Bellisimo

Software Requirements Specification

Elizabeth (EF) Bode 14310156

September 6, 2017

Contents

1 Business Requirements

1.1 Background

Bellisimo is a company aimed at providing an online platform for customers to browse clothing as well as food catalogues provided by the business located in Hatfi

eld. Information about specials and promotions will be published on the online platform.

1.2 Vision

The core of the system will be catalogues of items and their prices. Since Bellisimo is involved in clothing and food, the catalogues will have to ensure that these lines are well maintained. Sales and specials in each line will have to be accounted for and managed.

1.3 Scope

Bellisimo will be a web-based application that provides the functionalities of an online catalogue for food and clothing. The application will be accessible from anywhere using a specified URL. The goal of the system is to provide Bellisimos customers with the latest information on their food and clothing products and the specials associated with them.

1.4 System Functions

- An admin interface to allow for adding, removing and updating product items and specials
- An anonymous user interface to allow for browsing, searching and filtering products and viewing the details of a product and the specials that are running
- An admin user should be able to login, logout and manage their profile

1.5 Assumptions

- An admin user has to be preloaded onto the system, the system doesnt cater for adding admin users to it

1.6 Constraints

- The system must be implemented using the technologies stated in the provided specification
- Only open source software should be used for the system
- The system must be complete by the 6th September

1.7 Methodology

This system will be implemented using the Agile methodology as this methodology allows flexibility in regards to changing requirements. Scrum will also be incorporated into the Agile methodology as it provides a suitable framework for project management. Milestones and issues will be set up on Github to keep track of tasks that are outstanding. These milestones and issues will be linked to waffle.io in order to provide an interactive interface showing completed tasks and outstanding tasks. This will also be linked to burndown.io as it provides a burndown chart for your project showing overall progress according to time constraints.

2 System Requirements and Design

2.1 Architectural Design

2.1.1 Overview

The web application will consist of two subsystems that communicate via HTTP using REST Framework. The Java/Spring Boot application will be known as the "backend" application. The HTML5/Angular2 application will be known as the "frontend" application. The backend application is expected to communicate with the database and use Hibernate which can be imported into Maven, a dependency management tool whereas the frontend application will be hosted in the browser and NodeJS is expected to manage packages required for the application to run successfully.

2.1.2 Micro-services Architecture

The micro services implementation will consist on n spring boot applications where n = no. of modules or components in the system. These applications will be integrated by using spring cloud and Netflix OSS Zuul Integration. The integration server will be regarded as one component of the n components to be implemented.

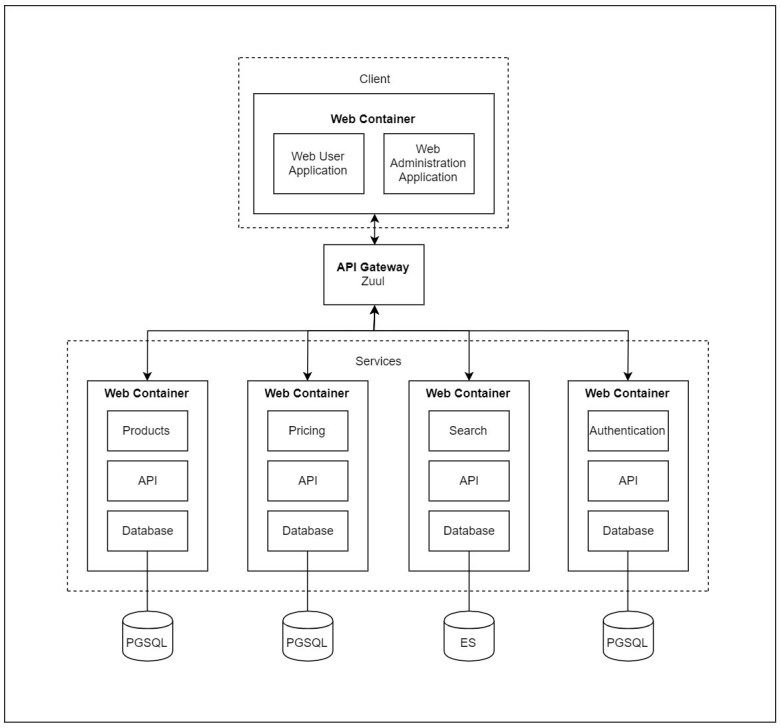


Figure 1: System components diagram

2.1.3 Non-Functional Requirements

- **Performance**

  System performance can be defined as the amount of work the system can perform in a measured time interval. The time interval is normally measured in seconds, where the amount of work can be defined as the throughput, latency or data transmission time.

  Requirements:

  - Function calls must be timed and benchmarked and this data should be logged
  - Network responses should be cached on server side to lighten the load on database as well as decreasing the round trip time of request - response
  - The system should never hang because of poor system performance, it should only be a result of a poor network connection

- **Integrability**

  Integrability refers to a systems ability to seamlessly integrate new modules into the existing system regardless of the varying technologies they were implemented in.

