Report - DAT076

André Höjmark - Github Project

Contents

1 Introduction

The website is an ecommerce site that is able to sell products to customers and then add new ones easily.

1.1 Front Page

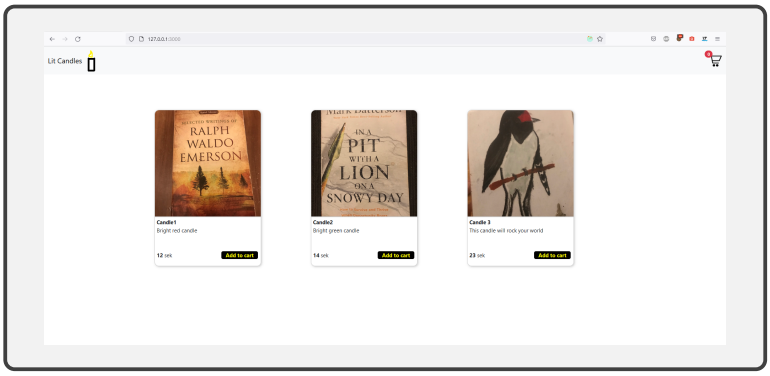

Figure 1. This is the view new users will first see when visiting the website

1.2 Add To Cart

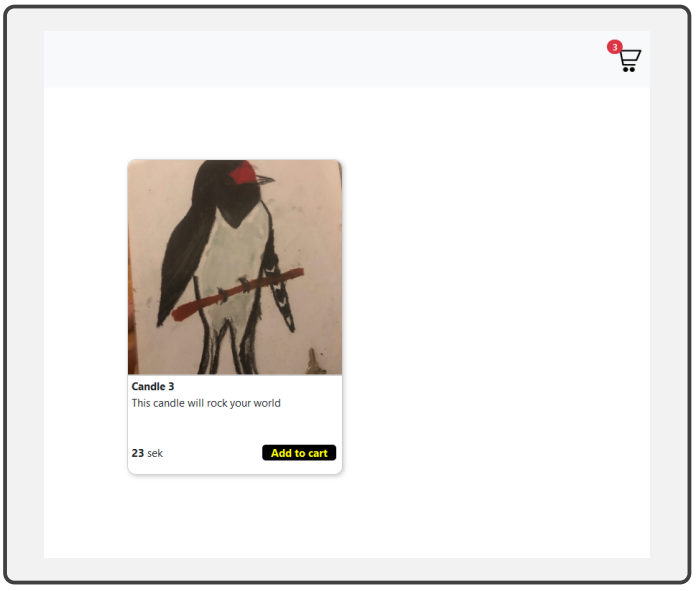

Figure 2. Upon pressing Add To Cart, an incremental number will be added to the cart icon at the top right

1.3 Cart Page

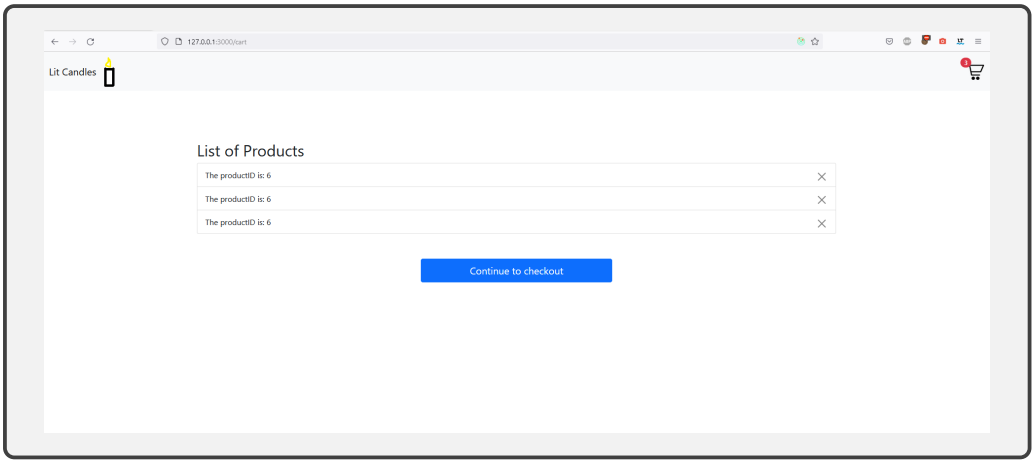

Figure 3. This cart page will show all the products added, and it's possible to remove items from the cart by pressing the X symbol

