Report

Johan Delissen [*]
Emil Lilja [†]
Rasmus Bergström [‡]

December 14, 2018

Executive Summary

Databases are a fundamental part of the way we perceive the web. In order to practice using one, an E-Commerce website was built. It was decided to sell berries and related goods.

The technology stack was based on Django on top of a PostgreSQL database. Due to the problem constraint, the database schema was developed independently, and then added to Django using its `inspectdb` command.

The service allowed for creating an account, adding items to a basket and checking out that basket. It featured sorting and search, as well as product review. It also provided tools for administrators, such as adding, removing and editing products and orders.

Though this exercise is a good way of learning to work with a database, it would be highly recommended to use the Django Object-Relational Mapping tool in a real-world scenario.

Github repo: `https://github.com/JRasmusBm/ECommerceProject`
Demo site: `http://lab.giffeln.se`

[*]johdel-5@student.ltu.se
[†]emilil-5@student.ltu.se
[‡]rasber-5@student.ltu.se

1 Introduction

The students were tasked with building a web application that was to be used for e-commerce. The focus was to be placed on the database and server layers, while constructing a usable website through which the user could purchase items.

1.1 Requirements

The system should be able to process orders. There should be the concept of a shopping cart to which a customer can add products, and then check out with payment. It was strongly recommended that the database be a relational one. The use of a ORM for creating schemas was disallowed, because the goal of the assignment was to learn to administer the database.

1.2 Choice of Merchandise

Multiple different suggestions for merchandise were discussed, including computers and cars. Eventually it was decided, that because the merchandise itself was *not* the main focus of the project, something less obvious might as well be chosen. It was decided to sell berries, and in order to make the variety of items sold a bit more diverse (for a more interesting project), it was also suggested that related supplies, such as filters and jars for jam production, would also be sold.

1.3 Technologies

The choice of technologies was very open, which warranted some research into different technologies. The choices were then made in favor of technologies that would offer the most learning.

The choice for most of the stack fell on Django [2], because,

1. It is a widely used, tried and tested technology
2. Knowing Django could be an advantage in a future career
3. Other colleagues told us it would be difficult, due to the constraint about ORMs (See Section 1.1).

It was decided to make the application as production ready as possible, hosting it on a remote server inside docker [3] containers. The Django server was run using gunicorn [4], which was then mirrored using nginx [6] for added robustness.

The choice for the database fell upon PostgreSQL [7] because it is a popular alternative to MySQL [5] and has been gaining popularity for many years. It is therefore a good technology to be acquainted with.

2 Design

2.1 Role Descriptions and Features

A common way to describe desirable features for a software project is to formulate them in the form of user stories. Such a user story usually follows the pattern shown in Figure 2.1.

```
As a <Role>
I want <Feature>
So that I can have <Benefit>
```

Figure 2.1: The standard pattern for a user story.

For the E-commerce website there are three kinds of users. Customers, visitors and administrators. Below, a summary of the roles can be found. For complete user stories, please refer to Appendix B. Due to lack of time the user stories are not featured in the standard pattern manner.

2.1.1 Role: Visitor

A Visitor is any user that is not logged in. Such a visitor is able to list products, view product information find their current availability. They are also offered the option of registering an account on the site, thus becoming a Customer.

2.1.2 Role: Customer

A Customer is a registered user **without** admin privileges. In order to access Customer privileges it is required that the Customer is logged in. A Customer can do everything that a Visitor can do, as well as adding items to a shopping cart and making orders.

2.1.3 Role: Admin

A Admin is a registered user **with** admin privileges. In order to access Admin privileges it is required that the Admin is logged in. An Admin can do everything that a Customer can do. An Admin can also add, update and remove Products and Product Types, as well as handling orders.

2.2 Database Schema

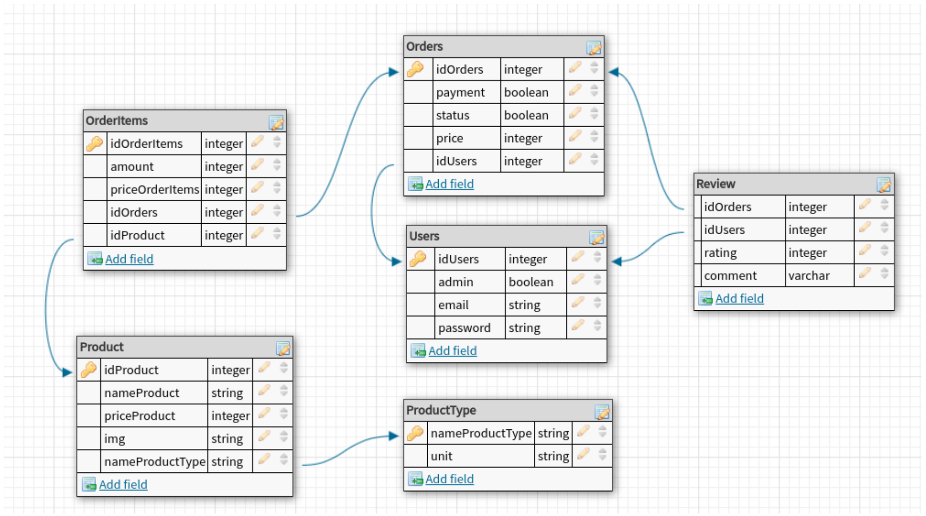


Figure 2.2: The database schema.

