SOFTWARE REQUIREMENTS SPECIFICATION

for

Experis Academy Case: Boxinator Project

Version 1.3
Winter 2022

By Nicholas Lennox & Craig Marais

Noroff Accelerate AS

March 2022

Contents

1. Revision History

Name	Date	Reason For Changes	Version
Initial Version	15.07.2020	Created Document	0.1
Autumn 2020	09.09.2020	Revision	1.0
Autumn 2020	29.09.2020	Corrections	1.1
March 2021	01.03.2020	Clarifications and API requirements	1.2
Winter 2022	04.03.2022	Update dates	1.3

2. Introduction

2.1. Purpose

This SRS (Software Requirements Specification) describes the software product to be created by one of the candidate groups in the *Experis Academy Remote* contingent of the *Fullstack Development* short course.

2.2. Document Conventions

For the purposes of this document the following definitions shall apply:

- The *course* refers to the *Fullstack Development* short course as currently offered at *Experis Academy Remote*; specifically the Winter 2022 intake of said course.
- A *candidate* refers to an individual currently enrolled in the course. The plural *candidates* refers to the candidates as they are arranged in groups of four individuals for this case.
- The *software* refers to the final software product to be created.
- *Mentors* refer to the industry experts giving their time to assist the candidates during the case period.

2.3. Intended Audience and Reading Suggestions

This document is intended for use by the mentors and the candidates. Mentors and candidates should read this document and familiarize themselves with the specifications of the software to be created.

2.4. Project Scope

The primary purpose of the software to be a capstone experience for the candidates; they are expected to utilize a selection of possible development options to produce a single software solution that demonstrates their capabilities as developers. The candidates must produce a software solution that is considered a final product. The software is to be produced over a period of three weeks.

2.5. Delivery

A deployed version of the software must completed by the morning of the 31^{st} of March or 1^{st} of April 2022. Nearing the completion of the software, candidates will be required to present their solutions twice. A mock presentation will take place on the 29^{th} of March or 30^{th} of March 2022, and the final presentation will take place on the 31^{st} of March or 1^{st} of April 2022.

2.6. References

Candidates should find the following documents on the Noroff Learning Management System[1] (*Moodle*) site:

- Group membership lists with mentors assigned.
- A copy of this document.

[1] `https://lms.noroff.no/`

3. Requirements Specifications

This case describes an application designed for calculating the shipping cost for mystery boxes to specific locations around the world. It is a web application, using a RESTful API to communicate with a server.

3.1. Product Perspective

For this case, the candidates are expected to design and develop a distributed software solution that gives access to a portal where they can create orders for mystery boxes to be shipped to addresses around the world. Users will be able to create accounts to track ongoing and previous shipments. An administrator has access to a portal to change the metadata of the shipping process, this being the countries that can be shipped to and the relative costs of shipping. The candidates are expected to write unit/component tests for both the back-end and front-end. The packages to do this will differ depending on the frameworks used.

The system comprises of the following components:

- A web front-end for customers
- A web front-end for administrators
- An API that receives data from the front-end and communicates with the database
- A relational database

3.2. Front-end Requirements

Using modern design concepts/frameworks, the system should be functional on both desktop computers and mobile devices.

The front-end should be built using one of the following modern JavaScript frameworks:

1. React
2. Vue.js
3. Angular

3.2.1. FE-01: Login page

This is a very simple login page where both users and administrators login. Users and administrators login with an email and password combination to authenticate themselves. There should be an option to register as a new user visible on the login screen.

3.2.2. FE-02: Register page

If a user selects the option to register themselves they should be presented with a form to populate with the following information:

- First Name - required
- Last Name - required
- E-mail - required, validate on client side
- Password - required, strong, and must be confirmed by repetition
- Date of birth
- Country of residence
- Zip code / postal code
- Contact number

All fields should contain validation to ensure they are in the correct format based on the descriptions stated above. The system must dynamically indicate any invalid entries. A confirmation e-mail should be sent before an account becomes activated.

Technical Note: Country

The user can register as residing in **any** country, but Boxinator services use Norway, Sweden, Denmark as source countries. There should be a flat (standard) fee of 200Kr for packages sent between these countries.

It should cost the same to send a parcel from Stockholm (Sweden) to Oslo (Norway) as it would to send a parcel from Copenhagen (Denmark) to Oslo (Norway). For countries outside of the listed ones, an additional fee is added. These additional fees are calculated based on the package weight and a country multiplier, this is detailed below.

3.2.3. FE-03: Main page

Once a user has logged in, they are presented with a simple page which shows any shipments they have under way and recently completed shipments. There should be an option to add a new shipment. When a new shipment is added a modal should be displayed with the following fields:

- Receiver name
- Weight option (kg)
- Box colour (colour picker)
- Destination Country

The boxes can be coloured to match any rgba value. The weight option is to select which tier of mystery box is being sent. This results in various predefined weights for the various options.

The **basic** tier is a 1Kg box, the **humble** tier is a 2Kg box, the **deluxe** tier is a 5Kg box, and the **premium** tier is a 8Kg box.