1.4 Payment page

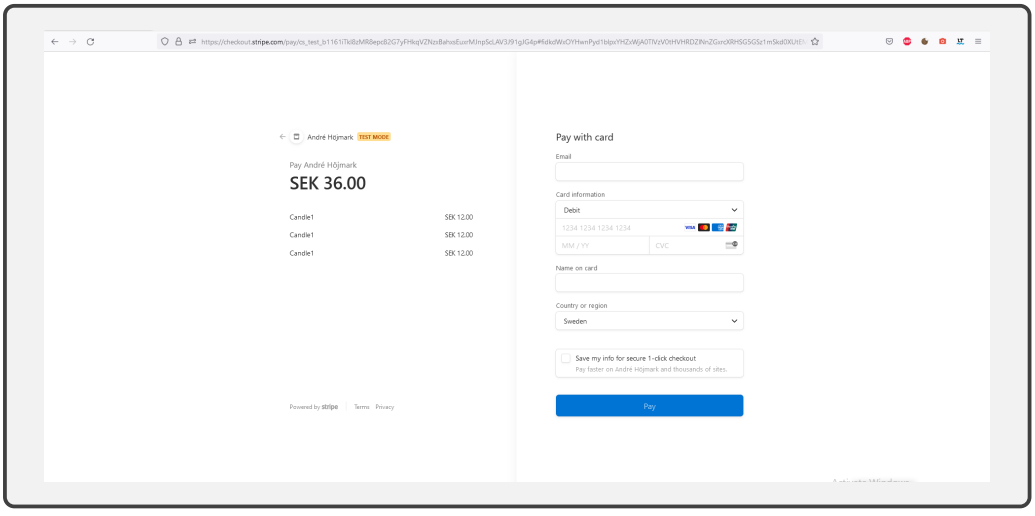

Figure 4. This is a stripe page that will list all the items from the cart page and allow the user to pay

1.5 Admin Panel Overview

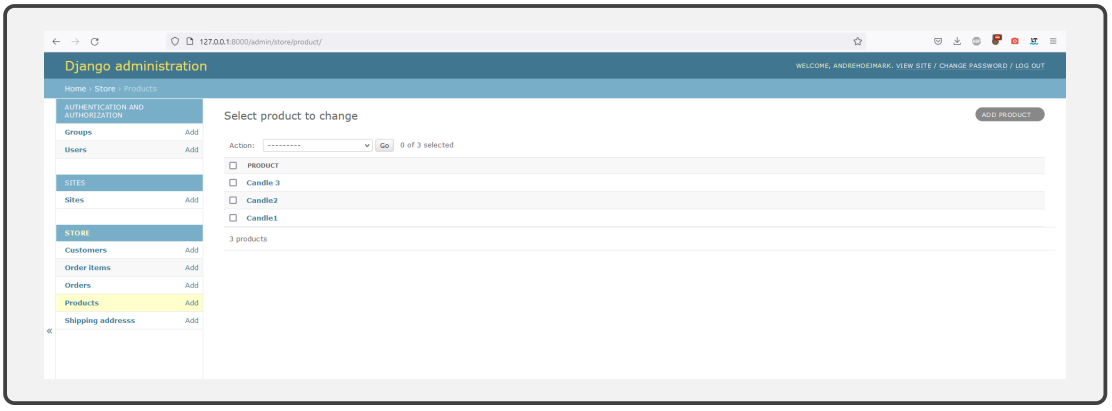

Figure 5. This is a snippet of the admin panel where a person with an admin account can view/add/remove products