  Requirements:

  - The system should allow technology neutral importing and exporting of data
  - The back-end system should integrate with a desktop web client
  - The system should be able to integrate with different back-end authentication services
  - Modules should be able to be swapped in and out regardless of the technologies they are implemented in

- **Maintainability**

  The system is to be designed in such a way that it is easily updated, modified or extended by the client in the future. In order to achieve these requirements, design patterns and best practices will be used to ensure uniformity and modularity across the system.

  Requirements:

  - All code should be documented in the applicable language documentation framework, such as JavaDocs for a Java based system, DOxygen for a C/C++ based system, etc.
  - A specified coding style should be adhered to so as to enforce readable code that is consistent and easier to maintain
  - System should be separated in distinct, concise and independent modules relating to separate concerns, to allow for easier maintenance

- **Scalability**

  Scalability refers to the application in questions ability to handle an above normal workload for extended time periods and the mechanisms employed to facilitate this.

  Requirements:

  - The system should be able to handle increased workloads towards the end of the month when people are paid
  - The system should be able to facilitate 100 concurrent user sessions at any time

- **Reliability**

  System reliability refers to the probability of the system performing an operation to a satisfactory standard at or for a specified time in a specified environment.

  Requirements:

  - The system should never go down as it could potentially cause Bellisimo to lose several customers
  - The system should have the ability to swap out modules without causing system downtime

- **Security**

  Security comprises of data security authentication and authroization. Authentication refers the systems ability to identify a user to be who they are. Authorization refers to the systems ability to provide the user in question with access to only the system functions that they have the authority to perform. Data security refers to data only being accessible through authorized channels.

  Requirements:

  - System should be resistant to SQL injections
  - Password hosting with a unique salt for each user should be used
  - A key derivation function should be used with passwords

- Database access must require authentication

- **Accessibility**

  System accessibility refers to the system being accessible to whomever requires it when they require it.

  Requirements:

  - The system should have a web interface thats hosted on a server so customers can access it wherever whenever
  - The system wont take into account any individual with any type of disability hindering their interaction with a web-based application

- **Flexibility**

  Flexibility refers to the systems ability to continue to function regardless of internal swapping of components. Furthermore, it refers to the systems ability to be able to adapt to changing constraints.

  Requirements:

  - The system should be decoupled from the database technology it uses and allow the client to select and change the database it uses in future
  - Authentication mechanisms used should be decoupled from the system, allowing them to be interchangeable
  - Modules should be decoupled from one another, allowing the system to be extensible without a break in service which is achieved by integrating new modules and swapping out existing ones

2.1.4 Architectural Patterns

- **Authentication Enforcer Pattern**

  The authentication enforcer pattern provides centralized management for authentication, to verify the identity of users and encapsulate the details of authentication. This pattern enforces that security checks are separated from business logic, ensuring cleaner and more robust code, promoting easier maintenance. Since authentication is centralized, this pattern also allows changes to be easily implemented in regards to the authentication procedure, allowing one to substitute authentication procedures without having to recode the entire system.

  Reasons for selecting this pattern:

  - This pattern promotes reliability because authentication is centralized allowing system modules to be swapped in and out or upgraded without affecting the systems reliability, as long as the authentication mechanism provides the same service to the system
  - This pattern promotes flexibility as it allows for integration of varying technologies, giving the developer freedom in regards to the technologies they want to use for authentication
  - This pattern promotes maintainability and integrability as authentication mechanisms are decoupled and interchangeable and can be used wherever the system requires authentication by simply plugging in the appropriate module

- **Client-Server Architectural Pattern**

  A network technology in which each computer connected on the network is either a server or client is called a client server architecture. An interface is provided by the client to allow a user to request services from the server and to display the results the server returns.

  1. Servers: are powerful computers which are used to process the client requests
  2. Client: is simply a computer connected to the network which is used to send requests for resources to the server

Reasons for selecting this pattern:

- This pattern promotes maintainability and flexibility as it allows decoupling of business logic and human adapter modules
- It enables system usability by not requiring users to incur large downloads to make use of the system
- The pattern promotes maintainability and reliability, since developers are able to upgrade and update individual system modules without affecting any other modules

- **Layered Architectural Pattern**

A design pattern in which software is divided up into individual layers by functionality. The layers interact with one another via requests and responses, and there are typically four of these layers:

1. Presentation/view: The user interface that the user interacts with, such as a desktop client
2. Application/controller: The service API that the presentation layer interacts with in order to perform system functions
3. Business logic: The functions that get called by the service API, to interact with system data
4. Data access: The layer that provides data access to the business logic layer

Reasons for selecting this pattern:

- This pattern promotes reliability, flexibility, integrability and maintainability as it enables trivial swapping of groups of modules
- The pattern promotes maintainability as it allows decoupling of the database and authentication modules from the rest of the system

- **Representational State Transfer (REST) Architectural Pattern**

REST is an architectural pattern for designing networked applications. The idea is instead of using complex mechanisms such as SOAP, CORBA or RPC to connect between machines or to make calls between machines, HTTP is used. A REST service is:

1. Language-independent (C# can communicate to Java, etc.)
2. Can be used with firewalls
3. Platform-independent (MAC, Windows, UNIX)
4. Standards-based (runs on top of HTTP)

