’s credentials before allowing access to the catalog.
- **3.1.5.3** The system shall allow the user to update their profile information, including personal details and preferences.

3.1.6 Personalized User Profile

- **3.1.6.1** The system shall display the user’s active and completed component configurations in their profile.
- **3.1.6.2** The system shall allow the user to view and modify past component configurations.
- **3.1.6.3** The system shall provide personalized recommendations based on the user’s configuration history.

3.1.7 Customer Support

- **3.1.7.1** The system shall provide access to online help, FAQs, and customer support options.
- **3.1.7.2** The system shall allow users to select the type of support they need (e.g., technical support, general inquiry).
- **3.1.7.3** The system shall display relevant contact information for customer support, including phone numbers and email addresses.

3.1.8 Email Notification System

- **3.1.8.1** The system shall maintain the user’s email address as part of their profile for communication purposes.
- **3.1.8.2** The system shall send email notifications to the user for important actions (e.g., configuration completion, updates).

3.1.9 Configuration Invoice

- **3.1.9.1** The system shall display an invoice for the user after a successful configuration, showing detailed pricing if applicable.
- **3.1.9.2** The system shall allow the user to download and print the invoice.

3.1.10 Component Shopping Cart

- **3.1.10.1** The system shall provide a shopping cart functionality for the user to manage their component selections.
- **3.1.10.2** The system shall allow the user to add and remove components from the shopping cart.

3.1.11 Configuration Options and Methods

- **3.1.11.1** The system shall offer multiple configuration options based on different use cases (e.g., basic, advanced).
- **3.1.11.2** The system shall provide real-time feedback on configuration errors (e.g., incompatible components).

3.1.12 User-Defined Tags

- **3.1.12.1** The system shall allow the user to tag components with custom labels to facilitate searchability and categorization.

3.1.13 Product Reviews and Feedback

- **3.1.13.1** The system shall enable users to leave reviews and ratings for components they have configured.
- **3.1.13.2** The system shall display ratings and reviews for each component in the catalog.

3.1.14 Secure Data Handling

- **3.1.14.1** The system shall ensure that all personal and configuration data is securely encrypted during transmission and storage.

3.2 Usability

3.2.1 Graphical User Interface (GUI)

- **3.2.1.1** The system shall provide a consistent, user-friendly interface across all screens, ensuring seamless navigation between components, configuration pages, and user profiles.
- **3.2.1.2** Each component in the catalog shall have an associated image or graphical representation.
- **3.2.1.3** The system shall use intuitive icons and toolbars to facilitate user interaction and configuration of components.

3.2.2 Accessibility

- **3.2.2.1** The system shall be fully accessible to users with disabilities, ensuring compatibility with screen readers.
- **3.2.2.2** The system shall support multiple languages for international accessibility.
- **3.2.2.3** The system shall provide features such as keyboard navigation and high-contrast modes to assist users with visual or motor disabilities.

3.3 Reliability & Availability

3.3.1 Backend Infrastructure

- **3.3.1.1** The system’s databases shall be replicated for redundancy and high availability, ensuring minimal downtime.

3.3.2 Network Availability

- **3.3.2.1** The system shall be hosted with an internet service provider offering efficient availability.

3.4 Performance

- **3.4.1** The system shall be web-based, ensuring it is accessible from any device with internet access.
- **3.4.2** The system's response time should be optimized, with a target initial load time of 2-3 seconds.

3.5 Security

3.5.1 Data Transmission

- **3.5.1.1** The system shall utilize secure sockets layer (SSL) encryption for all transactions involving sensitive data.
- **3.5.1.2** All sessions shall automatically log out after 15 minutes of inactivity.

3.5.2 Data Storage

- **3.5.2.1** The system shall never store plaintext passwords; they will be hashed and encrypted.
- **3.5.2.2** User data shall be encrypted both in transit and at rest to ensure confidentiality and integrity.

3.6 Supportability

3.6.1 Configuration Management

- **3.6.1.1** All source code for the system shall be managed using a configuration management tool to track changes and ensure version control.

3.7 Design Constraints

3.7.1 Development Tools

- **3.7.1.1** The system shall be developed using modern web technologies, ensuring it is cross-browser compatible (e.g., Chrome, Firefox, Safari).
- **3.7.1.2** The system shall adhere to industry best practices for GUI design and security.

3.8 Online User Documentation and Help System Requirements

- **3.8.1** The system shall provide a comprehensive online help system, including searchable documentation and user guides.
- **3.8.2** Users shall be able to access tutorial videos and FAQs directly within the platform.

3.9 Interfaces

3.9.1 User Interface

- **3.9.1.1** The user interface shall be compatible with all modern web browsers, providing a responsive design for both desktop and mobile platforms.

3.9.2 Software Interfaces

- **3.9.2.1** The system shall integrate with external configuration tools and databases if required (via APIs).

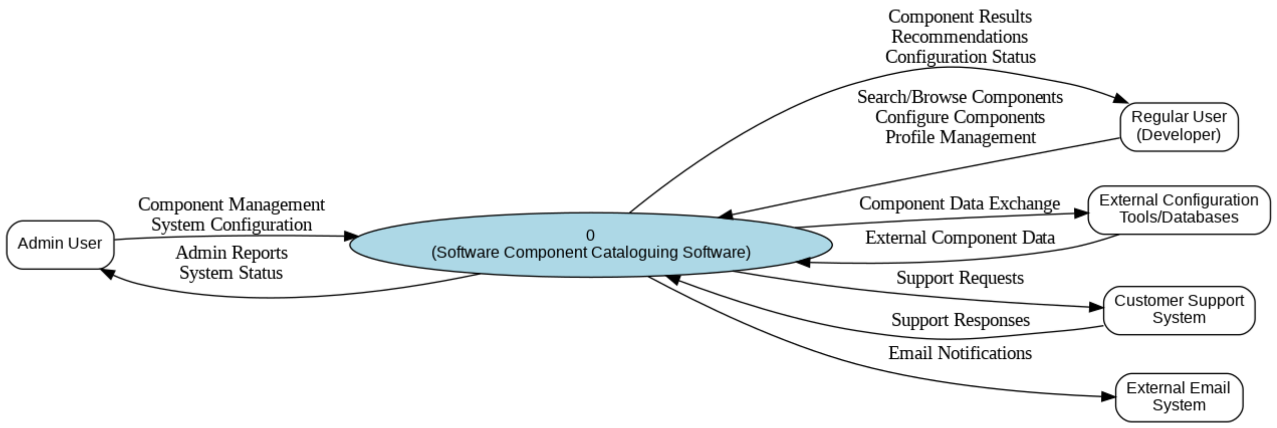
4. Supporting Information

Please refer to the following documents:

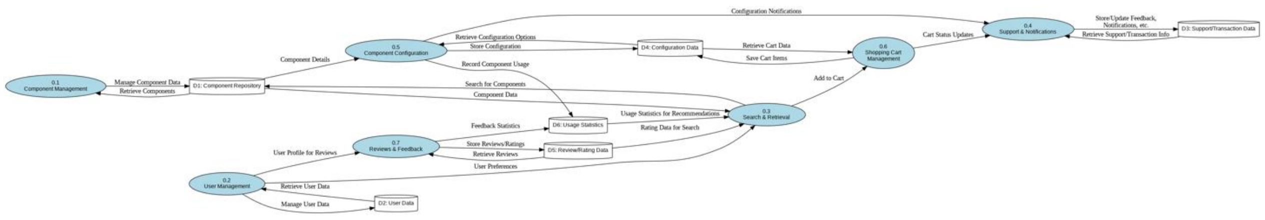
1. **Vision Document for Software Component Cataloguing Software (SCCS)**
   (This document outlines the overall vision, objectives, and target audience for the SCCS.)
2. **Use Case Analysis**
   (This document provides detailed analysis of use cases and user interactions within the system.)
3. **Structural Models**
   (This document contains architectural diagrams and detailed class structures for the system's components.)
4. **Behavioral Models**
   (This document includes sequence diagrams, state diagrams, and activity diagrams illustrating the system's behavior during various operations.)
5. **Non-Functional Requirements Model**
   (This document defines the system's performance, security, reliability, and other non-functional attributes.)
6. **Traceability Matrix**
   (This document links the functional requirements to corresponding system components and modules, ensuring all requirements are met.)
7. **Project Plan**
   (This document outlines the development phases, timelines, and milestones for the SCCS project.)

DFD (Data Flow Diagrams) Models

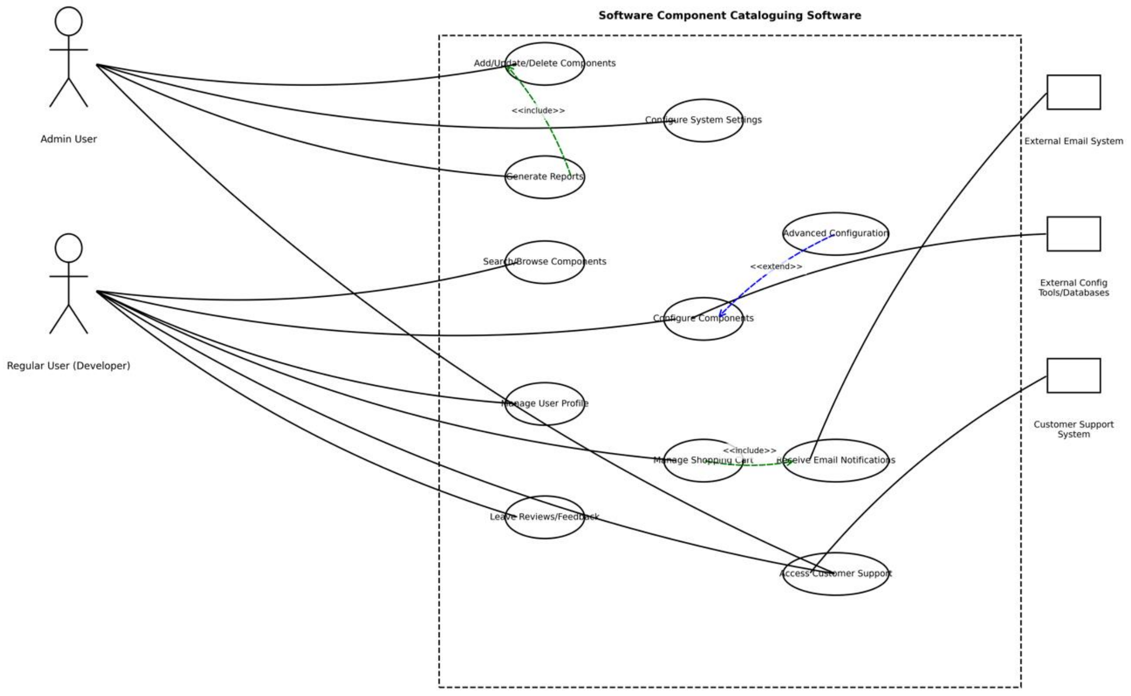
DFD Level - 0:



DFD Level - 1:



Use-Case Diagram



Use-Case Documentation

1: Add/Update/Delete Components

ID: 1
Title: Add/Update/Delete Components
Description: Allows the administrator(s), i.e. Admin User(s) to manage component entries in the cataloguing system, including creating new component entries, modifying existing component information, and removing outdated components.
Primary Actor: Admin User
Secondary Actor: None
Preconditions: User should be authenticated as an Admin, and he/she should be having proper access rights to manage components.
Postconditions: Component information would be successfully added, updated, or removed from the system.
Dependency: None
Generalization: None
Main Success Scenario: Admin user logs into the system. He/she navigates to the component management section, where there would be different options for adding/updating/removing components. For adding a component, the admin user selects 'Add Component' option, after which he/she enters the new component's details. The details, after validation by the system, are saved, and the component is added. For updating, the admin searches for an existing component in the system, then after selecting the required component, he/she would modify it according to his/her needs. Again, like the adding option, the details are validated by the system, and then saved. For deleting, the admin user searches for the existing component to delete. He/she selects the required component, removes it using the 'Delete' option; and after confirmation by the system, the component is deleted.
Extensions or Alternate Flow: System validation may fail, in which case it would be indicated to the admin user. The existing component which is to be deleted might be referenced by other entities, and so the system should warn the admin user about the dependencies. There may be other related errors in application of different options, operations etc., in which the system can display an error message to the admin about the same.
Frequency of Use: High
Status: Implemented
Owner: Project Head
Priority: High

2: Configure System Settings

ID: 2
Title: Configure System Settings
Description: Allows administrator(s) to manage system-wide configuration parameters that control the behavior and appearance of the software.
Primary Actor: Admin User
Secondary Actor: None
Preconditions: User should be authenticated as an Admin, and he/she should be having proper access rights to manage and configure system settings.

