Bellisimo

Software Requirements Specification

Elizabeth (EF) Bode 14310156

August 8, 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.

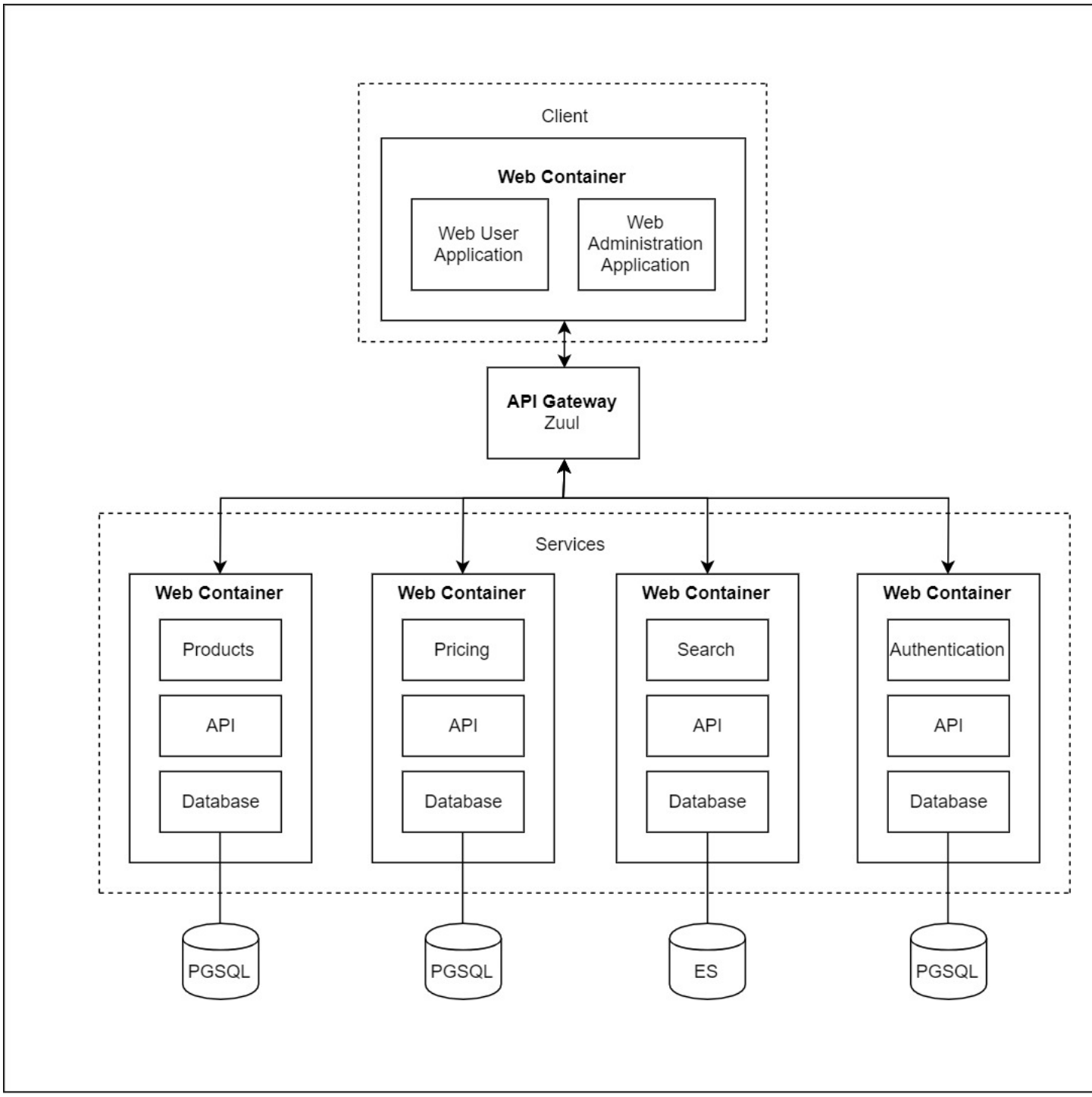

Figure 1: System components diagram

2.1.3 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