1.6 Admin Panel Add Products

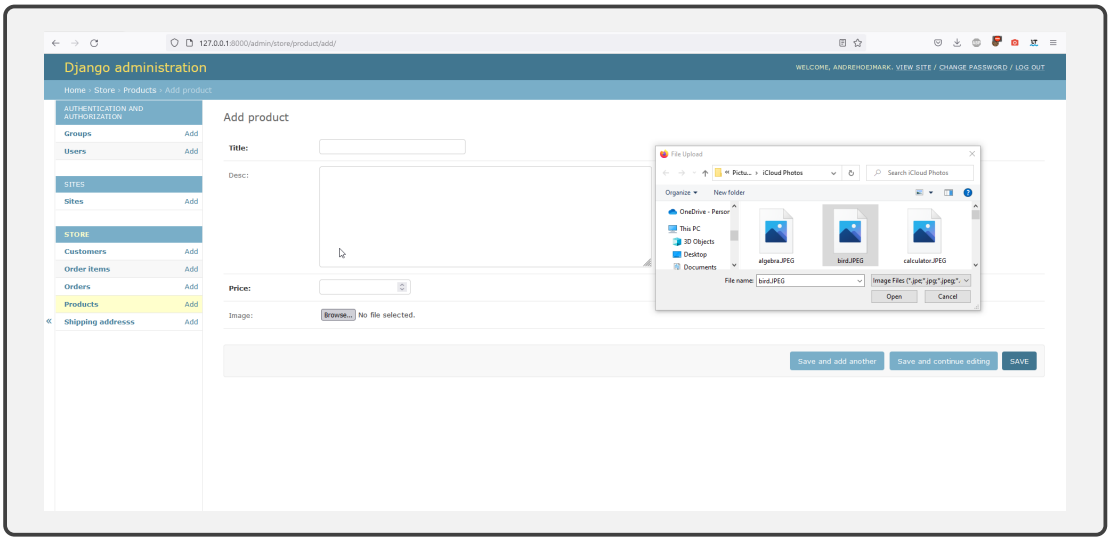

Figure 6. This is a snippet of the admin panel where a person with an admin account can add new products

2 A list of use cases

- View products on frontpage
- Add/Remove product to cart
- Add/Remove products to website with custom title, price, description and image
- Added/Removed products are automatically displayed on the website and have full functionality
- Payment integration with Stripe to pay for items in Cart

3 User manual

This section contains instructions on how to use the app from the user's point of view.

1. Adding Items To Purchase
   (a) Find item you want by looking at the images on the front page
   (b) Press Add To Cart
   (c) Repeat until you added all items you want
2. Removing Items from being purchased
   (a) Click on the cart icon upper right
   (b) Click on the X symbol from the product list to remove the item you no longer want
3. How to pay
   (a) Click on the cart icon upper right
   (b) Verify the items in the cart are the items you want to purchase
   (c) Press Continue to Checkout
   (d) Make your payment and a receipt will be sent to your email as well as your selected item within 7 days.

4 Design

This part will describe the technical construction of the application.

4.1 Front End

The design for the front end is made with a particular architectural pattern, which can be seen since all the components are put into a corresponding component folder with a descriptive name. There also are no code duplicates because everything is reused with components when something is redone, such as having multiple reused <Item/> components. I'm also using styled components to more easily have the components prestyled to just plug and play.

The project is made with bootstrap grid system to make it automatically adjust the grid layout for the items depending on the size of browser.

The frontend has some minor testing to see if a website is running. I've experimented a lot with mocking Redux, which is the library I used to control state variables across components and pages. In the end I came a long way but still had trouble to finish it off, and since testing in the frontend wasn't a requirement I skipped it.

There are some minor comments in the code, but mostly I try to name the functions and components in a descriptive manner so that it's not needed to make comments.

4.2 Back End

The design of the back end is made with a particular architectural pattern because it's divided into different services(applications in Django terms). The store application have views and models for the different products that could be sold, and the payment application contain the payment logic for stripe integration and future payment systems.

The backend is built using the model and view paradigm, which is default for using Django Rest-Framework. It has a config file in backend/config/settings.py that contains the config setup of the different functionalities such as the relational SQLite database, Google Cloud Platform Cloud Storage to store the images, CORS, my created services/applications and many more.

There also have been both unit testing and integrations testing to verify the functionality of the app. The unit test tested that the API URLs worked and that it's possible to get the products.The integrations test checked if it's able to create a new product and then make a get request to view it afterwards, as well if it was able to redirect to the stripe checkout for a particular item.

The testing file was put into the particular service/application which it tested. The use cases it tested were 1. View products on front page 2. Add products to website with custom title, price, description 3. Stripe payment redirection to the payment site 4. Added products are automatically displayed and with some functionality

There are different roles depending on if a customer or administrator. In the Django project, it's possible to create a superuser account and then give it certain permissions to add/remove products from the administrator's page which is what I've done.