Postconditions: System settings are configured and updated according to admin's specifications, and the changes are saved and applied.
Dependency: None
Generalization: None
Main Success Scenario: Admin navigates to the system configuration option. There he/she can see the current configuration settings, and can modify the desired parameters according to needs. After submitting the changes successfully, the changes are validated by the system, and after that they are saved and applied in the system.
Extensions or Alternate Flow: System validation may fail, in which it would be indicated to the admin user. There may be some invalid configuration options, which if selected should be highlighted by the system. The system can provide the related information about the issue caused to the admin user.
Frequency of Use: Medium
Status: Implemented
Owner: Project Head
Priority: Medium

3: Generate Reports

ID: 3
Title: Generate Reports
Description: Allows administrator(s) to create reports about component usage, user activity and system performance which helps in tracking system utilization.
Primary Actor: Admin User
Secondary Actor: None
Preconditions: User should be authenticated as an Admin, and he/she should be having proper access rights to manage and generate reports. The related data should also ofcourse be there in the system.
Postconditions: The requested report would be generated containing the required information.
Dependency: Includes: Add/Update/Delete Components.
Generalization: None
Main Success Scenario: Admin user navigates to the report section, in which there are different report types present. The admin user selects the desired report type and provides the necessary information for the required parameters. After filling the details, a report preview is displayed, which if saved by the admin, can then be downloaded and subsequently shared by the admin.
Extensions or Alternate Flow: There might be no data available for the required parameters, in which the system would notify the admin about the error caused. There may be report generation errors due to various factors, which would be mentioned by the system to the admin user.
Frequency of Use: Medium
Status: Implemented
Owner: Project Head
Priority: Medium

4: Search/Browse Components

ID: 4
Title: Search/Browse Components
Description: Allows users (developers) to find components within the catalogue by using search functionality or by browsing through the categories.
Primary Actor: Regular User (Developer)
Secondary Actor: None
Preconditions: User should be authenticated in the system. And the component catalogue should contain searchable/browsable components.
Postconditions: The user would find the relevant components based on the search criteria. The required results would be displayed.
Dependency: None
Generalization: None
Main Success Scenario: User navigates to search/browse option. For searching, the user enters the required terms, uses the needed filters or other related search criteria, which after processing by the system, displays the required component. For browsing, the user navigates through the different component categories or tags to find the required component. After successful searching/browsing of the component, the system provides detailed information of the component.
Extensions or Alternate Flow: There may be no results found, i.e. the needed component doesn’t exist in the list. So, in that case the system should inform the user about that.
There may even be too many results possible for a given search criteria, and so the system should suggest modifying and refining the search criteria to narrow down the choices. There may be other possible system related errors too, about which the user should be notified if they occur.
Frequency of Use: High
Status: Implemented
Owner: Project Head
Priority: High

5: Configure Components

ID: 5
Title: Configure Components
Description: Enables users (developers) to customize and configure component settings for specific needs.
Primary Actor: Regular User (Developer)
Secondary Actor: External Config Tools/Databases
Preconditions: User should be authenticated in the system. And the component should have customizable (modifiable) and configurable options.
Postconditions: Component would be configured according to user specifications, and saved in the system.
Dependency: Extends: Advanced Configuration.
Generalization: None
Main Success Scenario: User selects a component to configure. The related details of the component are displayed. The user can modify the needed parameters, according to requirements, which if confirmed, are validated by the system, and saved and applied successfully.
Extensions or Alternate Flow: Some cases of invalid configuration may be detected. This may occur due to several reasons, and should be highlighted to the user if it does.

Frequency of Use: High
Status: Implemented
Owner: Project Head
Priority: High

6: Manage User Profile

ID: 6
Title: Manage User Profile
Description: Allows users to view and update their profile information, preferences, and account settings within the system.
Primary Actor: Regular User (Developer)
Secondary Actor: None
Preconditions: User should be authenticated in the system.
Postconditions: The user profile information is updated as requested, and the profile data is saved.
Dependency: None
Generalization: None
Main Success Scenario: User navigates to the profile management section, where the current profile information is displayed. The user updates the desired information according to the needed requirements. After the changes are submitted, the updates are validated by the system and saved.
Extensions or Alternate Flow: There may be possible validation errors, which should be indicated properly to the user.
Frequency of Use: Medium
Status: Implemented
Owner: Project Head
Priority: Medium

7: Manage Shopping Cart

ID: 7
Title: Manage Shopping Cart
Description: Allows users to save components of interest to a shopping cart for later reference.
Primary Actor: Regular User (Developer)
Secondary Actor: None
Preconditions: User should be authenticated in the system. Components should also ofcourse be available in the system for selection.
Postconditions: Components are added to, or removed from the user's shopping cart, and saved.
Dependency: Includes: Receive Email Notifications.
Generalization: None
Main Success Scenario: User browses or searches for components. User selects the ‘Add to Cart’ option to add the desired components in shopping cart. After adding all the needed components, the user can choose to review the cart contents, remove items, or update quantities. The final cart saved can then further be processed for purchasing or downloading.

Extensions or Alternate Flow: Some components may be unavailable, which should be promptly notified to the user. The user may attempt to add a duplicate item, which should be detected by the system and should be indicated to the user.
Frequency of Use: High
Status: Implemented
Owner: Project Head
Priority: High

8: Advanced Configuration

ID: 8
Title: Advanced Configuration
Description: Provides extended advanced configuration options for components, allowing developers to access deeper levels of customization options.
Primary Actor: Regular User (Developer)
Secondary Actor: None
Preconditions: User should be authenticated in the system. And, the component should support advanced configuration.
Postconditions: The needed component would be configured with advanced settings, and would be saved.
Dependency: Extended by: Configure Components
Generalization: None
Main Success Scenario: User accesses the advanced configuration option in the required component, where he/she can modify and customize the advanced parameters. After confirming the changes, they are validated by the system, and then saved and applied.
Extensions or Alternate Flow: There may be issues/conflicts in the advanced configuration options, which should be promptly specified to the user.
Frequency of Use: Medium
Status: Implemented
Owner: Project Head
Priority: Medium

9: Receive Email Notifications

ID: 9
Title: Receive Email Notifications
Description: Enables the system to send automated email notifications to users about relevant events.
Primary Actor: Regular User (Developer)
Secondary Actor: External Email System
Preconditions: User should be authenticated in the system, and should have a valid email address connected to his/her profile. A required event should take place so that the notifications are sent.
Postconditions: The necessary email notification would be generated and sent to the user.
Dependency: None
Generalization: None
Main Success Scenario: When an event which triggers/necessitates a notification takes place, the system generates a notification with an appropriate message which is sent through mail to the user.

Extensions or Alternate Flow: The email delivery process may fail, which if it does, should be re-tried by the system. User may not even have the option of receiving email notifications switched on, and so the system should check those preferences first before trying to send the email notifications.
Frequency of Use: Medium
Status: To be implemented
Owner: Project Head
Priority: Medium

10: Leave Reviews/Feedback

ID: 10
Title: Leave Reviews/Feedback
Description: Allows users to provide ratings and written feedback reviews about components used.
Primary Actor: Regular User (Developer)
Secondary Actor: None
Preconditions: User should be authenticated in the system. The component about which the review is given should ofcourse exist in the catalogue, and should have been accessed by the user previously atleast once.
Postconditions: User's review/feedback would be recorded in the system and would be associated with the specific component.
Dependency: None
Generalization: None
Main Success Scenario: User navigates to the required component's details, where he/she selects the option to leave review/feedback. There the user can provide a written review or can provide an appropriate rating for the component, or both. After submitting the review, it is validated by the system, after which it is saved.
Extensions or Alternate Flow: The review may contain problematic inappropriate content/wording and so the user should be notified by the system that the review was rejected due to violation of guidelines. The user may want to update one of his/her existing reviews, in which the updated review would be saved after validation.
Frequency of Use: Medium
Status: Implemented
Owner: Project Head
Priority: Medium

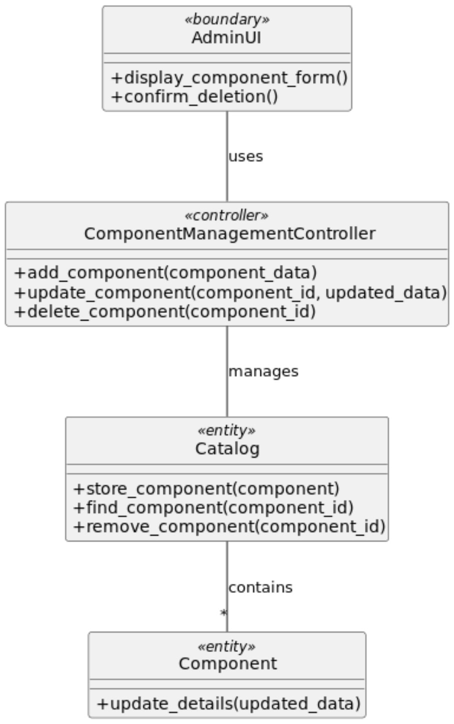
11: Access Customer Support

ID: 11
Title: Access Customer Support
Description: Enables users to contact customer support, submit help requests, report issues, or seek assistance.
Primary Actor: Regular User (Developer) or Admin User
Secondary Actor: Customer Support System
Preconditions: The user (admin/developer) should be authenticated in the system.
Postconditions: The support request would be successfully submitted, and a ticket would be generated for the same. The user would get an acknowledgement for it as well.
Dependency: None

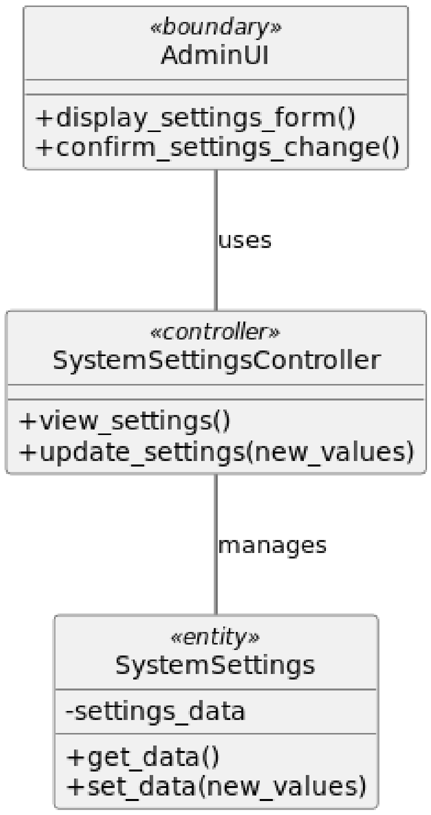
Generalization: None
Main Success Scenario: User navigates to the support section, where he/she fills the support request with the relevant details. After submitting the request, a ticket is created by the Customer Support System. The user gets a confirmation for the same as well. The support system processes the request and responds to the user accordingly.
Extensions or Alternate Flow: The support request submission may fail (which could be possible due to multiple reasons), which should be promptly specified to the user.
Frequency of Use: Medium
Status: To be implemented
Owner: Project Head
Priority: High

Domain Models

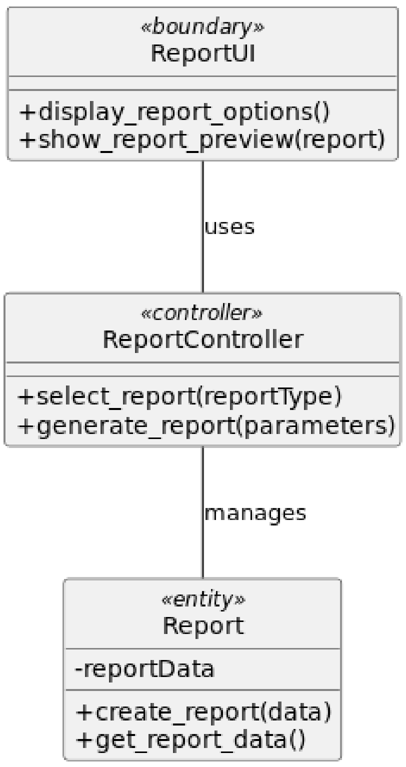
Domain Diagram - Use Case #1 (Add/Update/Delete Components)