Users: Contains the Users, both admins and non-admins. The email is used to identify the User at login, so no username is needed.

Orders: Used to represent the shopping basket before an Order was made (`payment == False`), an Order that has been paid but not yet processed and shipped (`payment == True and status == False`), as well as finished Orders (`status == True`). Each Order has one User.

OrderItems: The mapping between Orders and Products (Products were previously known as Items, that is why the name is still OrderItems, may be subject to change in a later release). Keeps track of how many of each product is included in the Order, as well as the price of the Product at the time it was paid.

ProductType: The different possible types of a product. With each ProductType there is an associated unit in which a given amount of that ProductType is measured. These can be used when searching for products.

Product: The merchandise, including name, availability, price, image and ProductType. Price and availability are used for sorting.

Review: A review by a User on a Product, containing a rating and an optional comment. Only one review per product is allowed, if the same user tries to add another review, it overwrites the previous one. The rating is used for sorting.

2.3 Test-cases

Support for automatic tests was added in the following steps. A test container was added to docker. When that container runs, it performs all the tests using pytest [8]. Code coverage analysis is then reported using coverage.py [1].

Support for Selenium [9] was also added, in order to support end-to-end testing. Selenium relies on the browsers Firefox/Google Chrome, and so far attempts to make those run in docker have been unsuccessful. There are hopes, however, that such support could be implemented without much hassle.

Thus, though no actual test cases have been written, the groundwork was laid for being able to perform automated tests.

In the absence of automatic tests, the software was tested manually and often. This was done by comparing the information stored in the database with the information displayed on the screen, and checking if it made sense. Every once in a while, the list of finished tasks was gone through manually to check that there was no regression on previously solved issues.

3 Discussion

3.1 Considerations

Here are a couple of explanations as to why we've implemented some of the features the way we have.

3.1.1 Handling Orders

When a user pays for an order, he can no longer remove the order. The admin on the other hand can remove an order after it's been paid (assuming the company sends money back to the customer). This can not be done after the order has been handled. He can also set the handled status of orders, both to handled and to not handled in case there is a problem or misunderstanding.

3.1.2 Availability

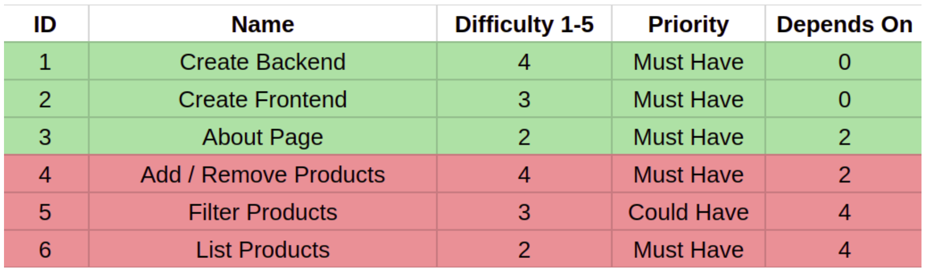
Products can not be added to orders in higher quantity than their availability. Each time the user interacts with the order basket, the price and the availability of the products are updated. In the case that the availability of a product is now lower than the amount of that same product in a users basket, the whole amount of that product is removed from the basket, to prevent confusion.

3.1.3 Filtering

Products can be filtered according to availability, rating, and price. This can be done `highest-lowest` and `lowest-highest`. When sorting for rating, products with no reviews are excluded, the same holds true for products without availability when sorting according to that metric.

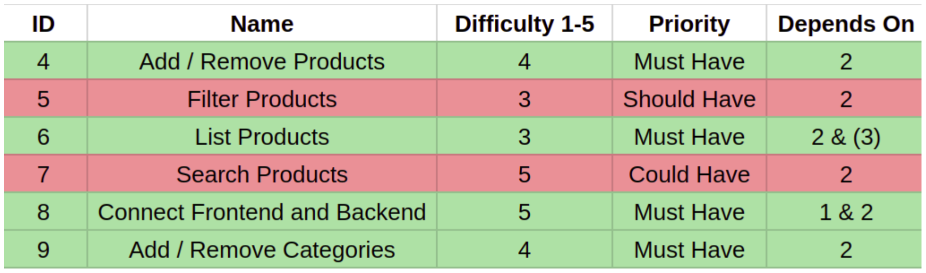
A Sprint Backlogs

In Figures A.1 - A.4, the issue backlog can be found. The list for each sprint consists of the issues that were planned for that particular sprint. A green background implies that the issue was actually addressed, a red background implies that it had to be postponed to a later sprint.



