TTC2080-3034, Harjoitustyödokumentointi

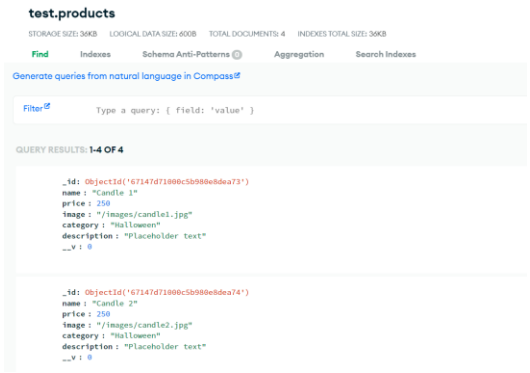
Tässä harjoitustyössä kehitin nettikauppasovelluksen tyttöystävälleni, joka tekee kynttilöitä ja myy niitä eteenpäin asiakkaille. Projektiin kuuluu backend- ja frontendosuus. Kehittämisen aloitin miettimällä ideaa, kuinka lähden toteuttamaan sitä ja mitä ominaisuuksia projektissa kuuluisi olla. Projekti on toteutettu noin 6 viikon aikana. Aloitin projektin lokakuun puolessa välissä, ja marraskuun 20. päivä saavuin siihen pisteeseen missä projekti on nyt. Tarkoituksenani kuitenkin on kehittää vielä tätä projektia eteenpäin ja tehdä kauppasivusta entistä parempi, esimerkiksi lisätä tietokanta käyttäjille jotka voivat rekisteröityä. Projektin kehittämisessä käytettyjä kirjastoja ja teknologioita ovat, React, MUI, node.js, express.js, MongoDB ja Exchangerate-API.

Backend-kansio sisältää express.js-sovelluksen lähdekoodin. Backend-kansiosta löytyy muun muassa products.js-koodi, jossa on tieto jokaisesta tuotteesta. Backend-kansiosta myös löytyy kuvat tuotteille. Minulla on seeder.js-scripti kehitettynä, jonka avulla voi päivittää tiedot tuotteille.

```
const products = [
  {
    name: 'Candle 1',
    price: 250.00,
    image: '/images/candle1.jpg',
    category: "Halloween",
    description: 'Placeholder text',
  },
  {
    name: 'Candle 2',
    price: 250.00,
    image: '/images/candle2.jpg',
    category: "Halloween",
    description: 'Placeholder text',
  },
  {
    name: 'Candle 3',
    price: 250.00,
    image: '/images/candle3.png',
    category: "Christmas",
    description: 'Placeholder text',
  },
  {
    name: 'Candle 4',
    price: 250.00,
    image: '/images/candle4.png',
    category: "Christmas",
    description: 'Placeholder text',
  },

];

module.exports = products;
```

```
const mongoose = require('mongoose');
const dotenv = require('dotenv');
const Product = require('./models/Product');
const products = require('./data/products');

dotenv.config();

// Connect to the database
mongoose.connect(process.env.MONGO_URI, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});

const importData = async () => {
  try {
    // Delete existing data
    await Product.deleteMany();

    // Insert new data
    await Product.insertMany(products);

    console.log('Data Imported!');
    process.exit();
  } catch (error) {
    console.error('Error with data import:', error);
    process.exit(1);
  }
};

importData();
```

Backend-kansiosta löytyy myös db.js ja .env tiedostot, joita on hyödynnetty siihen että MongoDB-yhteys onnistuu, db.js -tiedostossa on scripti joka määrittelee tuon MongoDB:hen luotavan yhteyden, .env tiedosto taas sisältää URL:n jota käytetään yhdistäessä MongoDB:hen. Alla olevassa kuvankaappauksessa myös näkyy, että MongoDB-yhteys toimii ja tuoteparametrit löytyvät MongoDB:stä.




```
const mongoose = require('mongoose');

const connectDB = async () => {
  try {
    const conn = await mongoose.connect(process.env.MONGO_URI);
    console.log(`MongoDB Connected: ${conn.connection.host}`);
  } catch (error) {
    console.error(`MongoDB connection failed: ${error.message}`);
    process.exit(1);
  }
};

module.exports = connectDB;
```

```
1   MONGO_URI=mongodb+srv://ab7920:W6JnmzU0gdD0u9n5@test.5tnch.mongodb.net/
```