«boundary»
AdminUI
+display_component_form()
+confirm_deletion()
uses
«controller»
ComponentManagementController
+add_component(component_data)
+update_component(component_id, updated_data)
+delete_component(component_id)
manages
«entity»
Catalog
+store_component(component)
+find_component(component_id)
+remove_component(component_id)
contains
*
«entity»
Component
+update_details(updated_data)

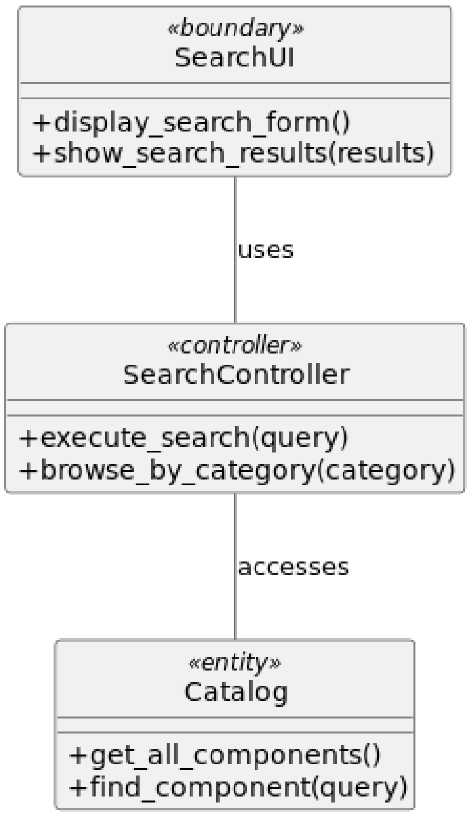
Domain Diagram - Use Case #2 (Configure System Settings)



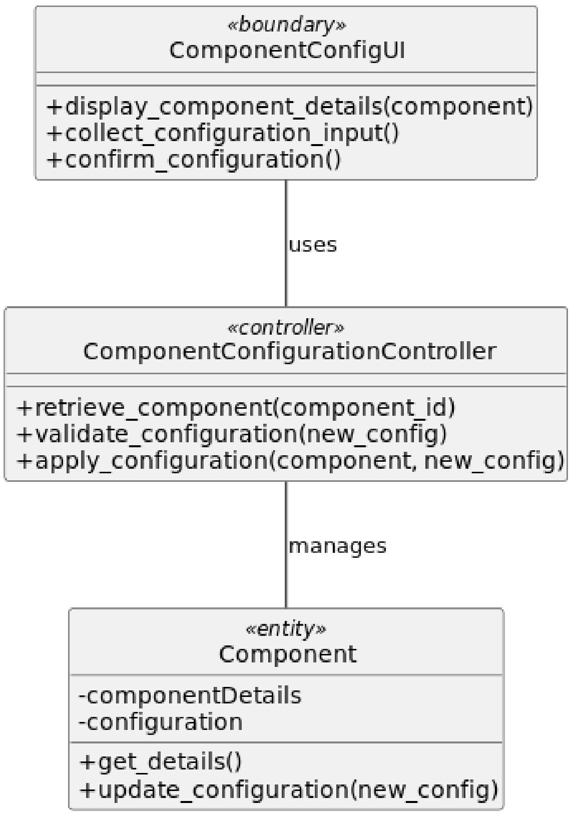
Domain Diagram - Use Case #3 (Generate Reports)



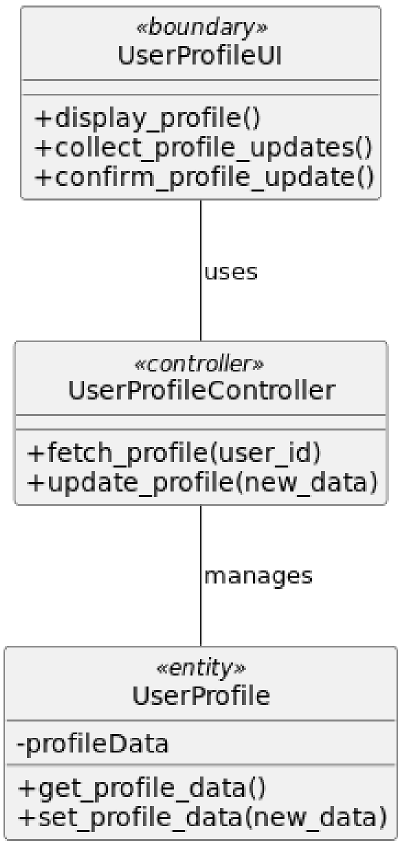
Domain Diagram - Use Case #4 (Search/Browse Components)



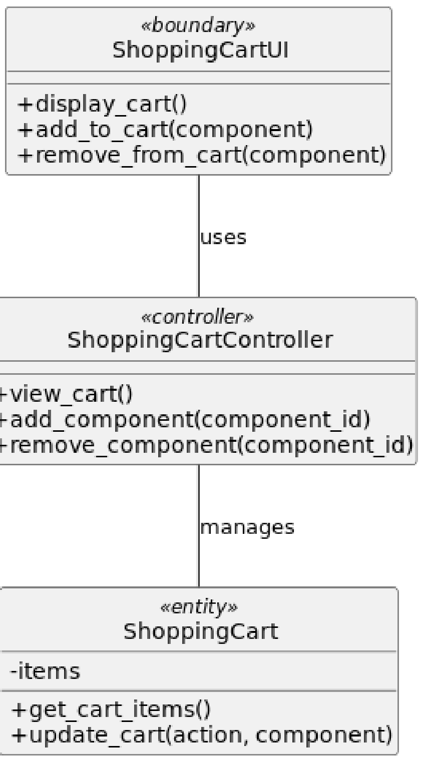
Domain Diagram - Use Case #5 (Configure Components)


«boundary»
ComponentConfigUI
+display_component_details(component)
+collect_configuration_input()
+confirm_configuration()
uses
«controller»
ComponentConfigurationController
+retrieve_component(component_id)
+validate_configuration(new_config)
+apply_configuration(component, new_config)
manages
«entity»
Component
-componentDetails
-configuration
+get_details()
+update_configuration(new_config)

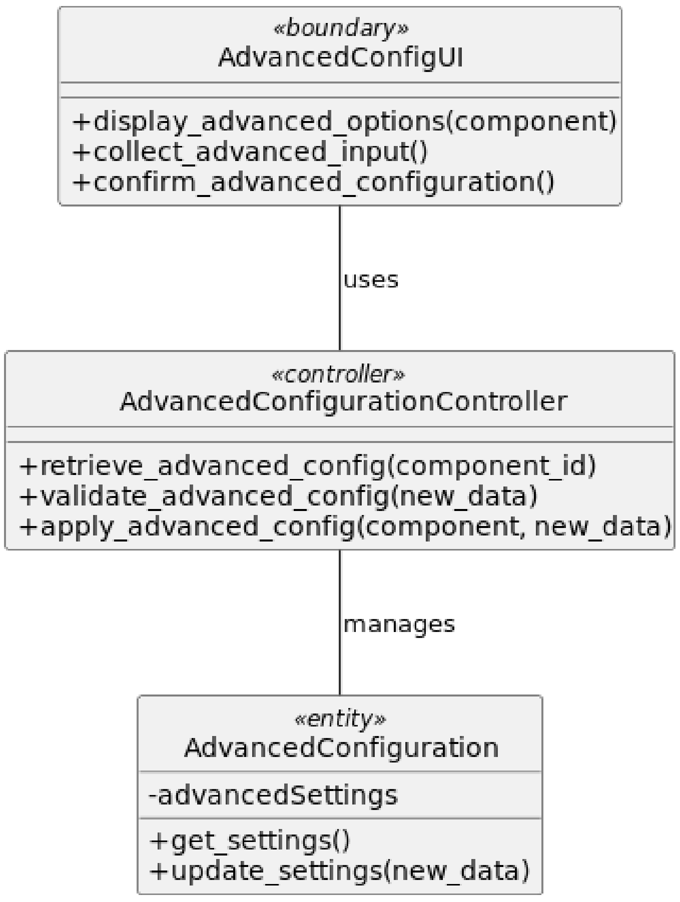
Domain Diagram - Use Case #6 (Manage User Profile)



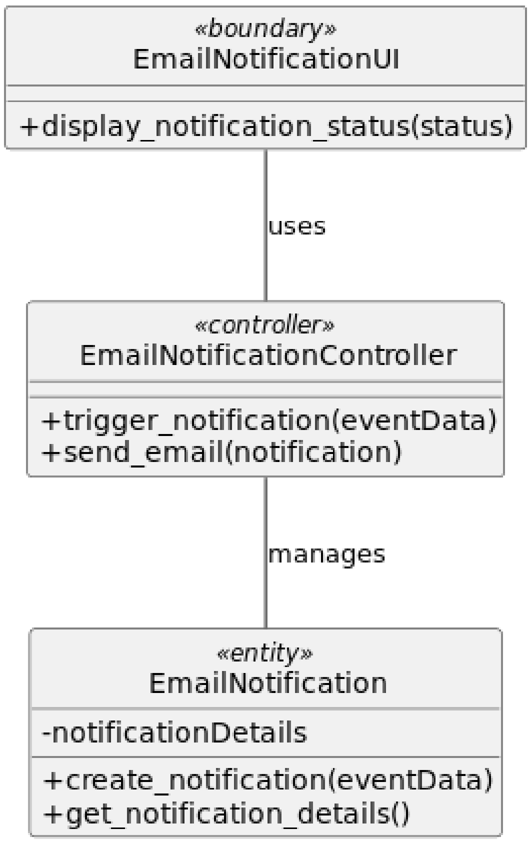
Domain Diagram - Use Case #7 (Manage Shopping Cart)



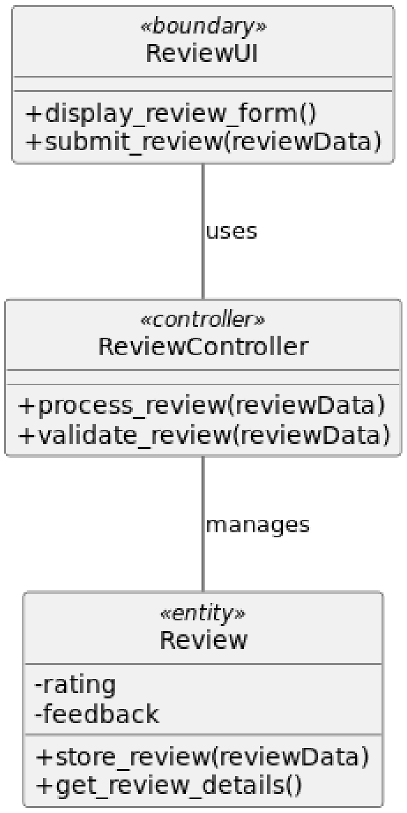
Domain Diagram - Use Case #8 (Advanced Configuration)


«boundary»
AdvancedConfigUI
+display_advanced_options(component)
+collect_advanced_input()
+confirm_advanced_configuration()
uses
«controller»
AdvancedConfigurationController
+retrieve_advanced_config(component_id)
+validate_advanced_config(new_data)
+apply_advanced_config(component, new_data)
manages
«entity»
AdvancedConfiguration
-advancedSettings
+get_settings()
+update_settings(new_data)

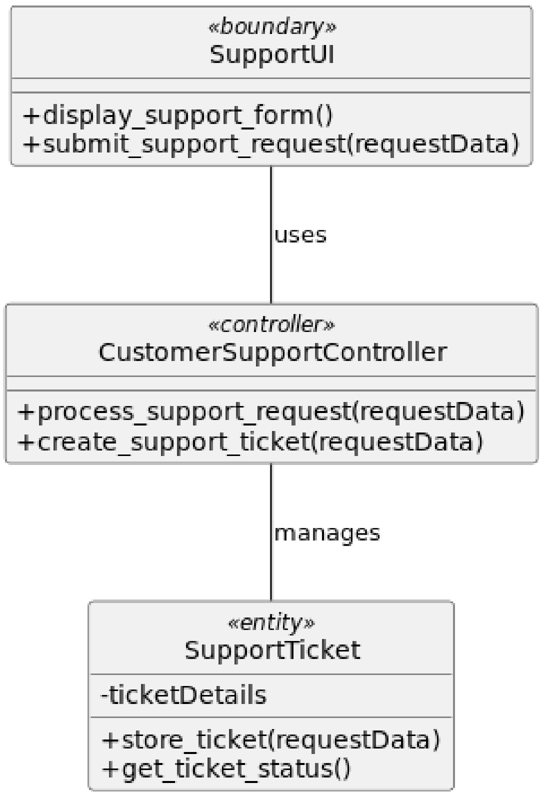
Domain Diagram - Use Case #9 (Receive Email Notifications)