There are a few security aspects

1. The images are stored using Google Cloud Platform and to access them there is a need to have a private json key. This key is only accessible during my development and never uploaded to the git repo(git ignore). This makes it so not anyone can upload images. When deploying the application on let's say Google Cloud Platform, there is no need to provide the JSON key since it automatically access the cloud storage on their own servers.

2. There is a Debug==True/False feature in Django's config file so that if there is an unexpected error when something happens they don't get any sensitive code back and the source code gets encrypted in a certain way with a secure key.

3. The pricing supplied to the Stripe payment is always done on the backend so we only supply the product ids from the frontend and then the backend finds the price. We never send the price from the react(frontend) to the backend and then supply that to the stripe payment system because then people could tinker with the prices if the post request was sent from react.

4. It also uses CORS which is a security feature so that only trusted domains can access the site.

5. Right now the stripe API key is hard coded which isn't ideal to have stored so that others who have access to the Github can view it. In the future, it's meant to use Google Secrets that provide/load the secret key upon run time.

4.3 API

Request: Stripe Checkout

Type: Post Request

URL: http://127.0.0.1:8000/payments/create-checkout-session

SUCCESS

Return: redirect(checkout_session.url, status=HTTP_302_Redirect)

FAIL

Return: Response("errors": error message, status=HTTP_500_INTERNAL_SERVER_ERROR)

INVALID INPUT

Return: Response("errors": serializer error message, status=HTTP_400_BAD_REQUEST)

Request: GetProducts

Type: Get Request

URL: http://127.0.0.1:8000/store

SUCCESS

Return: redirect("items": data, status=HTTP_200_OK)

FAIL

Return: Response("errors": error message, status=HTTP_500_INTERNAL_SERVER_ERROR)

5 Responsibilities & Contributions

Team size of 1.

Group number	12
Group name	Group 12
My name	André Højmark

Use the scale 1-10 and make an estimation of each group members contribution to the project. You should also give each member a fair salary (by contribution) by distributing 100 000 kr between all group members.

Member name	Engagement	Time spent	Technical skill	**Salary**
André H	7	10	8	100 000 kr
	1-10	1-10	1-10	distribute 100 000 kr

6 Tools

6.1 React Libraries

"dependencies": {
"@reduxjs/toolkit": "^1.7.2",
"@testing-library/user-event": "^13.5.0",
"@types/jest": "^27.4.0",
"@types/node": "^16.11.24",
"@types/react": "^17.0.39",
"@types/react-dom": "^17.0.11",
"@types/react-router": "^5.1.18",
"@types/react-router-dom": "^5.3.3",
"axios": "^0.26.0",
"axios-cookiejar-support": "^2.0.3",
"bootstrap": "^5.1.3",
"jest": "^27.5.1",
"jest-mock-axios": "^4.5.0",
"jquery": "^3.6.0",
"popper.js": "^1.16.1",
"react": "^17.0.2",
"react-bootstrap": "^2.1.2",
"react-dom": "^17.0.2",
"react-redux": "^7.2.6",
"react-router": "^6.2.1",
"react-router-dom": "^6.2.1",
"react-scripts": "5.0.0",
"redux-mock-store": "^1.5.4",
"styled-components": "^5.3.3",
"tough-cookie": "^4.0.0",
"typescript": "^4.5.5",

6.2 Django Modules

cachetools==5.0.0

certifi==2021.10.8

charset-normalizer==2.0.12

Django==3.2.5

django-cors-headers==3.11.0

django-extensions==3.1.3

django-rest-framework==0.1.0

django-storages==1.12.3

djangorestframework==3.12.4

google-api-core==2.5.0

google-auth==2.6.0

google-cloud-core==2.2.2

google-cloud-storage==2.1.0

google-crc32c==1.3.0

google-resumable-media==2.2.1

googleapis-common-protos==1.54.0

idna==3.3

Pillow==9.0.1

protobuf==3.19.4

pyasn1==0.4.8

pyasn1-modules==0.2.8

pytz==2021.3

requests==2.27.1

rsa==4.8

six==1.16.0

sqlparse==0.4.2

stripe==2.65.0

typing-extensions==4.1.1

urllib3==1.26.8