Reasons for selecting this pattern:

- It promotes flexibility since it allows platform independent communication between business logic and human adapter modules
- It promotes maintainability as it is language independent and allows for easy migration of services to different technologies while keeping communication methods well defined
- It satisfies the performance requirement as REST network responses can be cached

- **Services Orientated Architecture/ Micro-services Architecture**

An architectural pattern in software design in which system use cases are divided into one or more service operations, and these service operations are then implemented, either individually or combined with other service operations, by a reusable SOA service. The SOA architecture itself comprises five layers:

1. Consumer interface: The GUI for end users accessing application services
2. Business process: The representation of system use cases
3. Services: The consolidated inventory of all available services
4. Service components: The reusable components used to build the services, such as functional libraries

5. Operational system: Contains data repository, technological platforms, etc.

This architectural pattern provides an aggregated collection of services which implement the use cases of the system. The user or developer themselves can choose which of these available services to call on.

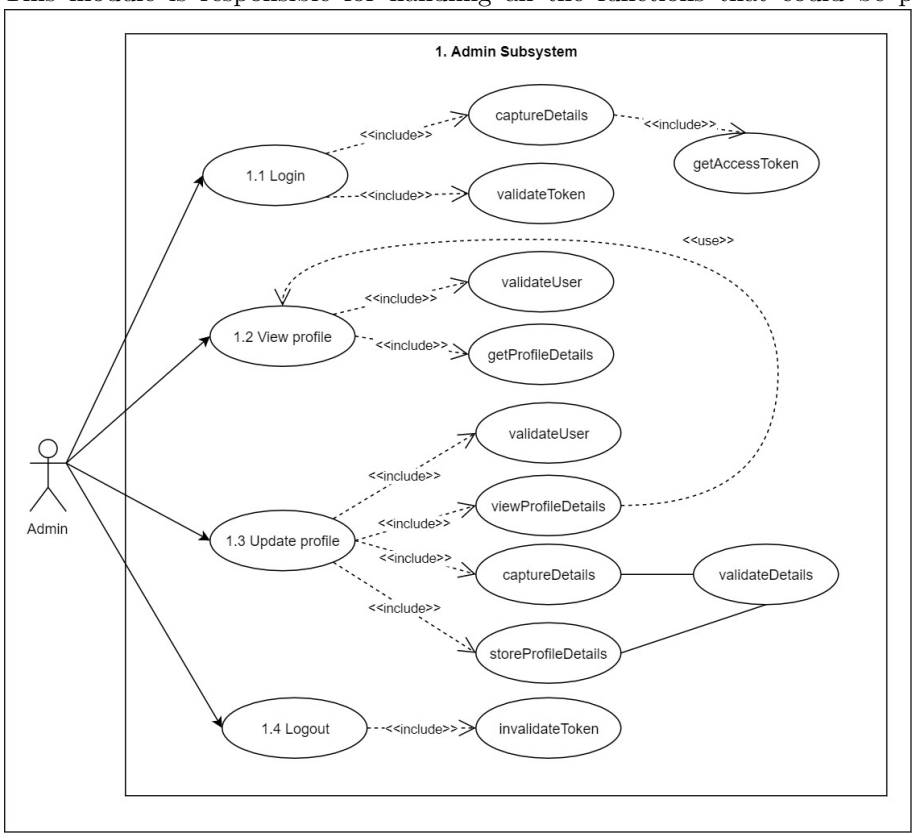
Reasons for selecting this pattern:

- This pattern promotes system reliability and scalability as it focuses on decoupling service objects so they can be distributed on different machines and can be implemented using varying technologies
- This pattern promotes system flexibility as the decoupling of service objects allows for interchangeable service providers, ensuring the system is never restricted by its service providers if conditions change and require upgrades or updates
- This pattern satisfies all the same quality requirements as the layered architecture pattern

3 System Modules

3.1 Admin Subsystem

This module is responsible for handling all the functions that could be performed by the administrator.



3.1.1 Login

- **Description:**
  This use case will be used by the REST web client to initiate administrator login via the back-end service.
- **Preconditions:**
  1. The user is registered as an administrator on the system
  2. The administrator doesnt have a valid login token
- **Post conditions:**
  1. The administrator is logged into the system
  2. The administrator has a valid login token

3.1.2 View profile

- **Description:**
  This use case will be used by the REST web client to view the profile of the administrator that is currently logged in.
- **Preconditions:**
  1. The administrator is logged into the system
- **Post conditions:**
  1. The administrators profile page will be displayed