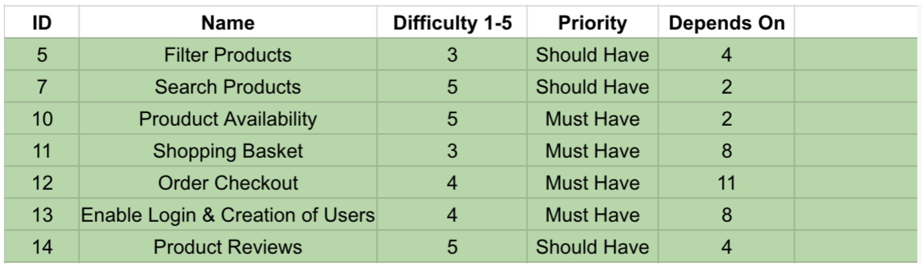
ID	Name	Difficulty 1-5	Priority	Depends On
1	Create Backend	4	Must Have	0
2	Create Frontend	3	Must Have	0
3	About Page	2	Must Have	2
4	Add / Remove Products	4	Must Have	2
5	Filter Products	3	Could Have	4
6	List Products	2	Must Have	4

Figure A.1: Issues, Sprint 1.



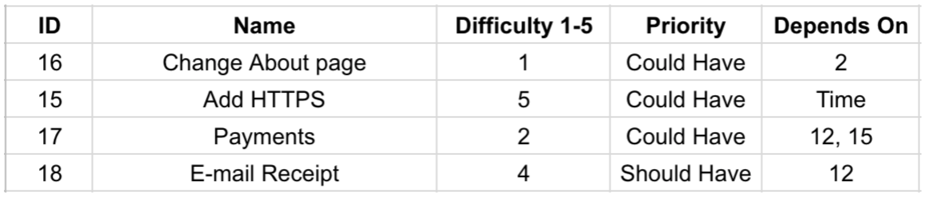
ID	Name	Difficulty 1-5	Priority	Depends On
4	Add / Remove Products	4	Must Have	2
5	Filter Products	3	Should Have	2
6	List Products	3	Must Have	2 & (3)
7	Search Products	5	Could Have	2
8	Connect Frontend and Backend	5	Must Have	1 & 2
9	Add / Remove Categories	4	Must Have	2

Figure A.2: Issues, Sprint 2.



ID	Name	Difficulty 1-5	Priority	Depends On	
5	Filter Products	3	Should Have	4	
7	Search Products	5	Should Have	2	
10	Prouduct Availability	5	Must Have	2	
11	Shopping Basket	3	Must Have	8	
12	Order Checkout	4	Must Have	11	
13	Enable Login & Creation of Users	4	Must Have	8	
14	Product Reviews	5	Should Have	4	

Figure A.3: Issues, Sprint 3.



ID	Name	Difficulty 1-5	Priority	Depends On
16	Change About page	1	Could Have	2
15	Add HTTPS	5	Could Have	Time
17	Payments	2	Could Have	12, 15
18	E-mail Receipt	4	Should Have	12

Figure A.4: Issues that have not yet been assigned to a sprint

B User Stories

B.1 Visitor Role and Functionality:

- The Visitor should be able to read about relevant information about the company to increase trustworthiness.
- The Visitor should be able to sort and view products given relevant categories e.g. Price, Category, Stock.
- The Visitor should be able to see quantity of products.
- The Visitor should be presented with deals, bundles of sorts for inspiration of purchase.
- The Visitor should be able to register for an acount on the site for Customer functionality.

B.2 Customer Role and Functionality:

- The Customer should be able to browse products and be able to add them to a shopping basket.
  - The added product should appear in the shopping basket.
  - The price of the product should be added to the price of the items in the shopping basket.
- The Customer should be able to checkout the items in the shopping basket.
  - The products in the shopping basket should be handled as an order
  - After checking out the basket should become empty.
  - The stock of each item ordered should be reduced according to its respective quantity.
  - After the customer checks out a receipt should be sent to the user via email.
- The Customer should be able to preform payments.
  - The money should be transferred to the company
  - Payment information should remain secure, this using Https.
- The Customer should be able to write reviews and leave ratings of products.
- The Customer should be able to recieve a receipt via E-mail.

B.3 Admin Role and Functionality:

- The admin should be able to do everything that the user can do.
- The admin should be able to add products.
- The admin should be able to remove products.
- The admin should be able to add categories.
- The admin should be able to remove categories.
- The admin should be able to alter the availability of the products.
- The admin should be able to change information about the company.
- The admin should be able to change handle status on order.

References

[1] Coverage.py. A tool for measuring code coverage of python programs. `https://coverage.readthedocs.io`.

[2] Django. The web framework for perfectionists with deadlines. `https://www.djangoproject.com/`.

[3] Docker. Protect your legacy, invest in your future. `https://www.docker.com/`.

[4] Gunicorn. Python wsgi http server for unix. `https://gunicorn.org/`.

[5] MySQL. The world's most popular open source database. `https://www.mysql.com/`.

[6] NGINX. High performance load balancer, web server and reverse proxy. `https://www.nginx.com/`.

[7] PostgresSQL. The world's most advanced open source relational database. `https://www.postgresql.org/`.

[8] Pytest. Helps you write better programs. `https://docs.pytest.org/en/latest/`.

[9] Selenium. Browser automation. `https://www.seleniumhq.org/`.