Kansiosta löytyy myös OrderRoutes.js ja ProductRoutes.js -tiedostot, jotka määrittelevät Express-reitit tuotteille ja reitit tilauksille. Productroutes esimerkiksi hakee tuotteet MongoDB:stä ja sen jälkeen palauttaa ne vastauksena. Orderroutes taas esimerkiksi luo uuden tilauksen, mutta ensin tarkistaa onko orderItems-taulukko tyhjä.

```
const express = require('express');
const router = express.Router();
const Order = require('../models/Order');

// @route   POST /api/orders
// @desc    Create a new order
router.post('/', async (req, res) => {
  const { orderItems, totalPrice } = req.body;

  if (orderItems && orderItems.length === 0) {
    return res.status(400).json({ message: 'No order items' });
  }

  const order = new Order({
    orderItems,
    totalPrice,
  });

  const createdOrder = await order.save();
  res.status(201).json(createdOrder);
});

module.exports = router;
```

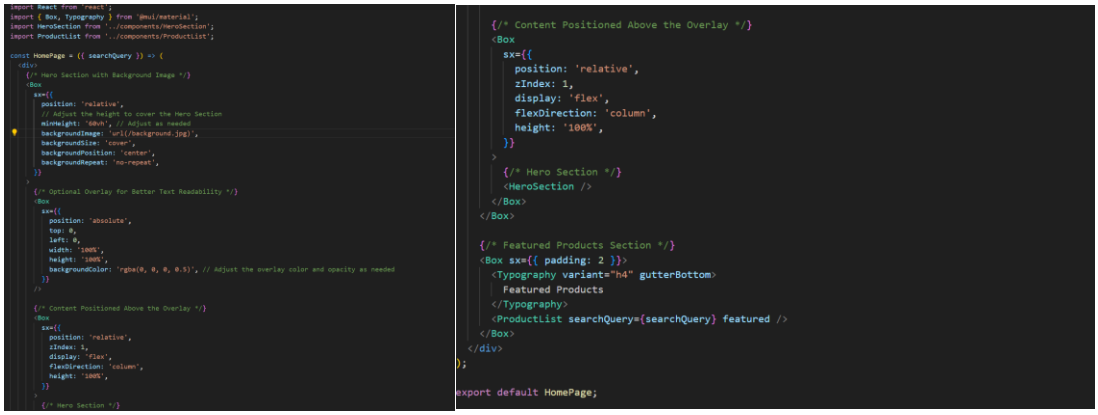
```
const express = require('express');
const router = express.Router();
const Product = require('../models/Product');

// @route   GET /api/products
// @desc    Get all products
router.get('/', async (req, res) => {
  try {
    const products = await Product.find({});
    res.json(products);
  } catch (error) {
    res.status(500).json({ message: 'Server Error' });
  }
});

// @route   GET /api/products/:id
// @desc    Get a product by ID
router.get('/:id', async (req, res) => {
  try {
    const product = await Product.findById(req.params.id);
    if (product) {
      res.json(product);
    } else {
      res.status(404).json({ message: 'Product not found' });
    }
  } catch (error) {
    res.status(500).json({ message: 'Server Error' });
  }
});

module.exports = router;
```

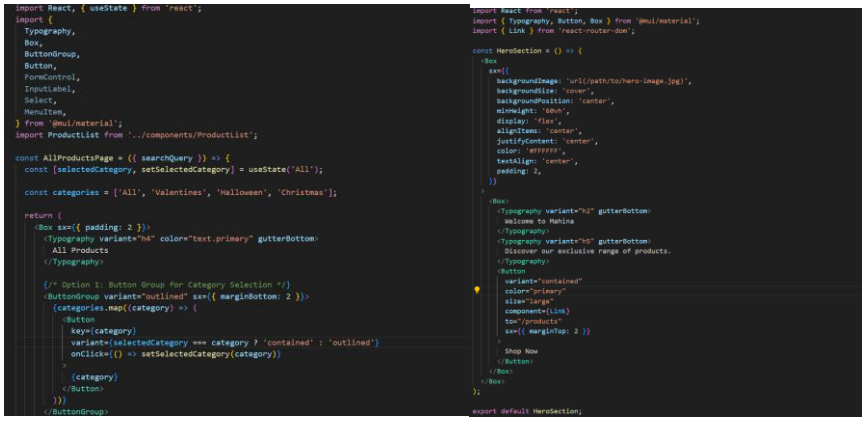
Näiden lisäksi backend sisältää myös product.js -koodin joka määrittelee tuotteen rakenteen hyödyntäen kenttiä name, price, category, image ja description. Order.js -tiedosto sisältää mallin tilauksille, tässä tiedostossa käytetään kenttiä orderItems, totalPrice, isPaid ja paidAt.

```
const mongoose = require('mongoose');

const productSchema = new mongoose.Schema({
  name: String,
  price: Number,
  image: String,
  category: String,
  description: String,
});

module.exports = mongoose.model('Product', productSchema);
```