3.1.3 Update profile

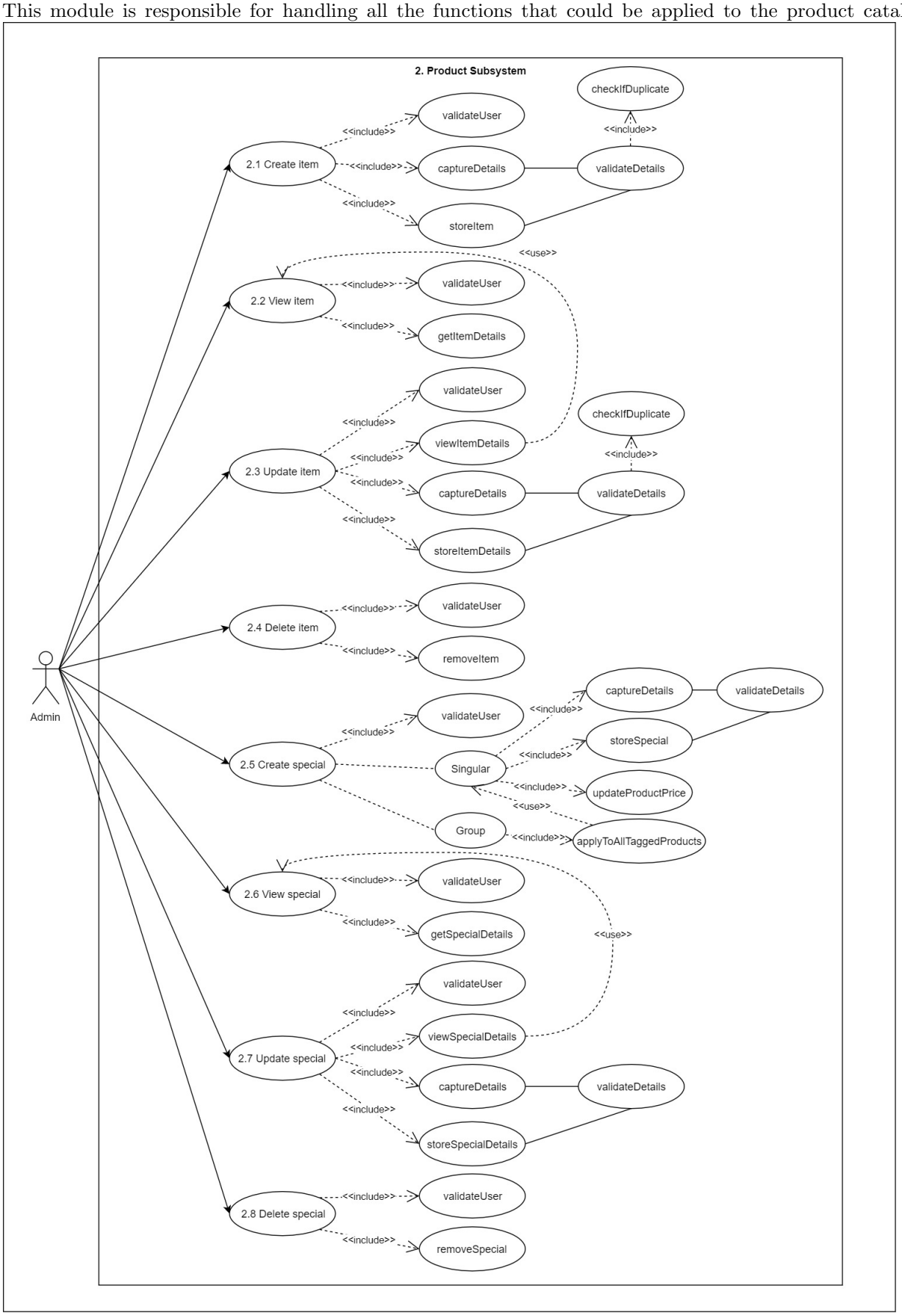
- **Description:**
  This use case will be used by the REST web client to view and update the profile of the administrator that is currently logged in.
- **Preconditions:**
  1. The administrator is logged into the system
- **Post conditions:**
  1. The administrators profile details will be updated with valid information in the database
  2. The administrators profile page will be displayed with the updated information

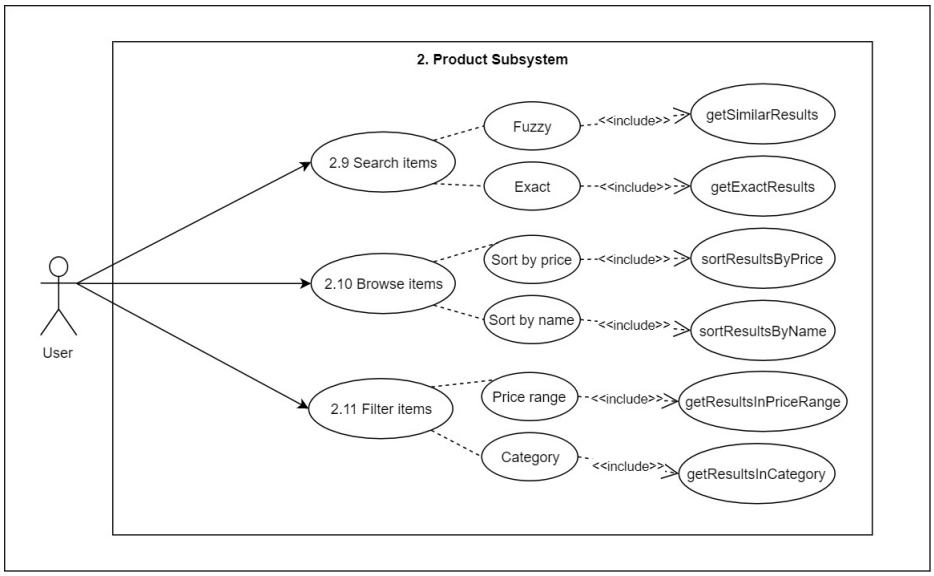
3.1.4 Logout

- **Description:**
  This use case will be used by the REST web client to log an administrator out of the system.
- **Preconditions:**
  1. The administrator is logged into the system
- **Post conditions:**
  1. The administrator will be logged out of the system

3.2 Product Subsystem

This module is responsible for handling all the functions that could be applied to the product catalogue.