The user can then submit this and it is saved to the database. All fields should be validated in ways that are intuitive (e.g. valid RGB(A) values).

The view of the finished and CREATED shipments are displayed in a table, ordered by date. Which each field captured being a column in the table, with the colour column being the actual colour and not the RGB values.

The cost is shown underneath and will be stored in the database (for historical accounting purposes), which is the weigh in kg multiplied by the country multiplier in addition to the 200Kr flat fee, resulting in the following formula: 'Total cost = Flat fee + (weight*multiplier)'. For example, shipping a deluxe package to Germany (with a multiplier of 5): Total cost = 200 (5*5) = 225Kr. The country multiplier is up to the group to decide, but it is based on distance from Oslo - use Germany with a multiplier of 5 as inspiration.

3.2.4. FE-04: Anonymous/Guest Usage

A guest user must be able to send a shipment by filling out the needed delivery information and providing an e-mail only for receipt purposes (this must be clearly communicated to the user), they will be stored as a guest role in the database, only having their email entered. This can be presented as a "register later" option when a user that is not logged in is trying to create a shipment.

The receipt e-mail must also contain a link that the user can use to create an account which will then also 'claim' the shipment.

3.2.5. FE-05: Administrator main page

Once an administrator has logged in they are able to view all current shipments and their respective status'. From here the administrator has the ability to change a shipment's state. Administrators may also update the country multipliers to affect the pricing of shipments.

3.2.6. FE-06: User Account Management

There should be a page that can display the user's account information. The user should also be able to update their account information.

A guest user must not have access to an account page. They will need to create an account via the email sent when ordering a shipment.

3.2.7. FE-07: Package Status

This solution is for demonstration purposes, so all packages will move between states when triggered by the Administrator. There should be a debug page that allows this to be done easily.

- `CREATED`: This is the 'new' package state
- `RECIEVED`: This represents when the package has been physically collected for transport, but has not been routed for delivery yet.
- `INTRANSIT`: Represents a package that is being transported.
- `COMPLETED`: Represents a package that has been delivered.

- `CANCELLED`: Represents a package that has been cancelled

The shipment statuses must be recorded for history. This means that a user is able to view the complete history of the changes in a shipments status. A hint for this is to have many 'shipment status' records for one 'shipment'. When a user sees a list of their shipments, they see the latest status for that shipment - keep in mind complete shipments should be shown separately.

3.3. API Requirements

The API forms the core of this project and gives your front-end access to your database. All requests must contain a Basic Authentication Token[1] in the HTTP header when possible.

3.3.1. API-01: No Query Parameters

The API must not use query parameters to pass objects through, that should be done in the Body (application/json).

Sorting and filtering can be passed through as query strings.

3.3.2. API-02: Login

`POST /login`

Authenticates a user. Accepts appropriate parameters in the request body as `application/json`. This should also return their account type (customer, or administrator – See Appendix A). You will need this to determine what is or is not displayed on your front end. Make use of best practices and use a Basic Authentication token in the HTTP Header.

A failed authentication attempt should prompt a `401 Unauthorized` response. This endpoint should also be subject to a rate limiting policy where if authentication is attempted too many times (unsuccessfully) then requests from the corresponding address should be temporarily ignored. Candidates should decide on an appropriate threshold for rate limiting.

[1] `https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Authorization`

3.3.3. API-03: Shipments

`GET /shipments`

Retrieve a list of shipments relevant to the authenticated user.

A user will see only their shipments.

An administrator will see all current (non-cancelled/non-completed) shipments that will be displayed on the administrator page.

This can also optionally be filterable by status type or using a date range (from - to). This could be done using query strings to provide these filters.

`GET /shipments/complete` Retrieve a list of **completed** shipments relevant to the authenticated user (as with previous).

`GET /shipments/cancelled` Retrieve a list of **completed** shipments relevant to the authenticated user (as with previous).

`POST /shipments`

Used to create a new shipment, the client data must be retrieved based on the authorization transmitted in the request header.

Administrators can also create shipments.

`GET /shipments/:shipment_id`

Retrieve the details of a single shipment, remember to consider if the current user has access the requested shipment. (Users can only view their own shipment)

`GET /shipments/customer/:customer_id`

Retrieve the details of all the shipments a given customer has made.

`PUT /shipments/:shipment_id`

This endpoint is used to update a shipment, but any non-Administrator users may **only** cancel a shipment. Meaning, only admins can change shipment statuses aside from simply canceling.

An administrator can make any changes they wish to a shipment. The administrator will use this to mark a shipment as completed.[2].

[2] An Administrator may also revert a shipment from complete back to *in progress*

`DELETE /shipments/:shipment_id`

This endpoint is used to delete a shipment only in extreme situations, and **only** accessible by an Administrator.

(This will also delete completed/cancelled shipments.)

3.3.4. API-04: Account

`GET /account/:account_id`

Retrieve the account information of a specific user. Their account information should be displayed on the account page.

`PUT /account/:account_id`

This request should be used to update a specific account with new information as needed.

