SOFTWARE REQUIREMENTS SPECIFICATION

for

Experis Academy Case: Vipps

Version 1.0
Winter 2020

By Craig Marais & Greg Linklater

Noroff Accelerate AS

March 2020

Contents

1 Revision History

Name	Date	Reason For Changes	Version
Jimmy's Initial Version	15/02/2019	Created Document	0.1
Craig's Revision	26/03/2019	Specifications Revision	0.2
Revision based on Jimm's comments	04/04/2019	Corrections	0.3
Autumn 2019	31.10.2019	Revision for Växjö cohort	0.4
Autumn 2019	01.03.2020	Date Corrections	1.0

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 in Oslo, Norway* contingent of the *.Net Full-stack* short course.

2.2 Document Conventions

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

- The *course* refers to the *.Net Full-stack* short course as currently offered at *Experis Academy in Oslo, Norway*; specifically the Winter 2020 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 four weeks.

2.5 Delivery

A deployed version of the software must completed by the morning of the 3^{rd} of April. Nearing the completion of the software, candidates will be required to present their solutions twice. A mock presentation will take place on the 31^{th} of March, and the final presentation will take place on the 2^{nd} of April.

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 a common situation that causes many headaches and bugs in the world of software engineering, commonly related to e-commerce transactions. Learning how to build more robust distributed systems will empower the students in the *.Net Full-stack* short course to make the world a more stable place.

The purpose here is to understand how to communicate between disparate services in an unreliable environment like the internet. When you send a message over the network, many things can go wrong along the way:

- Your message may never reach its recipient.
- Your message may take a long time to be handled.
- You may never get a response, even when your request was handled properly.
- And many more other conceivable outcomes.

This exercise should bring clarity on ways to send/receive messages in a way that only allows the intended action to happen, at most, once.

3.1 Product Perspective

For this case, the candidates are expected to design and develop a distributed software solution for accepting orders for a digital product, using a 3^{rd} party payment processor. The main components:

- A web front-end
- An API that receives data from the front-end and communicates with the database
- A Database
- A 3^{rd} party payment processor

3.2 Front-end Requirements

The front end should be a simplified (*symbolic*) store.

- **FE-01**: A landing page, with login options
  The front-end component should be very basic. All pages should be served `.html` files.
  A user will visit the site and have two option; 'login' or 'purchase'.
  Purchase: Lets a user proceed to the payment gateway (Anonymously).
  Login: An anonymous user can authenticate themselves (This need not be a fancy system).

- **FE-02**: A purchase page with generated items
  The system should generate zero to four items for purchase into a basket of goods. For zero items, the system should have appropriate warnings.
  (The items themselves can be random too.)

- **FE-03**: Desktop & mobile compatible
  Using modern design concepts/frameworks, the system should be function on both desktop computers and mobile devices.

- **FE-04**: Store/Retrieve user information
  Anonymous users must enter all payment information (Name, Address, etc.).
  Logged-in users need only approve the payment (using stored information).

- **FE-05**: User & purchase history
  Logged in users must also have an option to see information about past purchases.

- **FE-06**: Stripe integration
  The payment system must use an external API (such as Stripe) to validate the payment.

3.3 API Requirements

The API forms the core of this project.
The API must safely deal with all the possible errors that can occur during the processing of an order/payment.

- **API-01**:No query parameters:
  The API must not use query parameters at all.
  End points should use either `Headers` or `Body` (application/json)

- **API-02**: `GET /order`
  This gets all orders for the current user.

- **API-03**: `GET /order/<order_id>`
  Retrieves an order by it's id.
  Must only return if the order matches the current user.
  Invalid users must receive an error.

- **API-04**: `POST /order`
  Creates an order for the current user.
  The API must identify duplicate requests appropriately.

- **API-05**: Idempotent transactions
  If a duplicate order is received the system must handle it idempotently.
  The same transaction information is returned even if it is a duplicate, although it should **NOT** be doubly processed.
  Thus the client need not be aware that a duplication 'could' have occurred.

- **API-06**: Order processing
  New orders must be considered "*in progress*" until payment is confirmed and they are then marked as "*complete*".
  The front-end must reflect this status.

- **API-07**: 'Reasonable' number of simultaneous clients
  While the system is not expected to handle excessive numbers of users, it should react timeously under normal usage.

3.4 Database Requirements

- **DB-01**: An appropriate database design must be created and documented
  Certain parts of this solution will require database tables to store information.
  This is left to the candidates to design.

4 Other Considerations & Requirements

4.1 Security Requirements

4.1.1 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[1].

It is recommended to use an external authentication provider (such as Microsoft, Google, or Gitlab) and the *OpenID Connect Auth Code Flow* [2]. If desired, a self-hosted identity provider can be quickly deployed using Docker[3] and Keycloak[4].

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.

4.1.2 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.

4.1.3 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.

4.1.4 SEC-04: HTTPS

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

[1] `https://authy.com/what-is-2fa/`
[2] `https://openid.net/specs/openid-connect-core-1_0.html`
[3] `https://hub.docker.com/r/jboss/keycloak/`
[4] `https://www.keycloak.org/`

4.2 Performance Requirements

4.2.1 PRF-01: Loading Time

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

4.2.2 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.

4.3 Documentation Requirements

4.3.1 DOC-01: User Manual

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

4.3.2 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.

4.3.3 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.

4.4 Other / Miscellaneous Requirements

4.4.1 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.

4.4.2 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.