«boundary»
EmailNotificationUI
+display_notification_status(status)
uses
«controller»
EmailNotificationController
+trigger_notification(eventData)
+send_email(notification)
manages
«entity»
EmailNotification
-notificationDetails
+create_notification(eventData)
+get_notification_details()

Domain Diagram - Use Case #10 (Leave Reviews/Feedback)

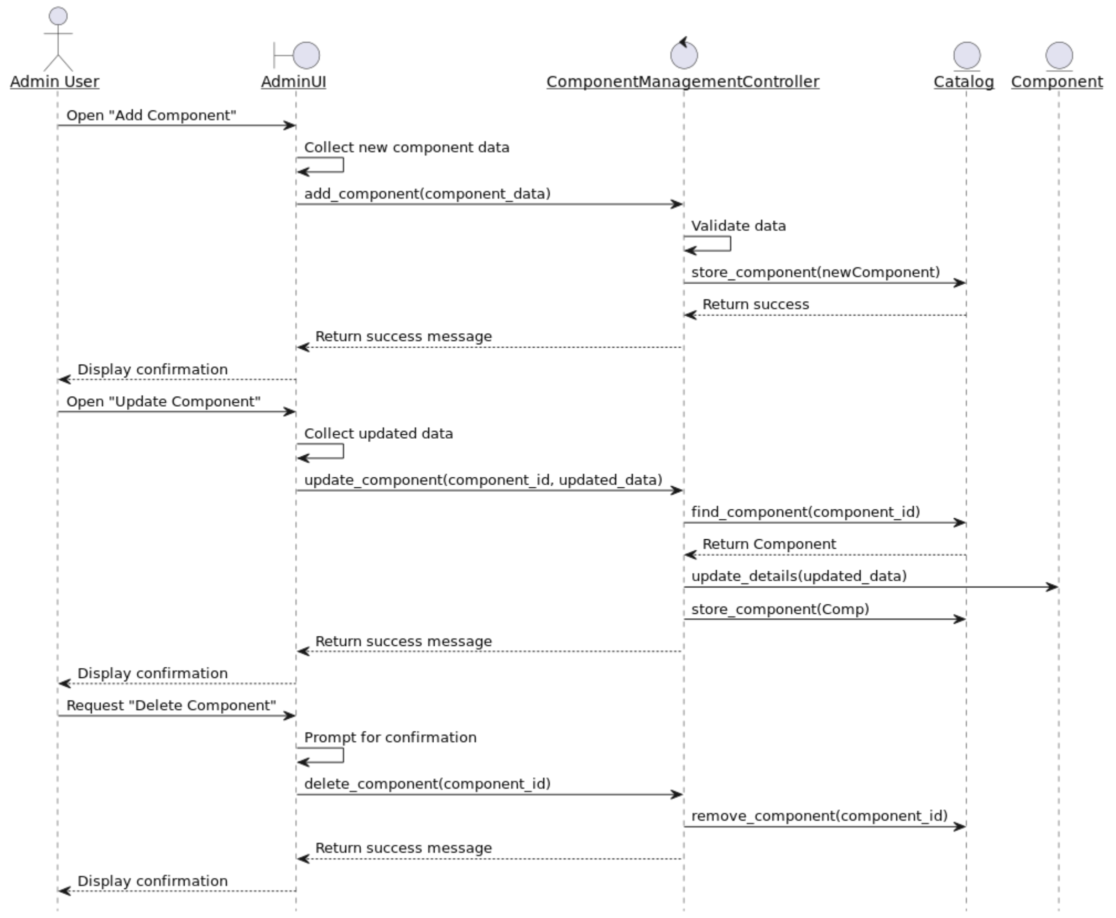


Domain Diagram - Use Case #11 (Access Customer Support)

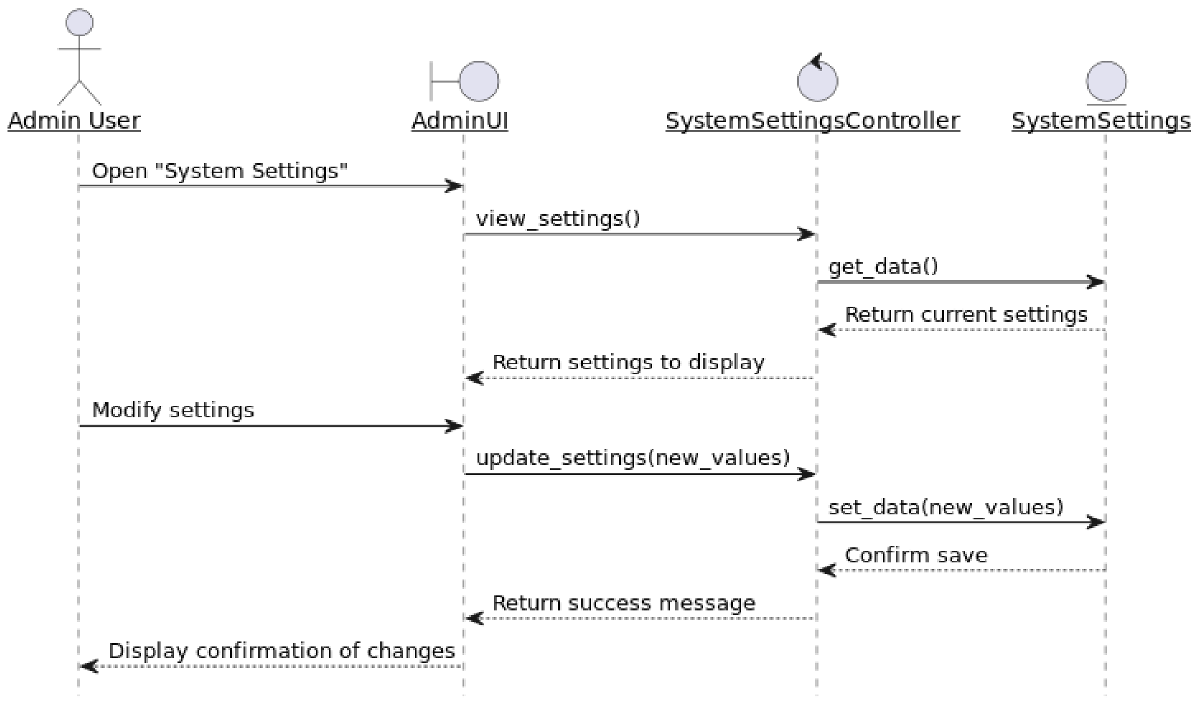

«boundary»
SupportUI
+display_support_form()
+submit_support_request(requestData)
uses
«controller»
CustomerSupportController
+process_support_request(requestData)
+create_support_ticket(requestData)
manages
«entity»
SupportTicket
-ticketDetails
+store_ticket(requestData)
+get_ticket_status()

Sequence Diagrams

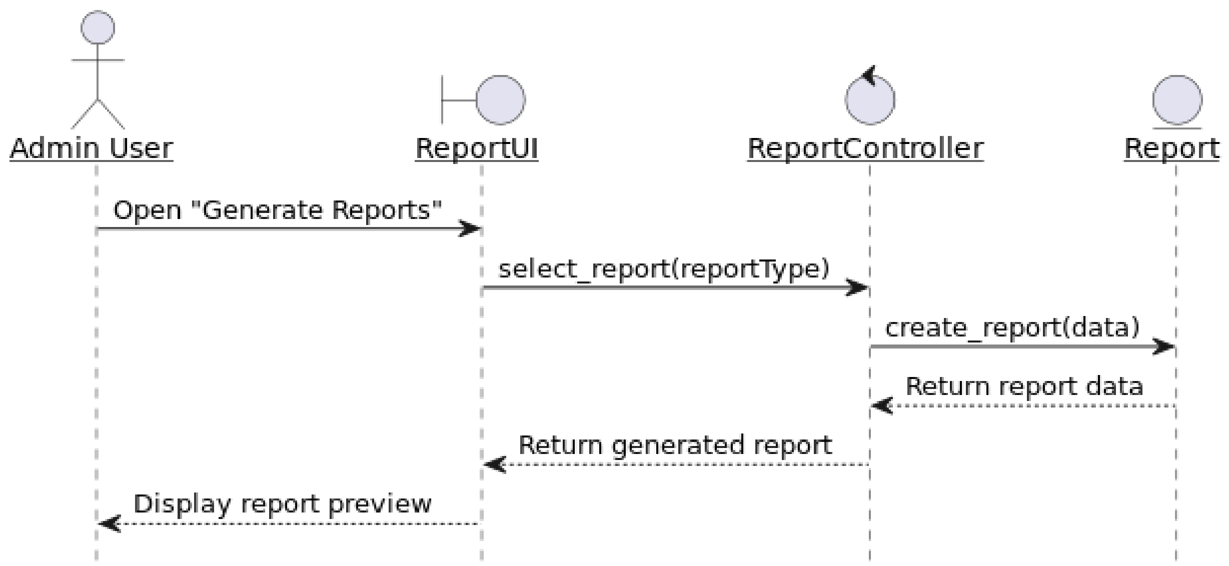
Sequence Diagram - Use Case #1 (Add/Update/Delete Components)



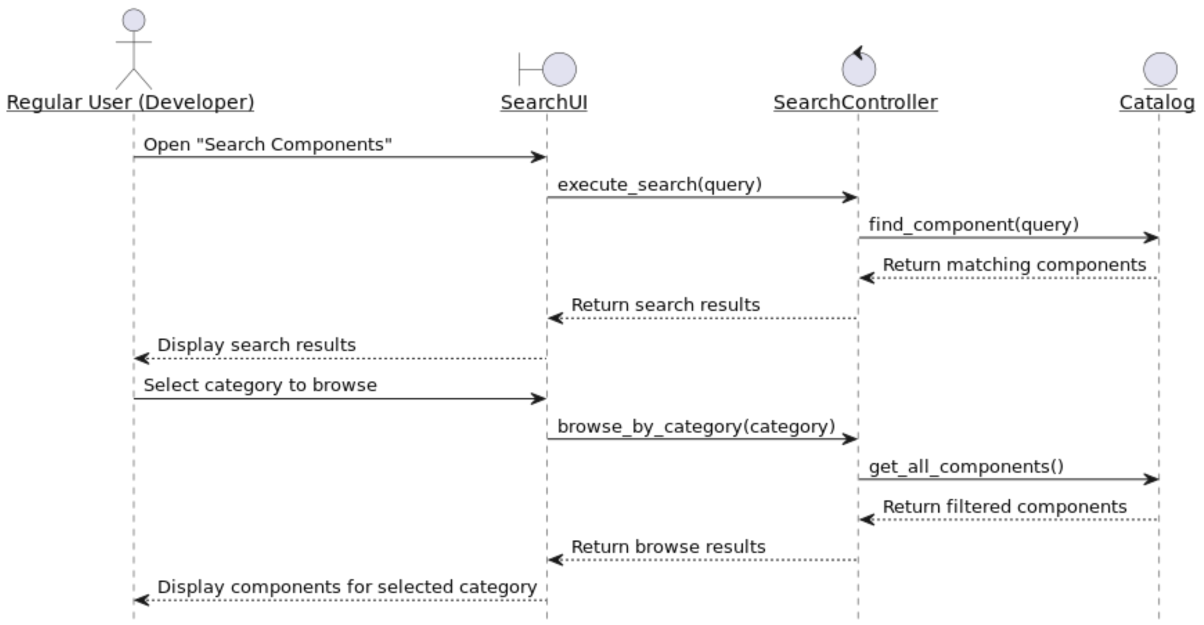
Sequence Diagram - Use Case #2 (Configure System Settings)