`POST /account` This endpoint is used to create a new account when needed.

`DELETE /account` This endpoint is used to delete an account only in extreme situations, and **only** accessible by an Administrator.

3.3.5. API-05: Settings

`GET /settings/countries`

Retrieve the country multiplier information stored in the database for viewing.

`POST /settings/countries:country_id`

This request should add a new country with all relevant information.

`PUT /settings/countries:country_id`

This request should be used to update a specific country with new information as needed.

3.4. Testing Requirements

3.4.1. DB-01: Unit tests

Unit tests should be created and passed for various aspects of the system. Some examples include:

- Creating a guest account.
- Registering a new user.
- Calculating correct costs for various shipments.
- Calculating correct total cost for all ordered/completed shipments.
- etc.

3.5. Database Requirements

3.5.1. DB-01: Relational Database

Candidates must select a relational database to deploy to store all the information of the properties and user accounts of the application. Candidates are expected to document the full structure of their database solution, such as in an ERD.

3.6. Security Requirements

3.6.1. SEC-01: User Authentication

Users should be authenticated appropriately before being allowed to interact with the system. All API endpoints (unless otherwise marked) should require an appropriate bearer token to be supplied in the `Authorization` header of the request. 2FA should be enforced[3].

It is recommended to use an external authentication provider (such as Microsoft, Google, or Gitlab) and the *OpenID Connect Auth Code Flow* [4]. If desired, a self-hosted

[3] `https://authy.com/what-is-2fa/`
[4] `https://openid.net/specs/openid-connect-core-1_0.html`

identity provider can be quickly deployed using Docker[5] and Keycloak[6].

Failed authentication attempts should prompt a `401 Unauthorized` response. Authentication related endpoints should also be subject to a rate limiting policy where, if authentication is attempted too many times (unsuccessfully), then requests from the corresponding address should be temporarily ignored. Candidates should decide on an appropriate threshold for rate limiting.

3.6.2. SEC-02: Input Sanitation

All endpoints that accept input from users must ensure that the provided data is appropriately sanitized or escaped so that it cannot be used in an XSS or injection attack.

3.6.3. SEC-03: Credential Storage

Care should be taken to ensure that administrative credentials (i.e. database credentials) are not hard coded into the application and are instead provided to the application using environment variables.

3.6.4. SEC-04: HTTPS

The final, public facing deployment of the application should enforce communication only over HTTPS.

[5] `https://hub.docker.com/r/jboss/keycloak/`
[6] `https://www.keycloak.org/`

3.7. Performance Requirements

3.7.1. PRF-01: Loading Time

The system is expected to be responsive at all times and not 'hang' or take excessive time to perform actions.

3.7.2. PRF-02: Error Messages

In the event of an error case, the user should be shown a meaningful error message describing what they have done wrong without being confusing. While error messages should be comprehensive, care should be taken not to expose sensitive information that could compromise the security of the application.

3.8. Documentation Requirements

3.8.1. DOC-01: User Manual

Candidates should submit a user manual containing instructions for how to perform each action within the system with accompanying screenshots.

3.8.2. DOC-02: API Documentation

Candidates should submit a detailed listing of each API endpoint they create with the following information:

1. Endpoint method.
2. Endpoint path.
3. Required and accepted headers.
4. Accepted parameters
5. Expected changes to the data.
6. Possible responses and their meanings.
7. Possible error cases with explanations.

3.8.3. DOC-03: README.md

Candidates should provide a `README` file with the following information:

1. A name for the project
2. A list of team members and project participants
3. Detailed installation instructions to run the service from scratch.

The previously mentioned API documentation ***MAY*** also be included in the `README` as opposed to being a separate document.

The user manual ***MAY NOT*** be included as part of the `README` and must be a separate document.

3.9. Other / Miscellaneous Requirements

3.9.1. MISC-01: Deliverables

The candidates are expected to deliver the following artifacts:

1. A `Git` repository containing all code and inline documentation.
2. Any and all project documentation, including plans, database schema and instructional material (such as the user manual).
3. A live, functional deployment of the completed product.
4. A presentation of their product.

3.9.2. MISC-02: Presentation

Candidates must produce a 30 minute presentation which introduces the team, and discuss their product in two parts:

Development (10 min). Candidates must discuss the choices made during development, the difficulties they faced and how they overcame them.

Feature Demonstration (10 min). Candidates must produce a brief overview of the functionality of their final product.

Questions (10 min). Sufficient time should remain for the audience to ask the candidates questions. Candidates should be prepared to defend decisions made throught the development process to the satisfaction of those present.

Candidates should create a 'walk-through' of the core functions of the system. This should include highlighting how the system behaves differently for different user types, and how each user type can achieve their individual goals within the system.

Appendices

A. Table Definitions

A.1. Account Type

Enumerator. One of the following:

- `GUEST`
- `REGISTERED_USER`
- `ADMINISTRATOR`

A.2. Shipment Status

Enumerator. One of the following:

- `CREATED`
- `RECIEVED`
- `INTRANSIT`
- `COMPLETED`
- `CANCELLED`