3.2.1 Create item

- **Description:**
  This use case will be used by the REST web client to create a product item.
- **Preconditions:**
  1. The administrator is logged into the system
- **Post conditions:**
  1. The non-duplicate product is created in the database with valid details

3.2.2 View item

- **Description:**
  This use case will be used by the REST web client to display a products details.
- **Preconditions:**
  1. The administrator is logged into the system/the user is exploring the system anonymously
  2. The product exists in the database
- **Post conditions:**
  1. The details of the product are displayed

3.2.3 Update item

- **Description:**
  This use case will be used by the REST web client to update the details of a product.
- **Preconditions:**
  1. The administrator is logged into the system
  2. The product exists in the database
- **Post conditions:**
  1. The non-duplicate products details will be updated with valid information in the database
  2. The products details will be displayed with the updated information

3.2.4 Delete item

- **Description:**
  This use case will be used by the REST web client to remove a specific product from the system.
- **Preconditions:**
  1. The administrator is logged into the system
  2. The product exists in the database
- **Post conditions:**
  1. The product is removed from the system and the database
  2. The products page is displayed with the products that are currently in the system

3.2.5 Create special

- **Description:**
  This use case will be used by the REST web client to create a product special. This use case handles both group and singular specials. Group specials according to tags will initiate this use case for every product that has the specified tag.
- **Preconditions:**
  1. The administrator is logged into the system
  2. The product/tag exists in the database
- **Post conditions:**
  1. The special is created in the database with valid details
  2. The price of the product/s reflects the specials discount

3.2.6 View special

- **Description:**
  This use case will be used by the REST web client to display a specials details.
- **Preconditions:**
  1. The administrator is logged into the system/the user is exploring the system anonymously
  2. The special exists in the database
- **Post conditions:**
  1. The specials details are displayed

3.2.7 Update special

- **Description:**
  This use case will be used by the REST web client to update the details of a special.
- **Preconditions:**
  1. The administrator is logged into the system
  2. The special exists in the database
- **Post conditions:**
  1. The specials details will be updated with valid information in the database
  2. The specials details will be displayed with the updated information

3.2.8 Delete special

- **Description:**
  This use case will be used by the REST web client to remove a specific special from the system. This use case can either be initiated by the administrator or by time. If the end date of a special has passed, then the special must be deleted automatically.
- **Preconditions:**
  1. The administrator is logged into the system/the end date of the special has passed
  2. The special exists in the database
- **Post conditions:**
  1. The special is removed from the system and the database
  2. If deleted by admin: The specials page is displayed with the products that are currently in the system

3.2.9 Search items

- **Description:**
  This use case will be used by the REST web client to apply either a fuzzy or exact search to the products catalogue and display the products retrieved.
- **Preconditions:**
  1. Products exist in the database
- **Post conditions:**
  1. The results retrieved by the search are displayed

3.2.10 Browse items

- **Description:**
  This use case will be used by the REST web client to display all the products that are available on the system. It also provides the functionality to sort the products by price or name in ascending or descending order.
- **Preconditions:**
  1. Products exist in the database
- **Post conditions:**
  1. The products are displayed according to product selection and sorting preference selection

3.2.11 Filter items

- **Description:**
  This use case will be used by the REST web client to filter the product results displayed to the user. The results can either be filtered by price range or by category.
- **Preconditions:**
  1. Products exist in the database
- **Post conditions:**
  1. The products are displayed according to the filter applied

4 System Services

4.1 Search Service

This service will be responsible for the search functionality. It will use an ElasticSearch database as it is known for quick search capabilities and provides the ability for data analysis, which could be of benefit to the business.