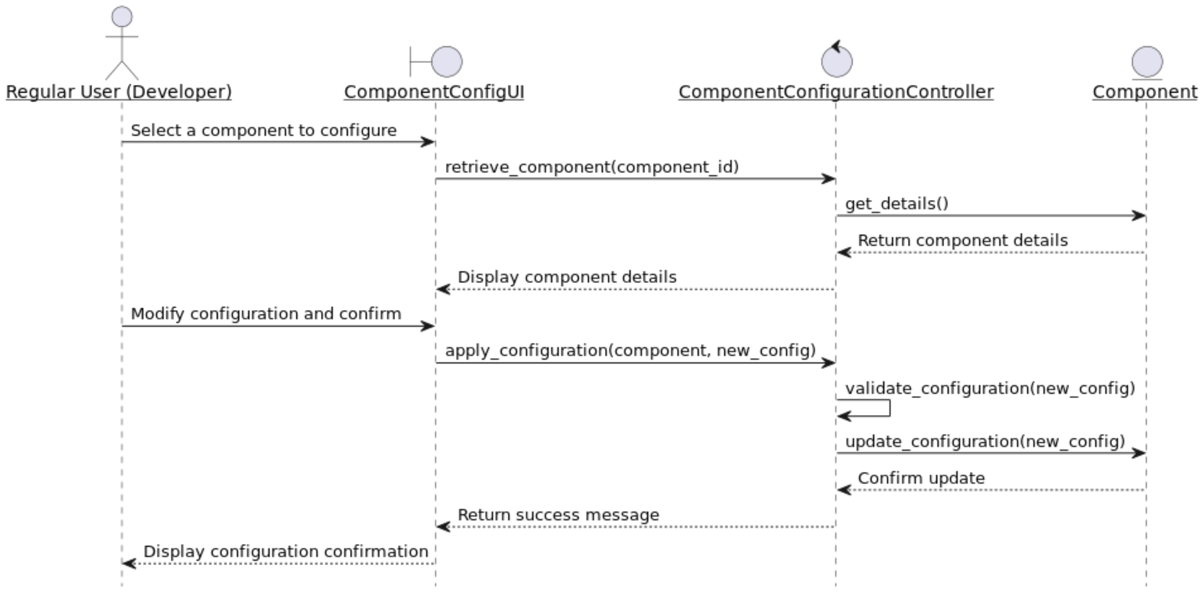
Sequence Diagram - Use Case #3 (Generate Reports)



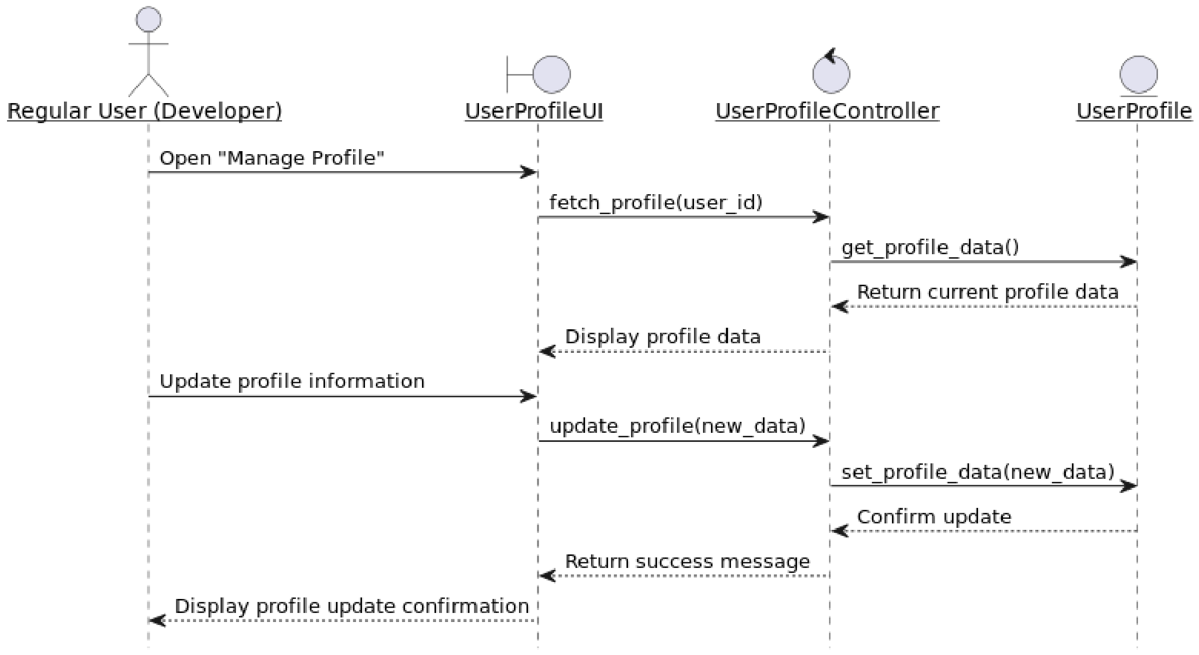
Sequence Diagram - Use Case #4 (Search/Browse Components)



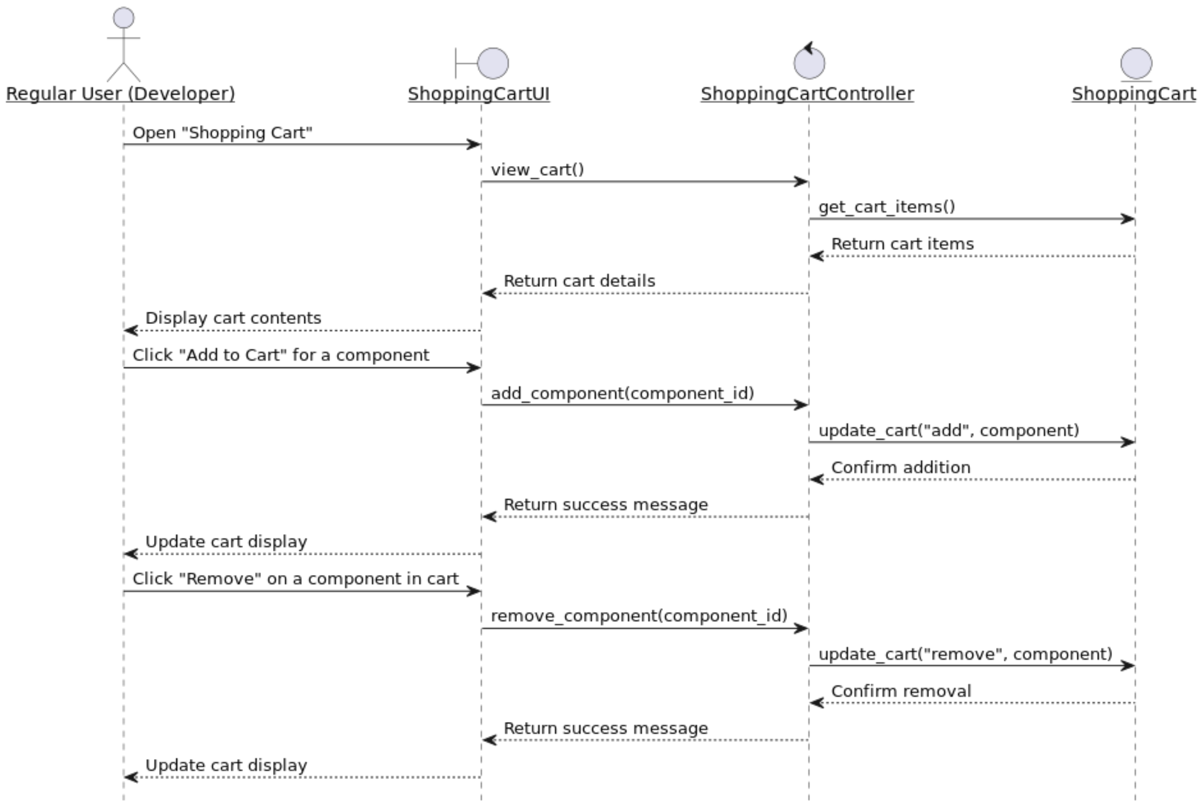
Sequence Diagram - Use Case #5 (Configure Components)


Regular User (Developer)
ComponentConfigUI
ComponentConfigurationController
Component
Select a component to configure
retrieve_component(component_id)
get_details()
Return component details
Display component details
Modify configuration and confirm
apply_configuration(component, new_config)
validate_configuration(new_config)
update_configuration(new_config)
Confirm update
Return success message
Display configuration confirmation

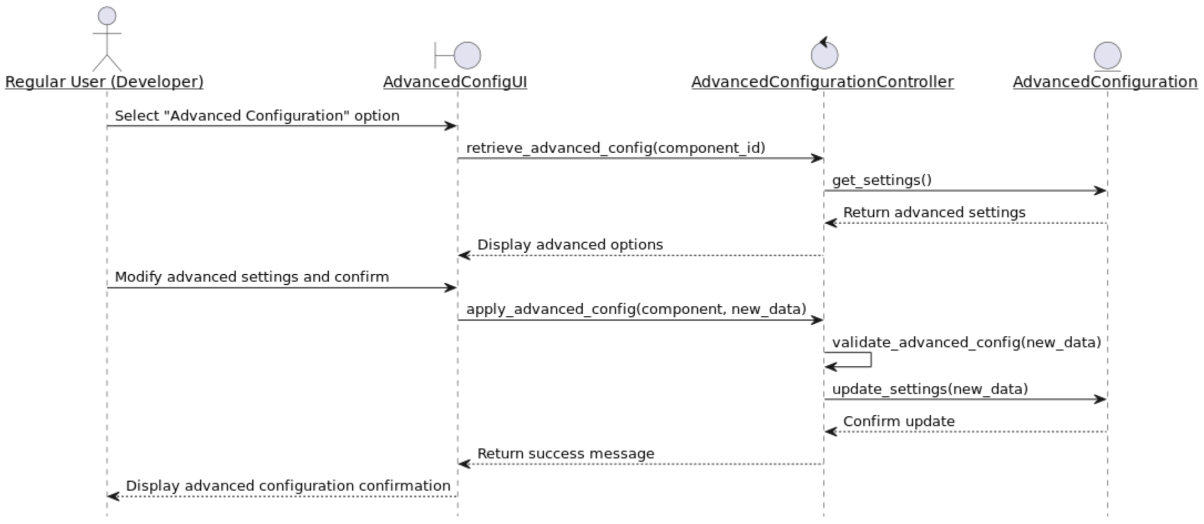
Sequence Diagram - Use Case #6 (Manage User Profile)



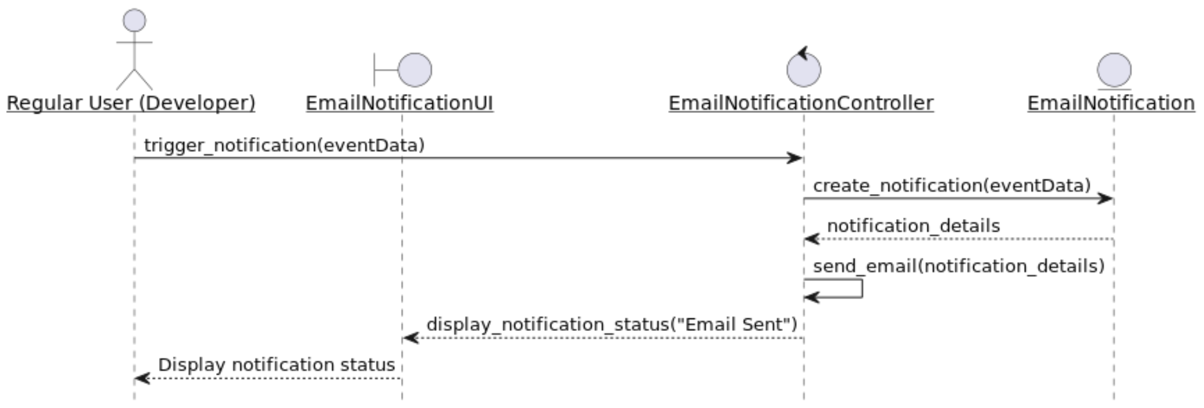
Sequence Diagram - Use Case #7 (Manage Shopping Cart)



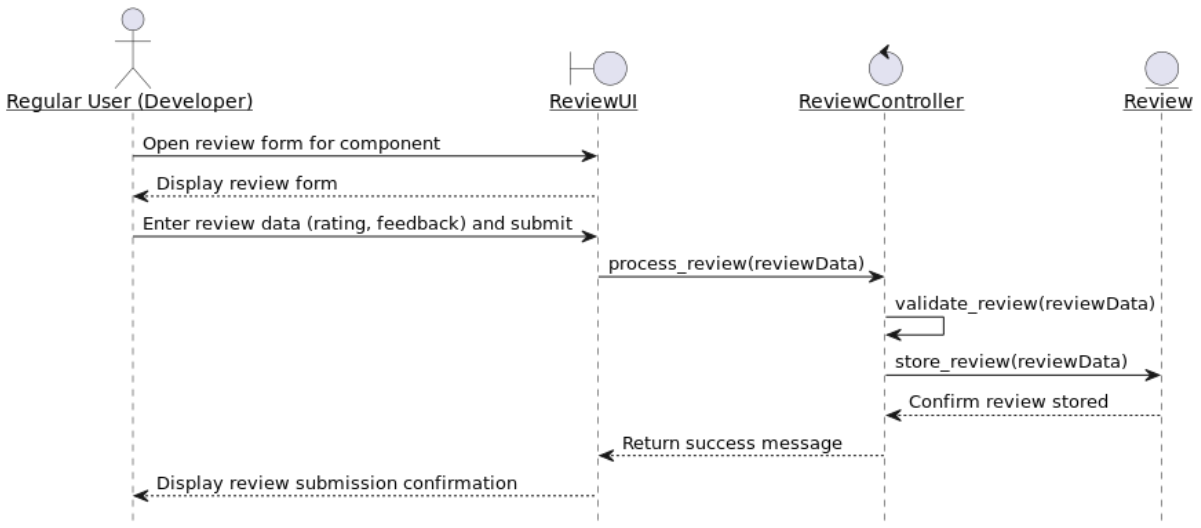
Sequence Diagram - Use Case #8 (Advanced Configuration)


Regular User (Developer)
AdvancedConfigUI
AdvancedConfigurationController
AdvancedConfiguration
Select "Advanced Configuration" option
retrieve_advanced_config(component_id)
get_settings()
Return advanced settings
Display advanced options
Modify advanced settings and confirm
apply_advanced_config(component, new_data)
validate_advanced_config(new_data)
update_settings(new_data)
Confirm update
Return success message
Display advanced configuration confirmation

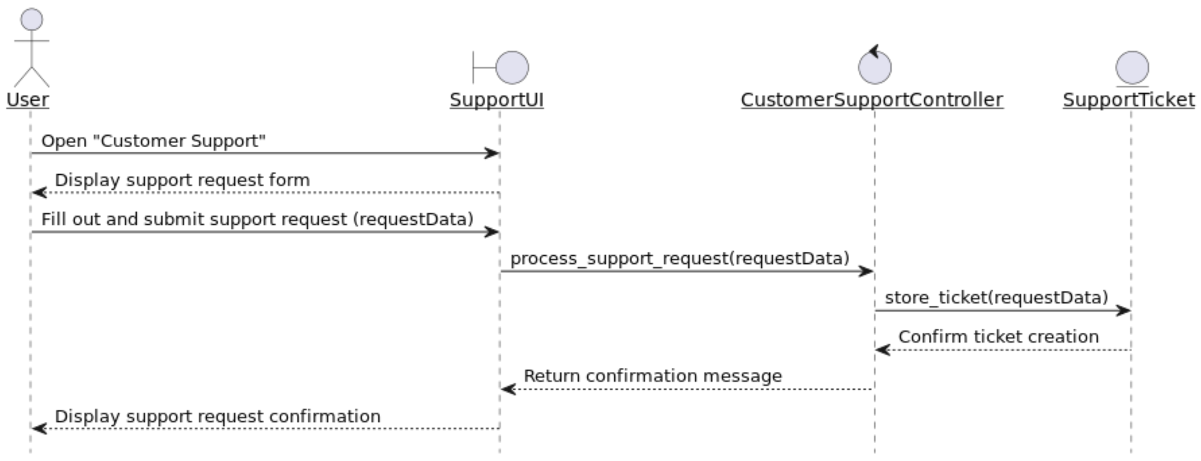
Sequence Diagram - Use Case #9 (Receive Email Notifications)


Regular User (Developer)
EmailNotificationUI
EmailNotificationController
EmailNotification
trigger_notification(eventData)
create_notification(eventData)
notification_details
send_email(notification_details)
display_notification_status("Email Sent")
Display notification status

Sequence Diagram - Use Case #10 (Leave Reviews/Feedback)

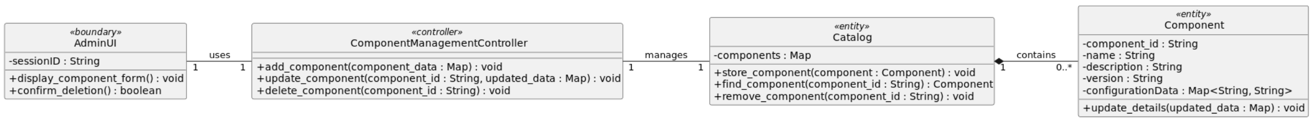

Regular User (Developer)
ReviewUI
ReviewController
Review
Open review form for component
Display review form
Enter review data (rating, feedback) and submit
process_review(reviewData)
validate_review(reviewData)
store_review(reviewData)
Confirm review stored
Return success message
Display review submission confirmation

Sequence Diagram - Use Case #11 (Access Customer Support)

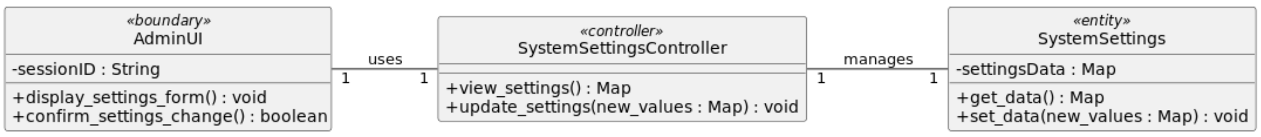

User
SupportUI
CustomerSupportController
SupportTicket
Open "Customer Support"
Display support request form
Fill out and submit support request (requestData)
process_support_request(requestData)
store_ticket(requestData)
Confirm ticket creation
Return confirmation message
Display support request confirmation

Class Diagrams

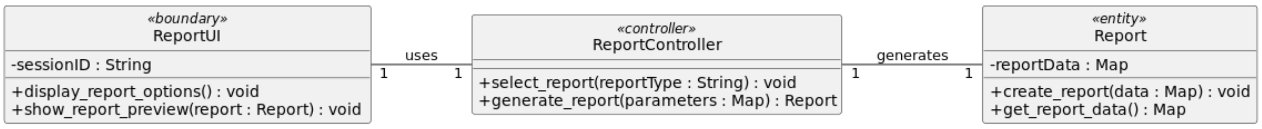
Class Diagram - Use Case #1 (Add/Update/Delete Components)


«boundary»
AdminUI
-sessionID : String
+display_component_form() : void
+confirm_deletion() : boolean
uses
1
1
«controller»
ComponentManagementController
+add_component(component_data : Map) : void
+update_component(component_id : String, updated_data : Map) : void
+delete_component(component_id : String) : void
manages
1
1
«entity»
Catalog
-components : Map
+store_component(component : Component) : void
+find_component(component_id : String) : Component
+remove_component(component_id : String) : void
contains
1
0..*
«entity»
Component
-component_id : String
-name : String
-description : String
-version : String
-configurationData : Map<String, String>
+update_details(updated_data : Map) : void

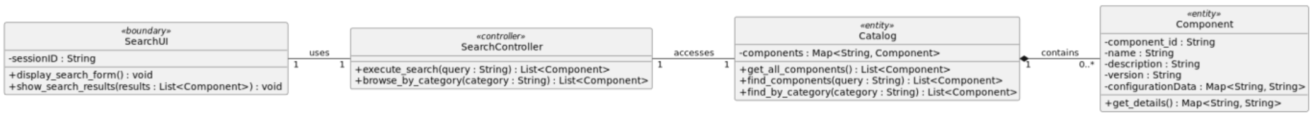
Class Diagram - Use Case #2 (Configure System Settings)



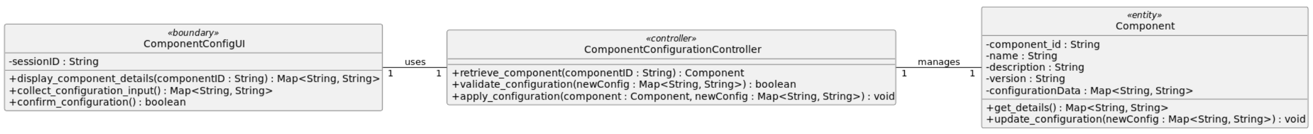
Class Diagram - Use Case #3 (Generate Reports)


«boundary»
ReportUI
-sessionID : String
+display_report_options() : void
+show_report_preview(report : Report) : void
uses
1
1
«controller»
ReportController
+select_report(reportType : String) : void
+generate_report(parameters : Map) : Report
generates
1
1
«entity»
Report
-reportData : Map
+create_report(data : Map) : void
+get_report_data() : Map

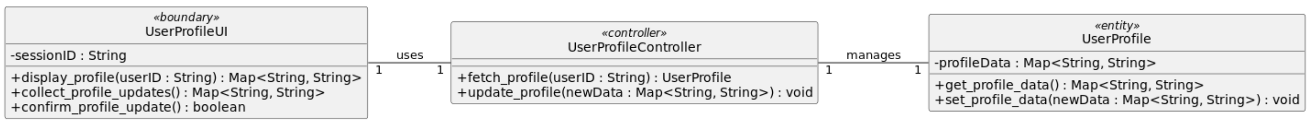
Class Diagram - Use Case #4 (Search/Browse Components)


«boundary»
SearchUI
-sessionID : String
+display_search_form() : void
+show_search_results(results : List<Component>) : void
uses
1
1
«controller»
SearchController
+execute_search(query : String) : List<Component>
+browse_by_category(category : String) : List<Component>
accesses
1
1
«entity»
Catalog
-components : Map<String, Component>
+get_all_components() : List<Component>
+find_components(query : String) : List<Component>
+find_by_category(category : String) : List<Component>
contains
1
0..*
«entity»
Component
-component_id : String
-name : String
-description : String
-version : String
-configurationData : Map<String, String>
+get_details() : Map<String, String>

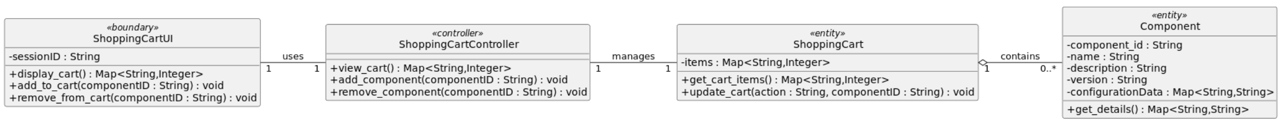
Class Diagram - Use Case #5 (Configure Components)


«boundary»
ComponentConfigUI
-sessionID : String
+display_component_details(componentID : String) : Map<String, String>
+collect_configuration_input() : Map<String, String>
+confirm_configuration() : boolean
uses
1
1
«controller»
ComponentConfigurationController
+retrieve_component(componentID : String) : Component
+validate_configuration(newConfig : Map<String, String>) : boolean
+apply_configuration(component : Component, newConfig : Map<String, String>) : void
manages
1
1
«entity»
Component
-component_id : String
-name : String
-description : String
-version : String
-configurationData : Map<String, String>
+get_details() : Map<String, String>
+update_configuration(newConfig : Map<String, String>) : void

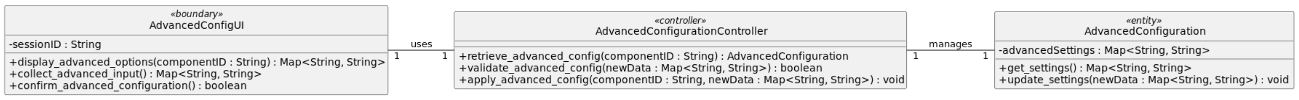
Class Diagram - Use Case #6 (Manage User Profile)


«boundary»
UserProfileUI
-sessionID : String
+display_profile(userID : String) : Map<String, String>
+collect_profile_updates() : Map<String, String>
+confirm_profile_update() : boolean
uses
1
1
«controller»
UserProfileController
+fetch_profile(userID : String) : UserProfile
+update_profile(newData : Map<String, String>) : void
manages
1
1
«entity»
UserProfile
-profileData : Map<String, String>
+get_profile_data() : Map<String, String>
+set_profile_data(newData : Map<String, String>) : void

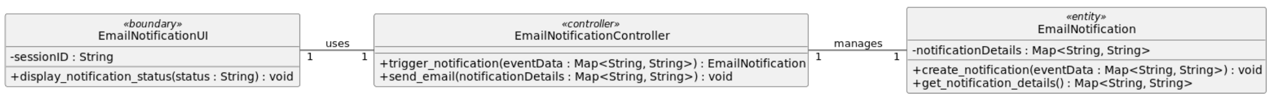
Class Diagram - Use Case #7 (Manage Shopping Cart)