4.2 Authentication Service

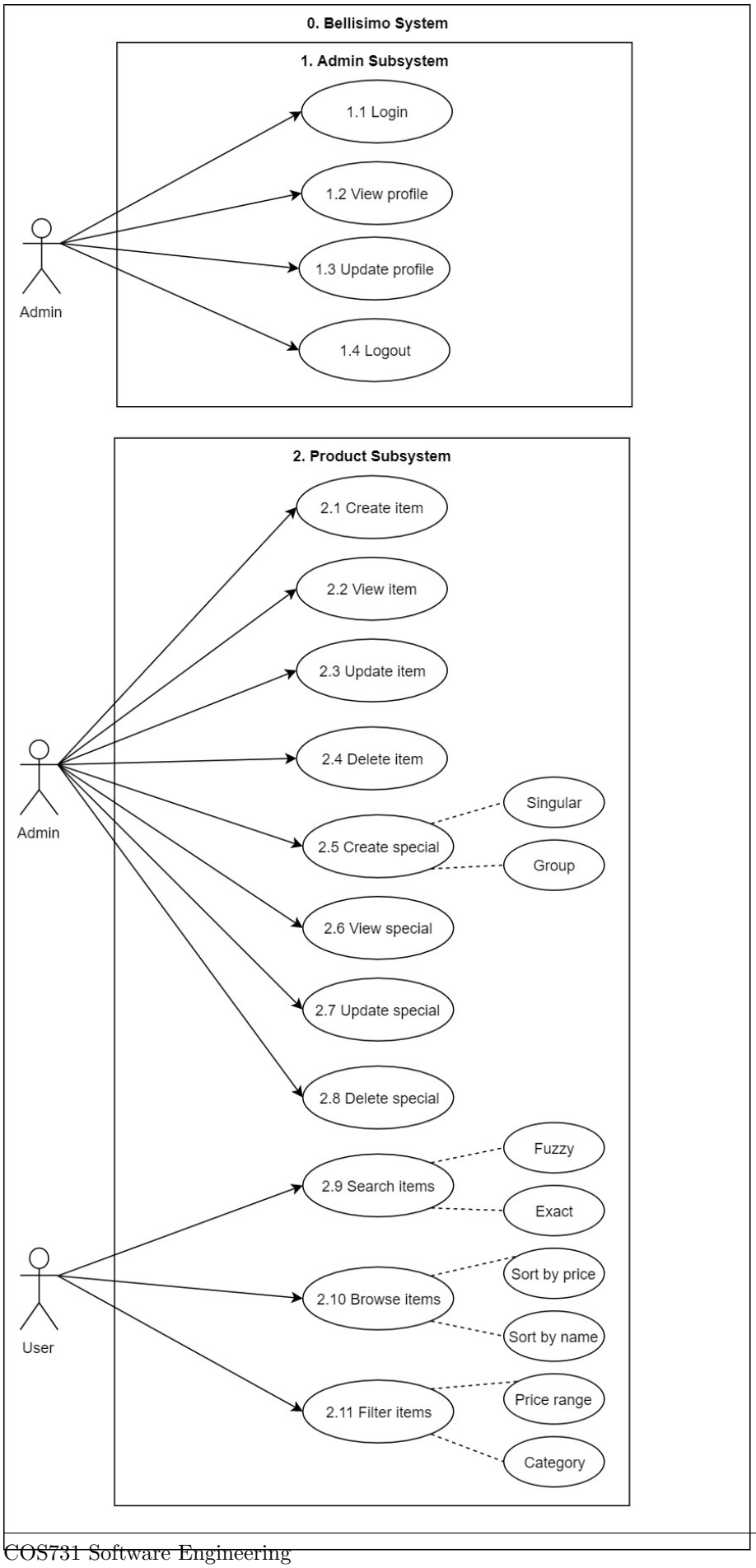
This service will be responsible for handling administrator authentication. This includes hiding privileges from anonymous users.

4.3 Product Service

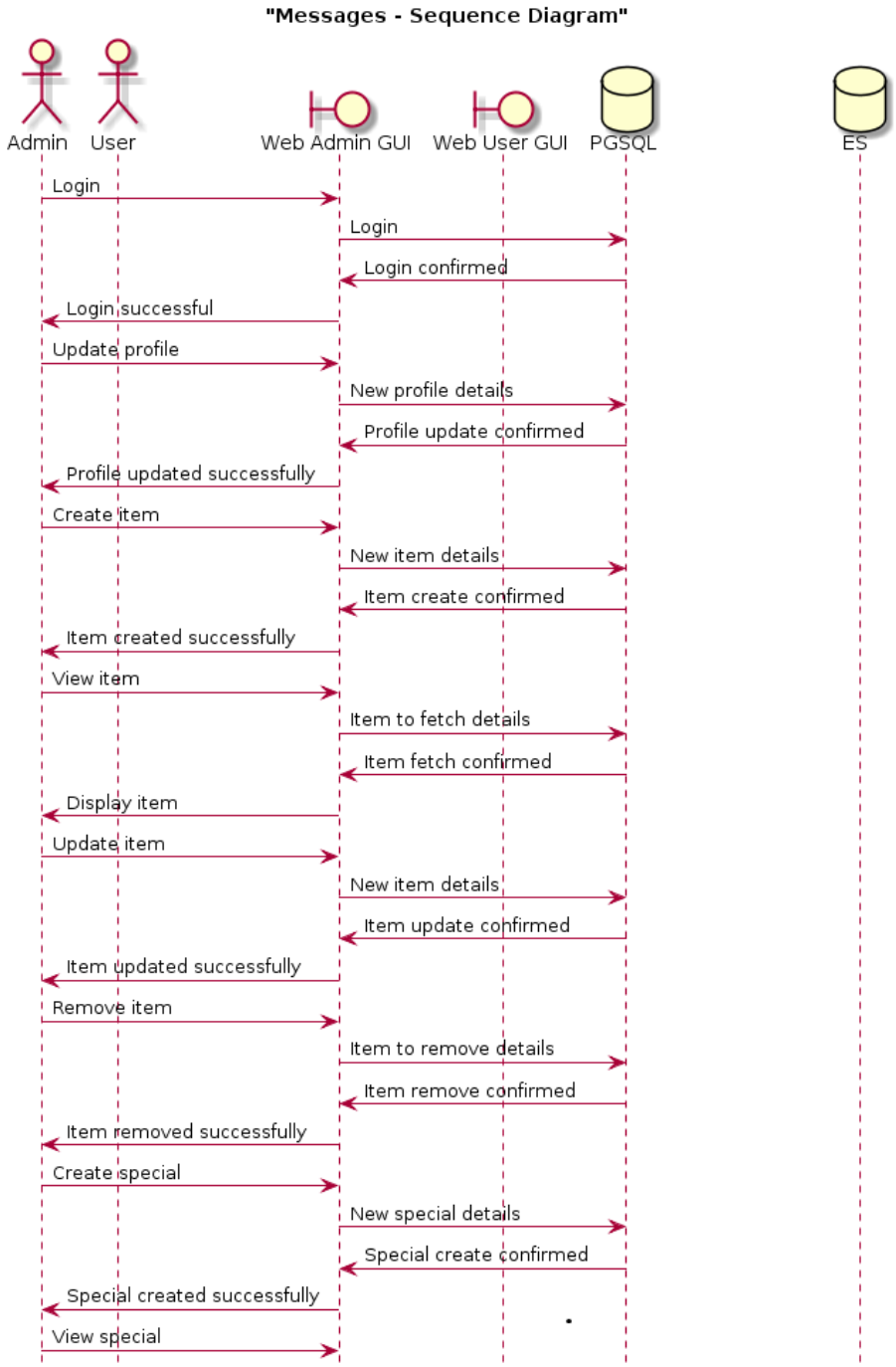
This service will be responsible for the all product requests. Such as viewing, creating, updating and deleting products and viewing, creating, updating and deleting specials. It will also handle the filtering by price range and category and the sorting by price and name.