«boundary»
ShoppingCartUI
-sessionID : String
+display_cart() : Map<String,Integer>
+add_to_cart(componentID : String) : void
+remove_from_cart(componentID : String) : void
uses
1
1
«controller»
ShoppingCartController
+view_cart() : Map<String,Integer>
+add_component(componentID : String) : void
+remove_component(componentID : String) : void
manages
1
1
«entity»
ShoppingCart
-items : Map<String,Integer>
+get_cart_items() : Map<String,Integer>
+update_cart(action : String, componentID : String) : void
contains
1
0..*
«entity»
Component
-component_id : String
-name : String
-description : String
-version : String
-configurationData : Map<String,String>
+get_details() : Map<String,String>

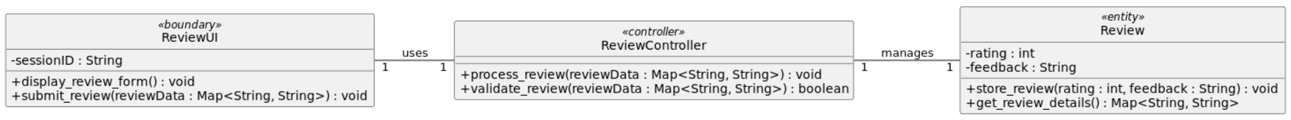
Class Diagram - Use Case #8 (Advanced Configuration)


«boundary»
AdvancedConfigUI
-sessionID : String
+display_advanced_options(componentID : String) : Map<String, String>
+collect_advanced_input() : Map<String, String>
+confirm_advanced_configuration() : boolean
uses
1
1
«controller»
AdvancedConfigurationController
+retrieve_advanced_config(componentID : String) : AdvancedConfiguration
+validate_advanced_config(newData : Map<String, String>) : boolean
+apply_advanced_config(componentID : String, newData : Map<String, String>) : void
manages
1
1
«entity»
AdvancedConfiguration
-advancedSettings : Map<String, String>
+get_settings() : Map<String, String>
+update_settings(newData : Map<String, String>) : void

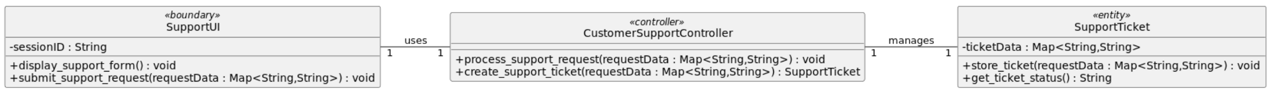
Class Diagram - Use Case #9 (Receive Email Notifications)


«boundary»
EmailNotificationUI
-sessionID : String
+display_notification_status(status : String) : void
uses
1
1
«controller»
EmailNotificationController
+trigger_notification(eventData : Map<String, String>) : EmailNotification
+send_email(notificationDetails : Map<String, String>) : void
manages
1
1
«entity»
EmailNotification
-notificationDetails : Map<String, String>
+create_notification(eventData : Map<String, String>) : void
+get_notification_details() : Map<String, String>

Class Diagram - Use Case #10 (Leave Reviews/Feedback)


«boundary»
ReviewUI
-sessionID : String
+display_review_form() : void
+submit_review(reviewData : Map<String, String>) : void
uses
1
1
«controller»
ReviewController
+process_review(reviewData : Map<String, String>) : void
+validate_review(reviewData : Map<String, String>) : boolean
manages
1
1
«entity»
Review
-rating : int
-feedback : String
+store_review(rating : int, feedback : String) : void
+get_review_details() : Map<String, String>

Class Diagram - Use Case #11 (Access Customer Support)


«boundary»
SupportUI
-sessionID : String
+display_support_form() : void
+submit_support_request(requestData : Map<String,String>) : void
uses
1
1
«controller»
CustomerSupportController
+process_support_request(requestData : Map<String,String>) : void
+create_support_ticket(requestData : Map<String,String>) : SupportTicket
manages
1
1
«entity»
SupportTicket
-ticketData : Map<String,String>
+store_ticket(requestData : Map<String,String>) : void
+get_ticket_status() : String

Project Setup

- The prerequisites:
Ensure the following are installed on your local system:

- [Node.js and npm] (https://nodejs.org/)
- A modern web browser (e.g., Chrome, Edge, Firefox)
- (Optional) Git for version control: [Install Git] (https://git-scm.com/)

Clone the project from GitHub.

- The dependencies:
Install the required node modules:

```
bash
npm install
```

- Firebase Configuration:

- Create a Firebase project from [Firebase Console] (https://console.firebase.google.com/)
- Enable Authentication, Realtime Database, and Storage.
- Add Firebase config details to src/firebase.js:

```
js
const firebaseConfig = {
  apiKey: 'YOUR_API_KEY',
  authDomain: 'your-project-id.firebaseapp.com',
  databaseURL: 'https://your-project-id.firebaseio.com',
  projectId: 'your-project-id',
  storageBucket: 'your-project-id.appspot.com',
  messagingSenderId: 'YOUR_SENDER_ID',
  appId: 'YOUR_APP_ID'
};
```

- Run app on local development server:
Start the development server:

```
bash
npm start
```

Access the app at: [http://localhost:3000] (http://localhost:3000)

- Structure Overview:

```
Software/
├── public/
├── src/
```

```
│   ├── components/
│   │   ├── addcomponent/
│   │   ├── allcomponents/
│   │   ├── dashboard/
│   │   ├── header/
│   │   ├── home/
│   │   ├── …
│   │   └── view/
│   ├── firebase.js
│   ├── index.js
│   └── global.css
├── package.json
├── package-lock.json
└── README.md
```

The major components and core files here:

- `src/index.js`: The entry point of the React app. It renders the root component (typically `App`) into the HTML DOM.

- `src/firebase.js`: Handles Firebase initialization and exports Firebase services like Authentication, Realtime Database, or Firestore.

- `src/global.css`: Contains global CSS styles applied throughout the app.

- `src/components/`: Contains all the modular components. Each folder represents a UI unit or page:

`addcomponent/` -

- `addcomponent.js`: UI and logic to add a new software component to the catalog.
- `addcomponent.css`: Styles specific to the add component form.

`allcomponents/` -

- `allcomponents.js`: Lists all added components from the Firebase database.
- `allcomponents.css`: Styling for the list/grid display.

`dashboard/` -

- `dashboard.js`: The main dashboard users see after login, usually showing summaries or quick actions.
- `dashboard.css`: Dashboard layout and visual styling.

`header/` -

- `header.js`: Contains the site's top navigation bar.
- `header.css`: Header-specific styles.

`home/` -

- `home.js`: Landing page (i.e. welcome screen).
- `home.css`: Styling for the homepage layout.

`login/` -

- `login.js`: Handles user login via Firebase Authentication.
- `login.css`: Styling for login form and error messages.

`searchbar/` -

- `searchbar.js`: A search interface for filtering components based on name, type, etc.
- `searchbar.css`: Search bar styling.

`admin/`, `adminlogin/`, `profile/`, `collections/`, `view/` -

- These folders manage admin features, login/authentication for administrators, user-specific collections, viewing a single component, and user profile information respectively.
- Each contains JS for logic and CSS for layout/design.

- `public/` Folder:
  - ‘index.html`: HTML file where the React app is injected.
  - `manifest.json`, `favicon.ico`, `robots.txt`: Standard files for PWA support and metadata.
  - Lottie Animation JSONs (`loading_anim.json`, etc.): Used for animations in the UI.

- `package.json`: Defines all project dependencies, scripts, and metadata.
- `.gitignore`: Ensures temporary, sensitive, or build files like `node_modules`, `.env`, and `build/` aren’t pushed to GitHub.

Results

The overall technical structure and the broad interaction flow of the Software Component Cataloguing System is given below:

1. Home Page (`home.js`):
Functionality:
- Entry point of the application (`/` route).
- Renders a welcome UI with call-to-action buttons for login.
- Lightweight for fast loading.

Technical Details:
- Uses `React Router` for navigation.
- No authentication required.
- Firebase not initialized here.

2. Login Page (`login.js`):
Functionality:
- Allows users (and admins via separate route) to authenticate via email/password.

Technical Details:
- Integrates with `Firebase Authentication`.
- On successful login: User role is fetched from Firestore or inferred, and then it redirects based on role using `useNavigate`.

3. User Dashboard (`dashboard.js`):
Functionality:
- Primary interface post-authentication.
- Provides navigational access to cataloging features.

Technical Details:
- Renders role-based UI elements.
- Uses React context or props to display session-specific content.
- Pulls user session info from Firebase Auth.

4. Add Component (`addcomponent.js`):
Functionality:
- UI to submit new software components.

Technical Details:
Form with:
- `name`, `description`, `category`, `tags`, `documentation URL`, etc.
- Validates form inputs.
- Submits to `Firebase Firestore` under `components` collection.

5. All Components (`allcomponents.js`):
Functionality:
- Displays a real-time list of all components.

Technical Details:
- Uses Firestore `onSnapshot` listener for live updates.
- Integrated with `searchbar.js` for dynamic filtering.

Data Query:

js
db.collection("components").orderBy("timestamp", "desc")

6. View Component (`viewcomponent.js`):
Functionality:

- Shows detailed metadata of selected component.

Technical Details:

- Fetches component by ID via `useParams`.
- Reads directly from Firestore:
  js
  db.collection("components").doc(componentId).get()
  Allows user to add to collection.

7. Collections Page (`UserCollections.js`):
Functionality:

- User's personalized set of saved components.

Technical Details:

- Collection data stored under a nested Firestore structure:
  json
  users/{userId}/collections/{componentId}
  Uses hooks to fetch user-specific data.

8. Profile Page (`ProfilePage.js`):
Functionality:

- Displays logged-in user's information and session settings.

Technical Details:

- Read from Firebase Auth.
- Logout button linked to `firebase.auth().signOut()`.

9. Admin Panel (`AdminPanel.js`):
Functionality:

- Admin-exclusive interface for managing users and components.

Technical Details:

- Role verified via Firestore field (`role: "admin"`).
- Can perform CRUD operations on all components.
- Uses modal or inline forms for edit/delete.
- Uses Firebase Security Rules to allow admin-only access.

10. Admin Login Page (`adminlogin.js`):
Functionality:

- Special login route for admin users with elevated privileges.

Technical Details:

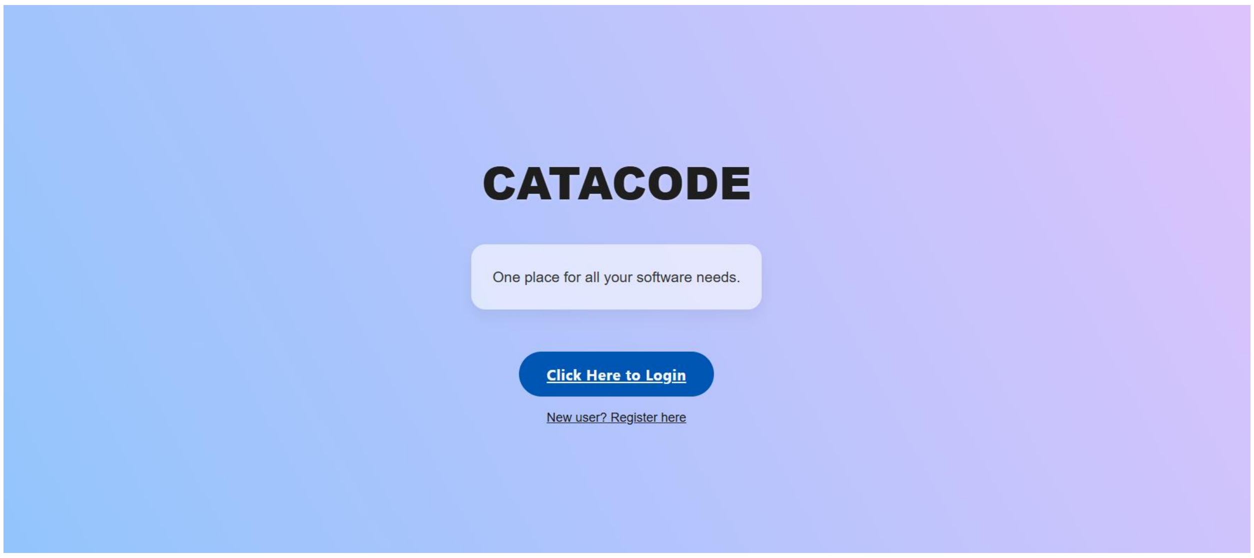
- Identical auth method to user login.
- Role-based route check immediately after login:
  js
  if (user.role === 'admin') navigate('/admin')

11. Loading & Error States:
Functionality:

- Feedback during data load, auth, or error handling.

Technical Details:

- Uses Lottie animations for enhanced UX.

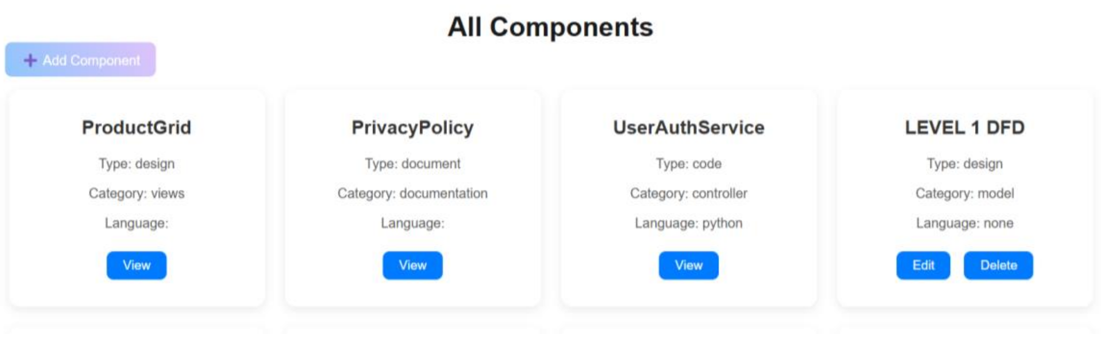


CATACODE

One place for all your software needs.

Click Here to Login

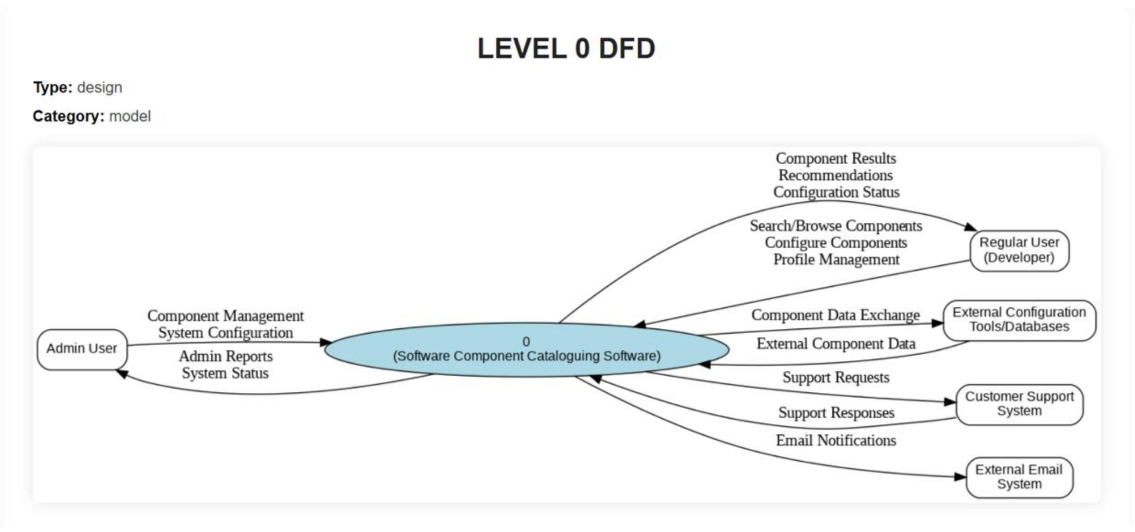
New user? Register here



All Components

+ Add Component

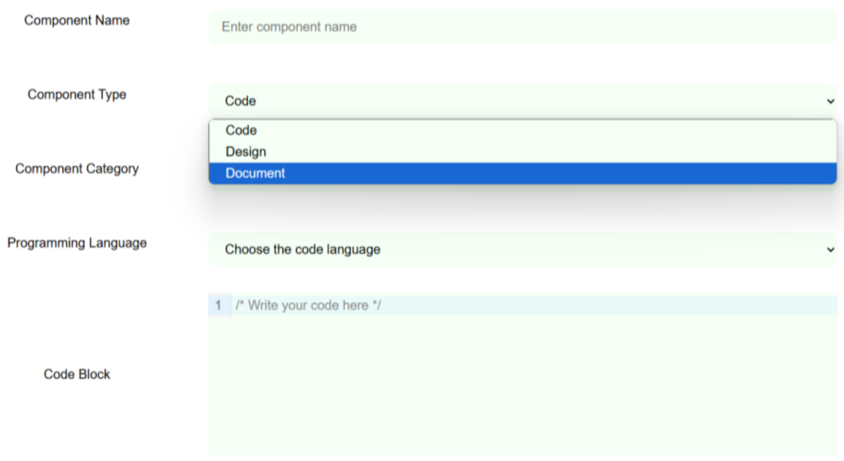
ProductGrid	PrivacyPolicy	UserAuthService	LEVEL 1 DFD
Type: design	Type: document	Type: code	Type: design
Category: views	Category: documentation	Category: controller	Category: model
Language:	Language:	Language: python	Language: none
View	View	View	Edit Delete

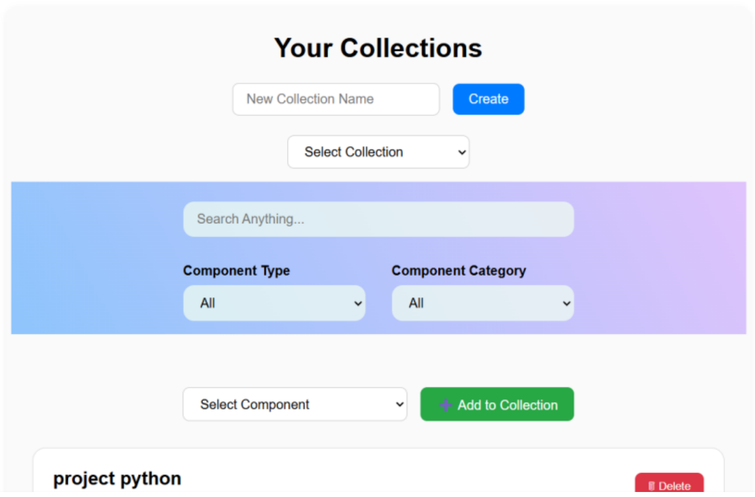


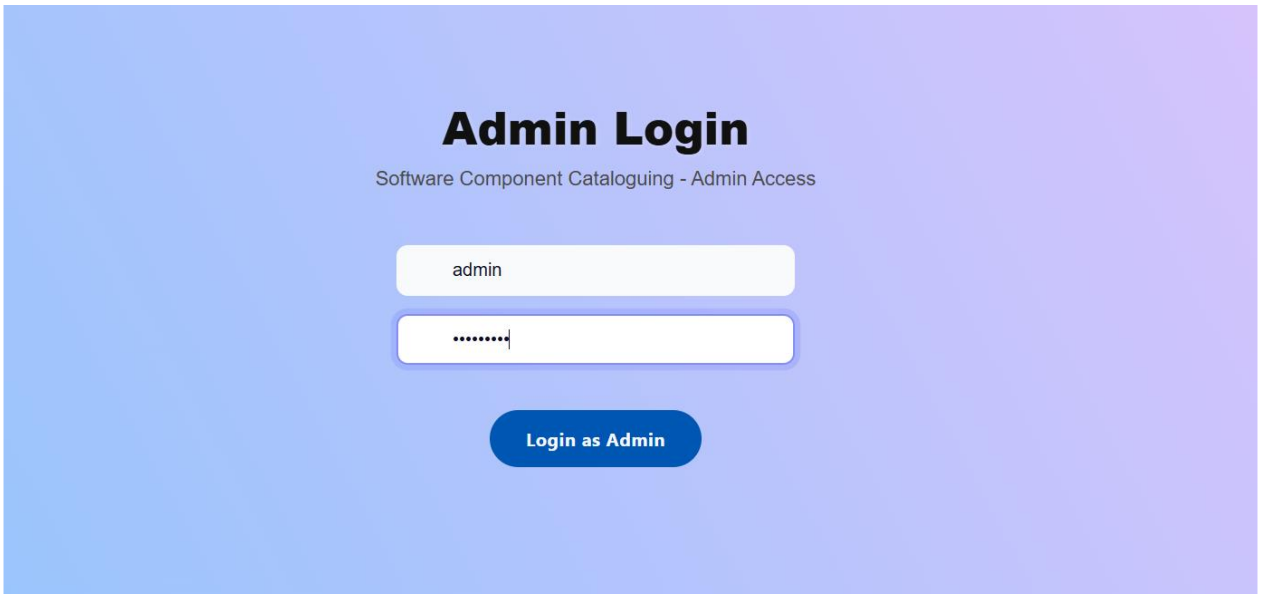
LEVEL 0 DFD

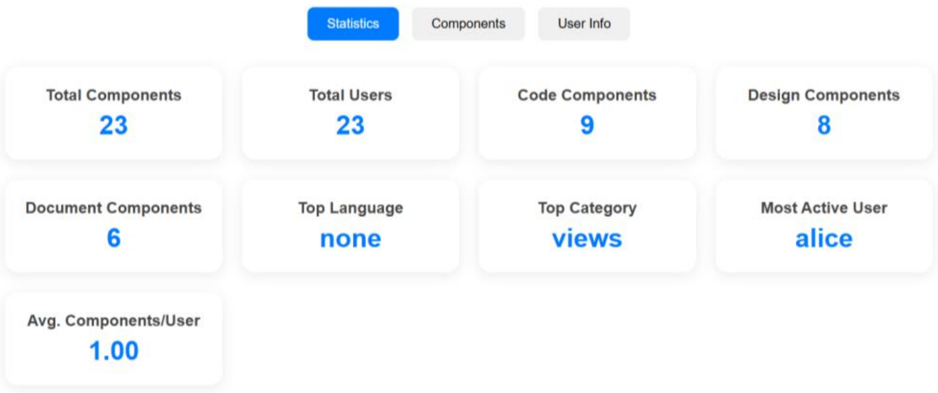
Type: design

Category: model


Component Name
Enter component name
Component Type
Code
Code
Design
Document
Component Category
Programming Language
Choose the code language
/* Write your code here */
Code Block


Your Collections
New Collection Name
Create
Select Collection
Search Anything...
Component Type
All
Component Category
All
Select Component
Add to Collection
project python
Delete


Admin Login
Software Component Cataloguing - Admin Access
admin
•••••••••
Login as Admin



Total Components 23

Total Users 23

Code Components 9

Design Components 8

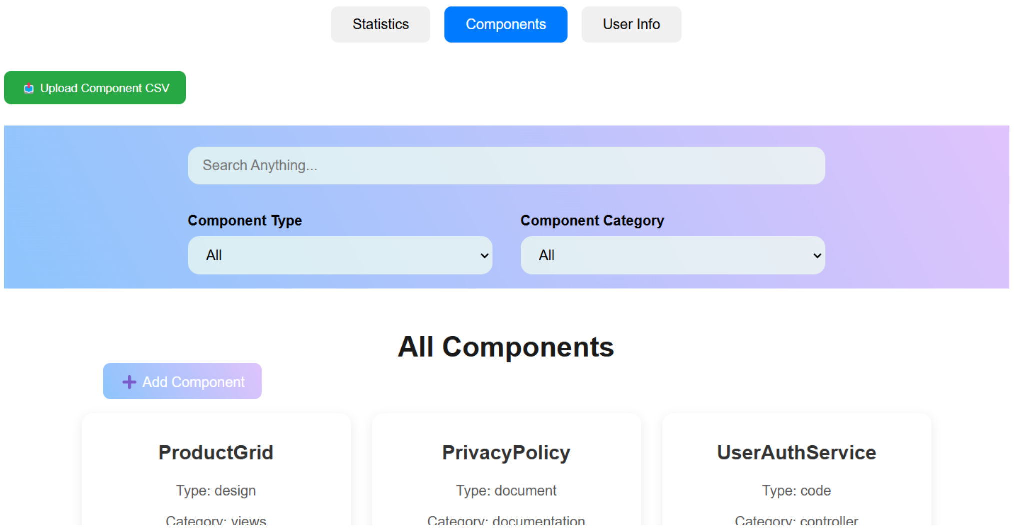
Document Components 6

Top Language none

Top Category views

Most Active User alice

Avg. Components/User 1.00



All Components

ProductGrid
Type: design
Category: views

PrivacyPolicy
Type: document
Category: documentation

UserAuthService
Type: code
Category: controller

<u>Summary</u>

The Software Component Cataloguing Software was successfully implemented.