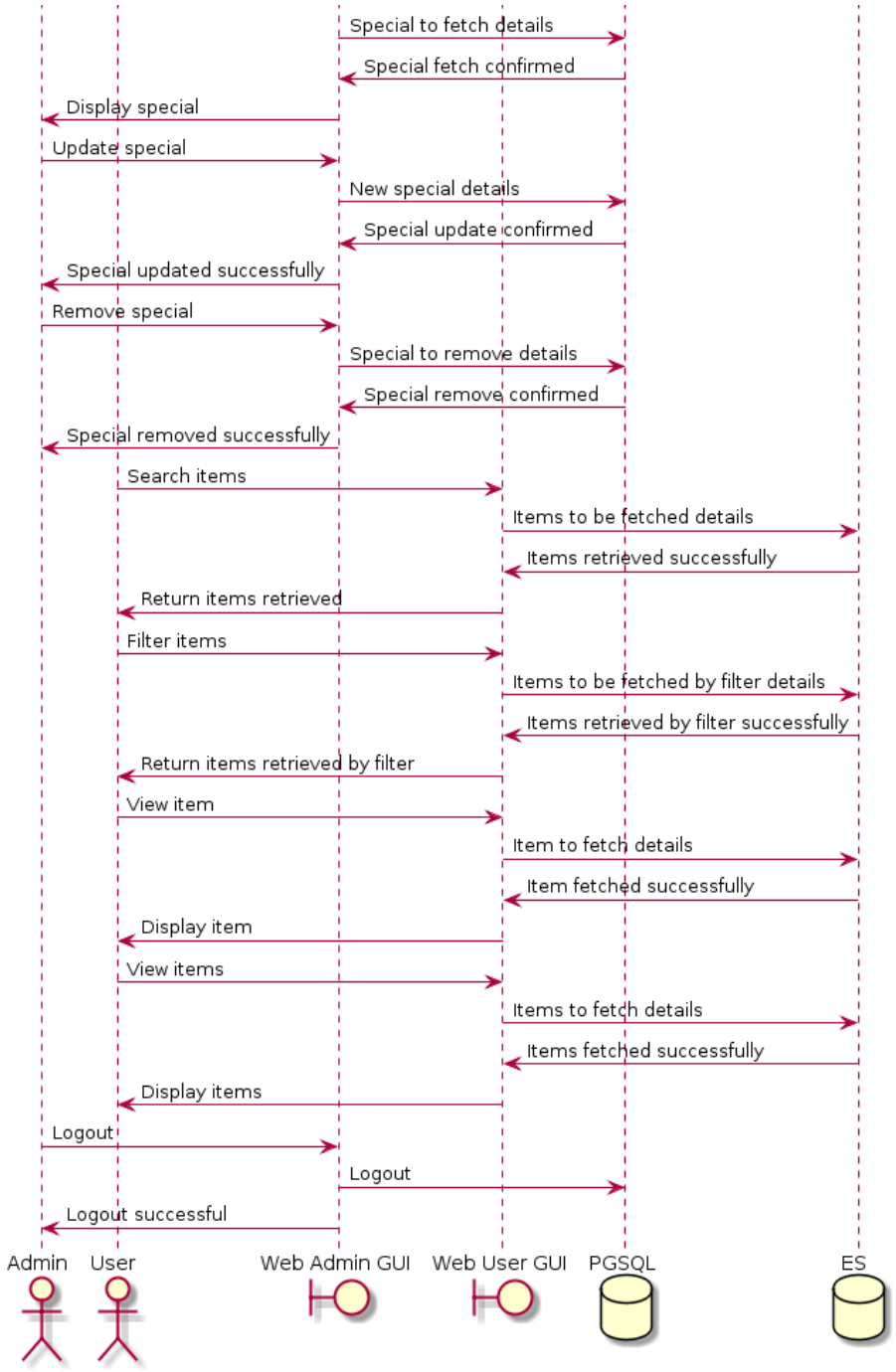
5 System Models

5.1 System Use Case Diagram

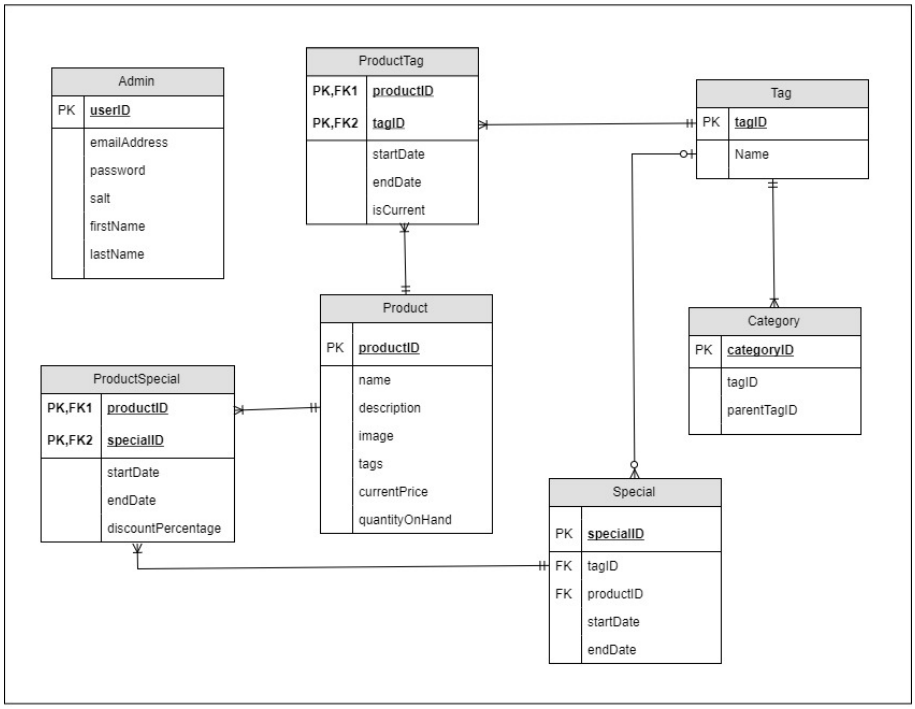


5.2 System Sequence Diagram




Special to fetch details
Special fetch confirmed
Display special
Update special
New special details
Special update confirmed
Special updated successfully
Remove special
Special to remove details
Special remove confirmed
Special removed successfully
Search items
Items to be fetched details
Items retrieved successfully
Return items retrieved
Filter items
Items to be fetched by filter details
Items retrieved by filter successfully
Return items retrieved by filter
View item
Item to fetch details
Item fetched successfully
Display item
View items
Items to fetch details
Items fetched successfully
Display items
Logout
Logout
Logout successful
Admin
User
Web Admin GUI
Web User GUI
PGSQL
ES

5.3 Domain Model



1. **Admin**
   The Admin table is responsible for storing the credentials of system administrators. In order to be added as a system administrator, you have to be added to the database by Bellisimo's owner.

2. **Product**
   The Product table is responsible for storing all the product details.

3. **Tag**
   The Tag table is responsible for storing the various tags that exist for classifying products, for example, Food.

4. **Category**
   The Catgeory table is responsible for storing the hierarchy of categories offered by Bellisimo. The categoryID is unique to a category. The tagID identifies a tag node, which the hierarchy consists of. The parentTagID holds the value of the tag node's parent. Meaning, if the parentTagID is null, then it is the root of the hierarchy. For example, given that a shirt falls under Clothes ¿ Women ¿ Blouses, with the Blouse tag being retrievable from the Product's tag field, the hierarchy of tags can be built from retrieving Blouse's parent until its parent is null and we have reached the parent.

5. **Special**
   In the Special table, only one of either productID or tagID will be filled in as it can be either a singular or a group special.

6. **ProductTag**
   The ProductTag table is responsible for storing singular product tags. For example, a specific dress would belong under Woman ¿ Dresses so the ID for the Dresses tag would be stored for that dress so that the hierarchy of categories can be built from the product upward, in case it is needed during implementation.

7. **ProductSpecial**
   The ProductSpecial table will hold singular specials for products, if a special is applied to a tag, an entry will be created for every product that falls within that tag in the product special table.