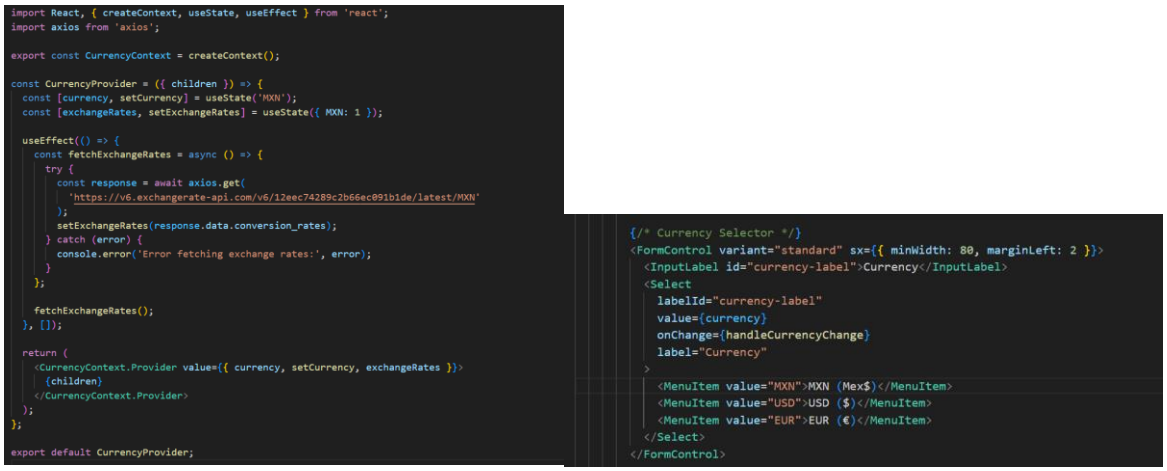
```
const mongoose = require('mongoose');

const orderSchema = new mongoose.Schema({
  orderItems: [
    {
      product: { type: mongoose.Schema.Types.ObjectId, ref: 'Product' },
      qty: Number,
    },
  ],
  totalPrice: Number,
  isPaid: { type: Boolean, default: false },
  paidAt: Date,
});

module.exports = mongoose.model('Order', orderSchema);
```

Projektin frontend-kansiosta löytyy jokaiselle sivulle koodi, joka nettisivulta löytyy. Näitä ovat muun muassa HomePage.jsx, AllProductsPage.jsx ja Cartpage.jsx. Näiden sivujen kehittämisessä on hyödynnetty MaterialUI-kirjastoa, jotta sivut näyttäisi paremmalta silmään. Kehitin myös HeroSection.jsx-koodin, jolla mahdollistin sen että sivun banneri näyttäisi paremmalta.





Frontend-kansiosta löytyy myös Currencycontext.js-tiedosto, joka mahdollistaa valuutanmuunnokset sivulla. Tässä on hyödynnetty Exchangerate-API-kirjastoa, joka mahdollistaa reaaliaikaiset valuuttamuunnokset. Valuuttana joka on ”defaulttina” toimii Meksikon peso ja muut valuutat lasketaan kurssimuunnoksina tästä, esimerkiksi pesoista euroihin.



Muita tiedostoja, mitä frontend-kansiosta löytyy, ovat myös Navbar.css ja Navbar.jsx, jotka määrittelevät navigaatiopalkin ulkonäön sekä toimivuuden. Navbar.jsx-tiedostosta löytyy Currency selector -osuus, johon on tällä hetkellä lisätty vaihtoehdoiksi MXN, USD ja EUR. Frontend-kansio sisältää myös Productlist.jsx -tiedoston, joka on linkitetty esimerkiksi AllProductsPage.jsx -tiedostoon. Productcard.jsx -tiedosto määrittää jokaiselle ”tuotekortille” ulkoasun ja hyödyntää myös CurrencyContext -scriptiä laskeakseen oikean hinnan tietyssä valuutassa tuotteelle, kuten nähdään alla olevista kuvankaappauksista.

```jsx
import React, { useContext } from 'react';
import {
  Card,
  CardMedia,
  CardContent,
  Typography,
  CardActions,
  Button,
} from '@mui/material';
import { Link } from 'react-router-dom';
import { CurrencyContext } from '../context/CurrencyContext'; // Import CurrencyContext

const ProductCard = ({ product }) => {
  const { currency, exchangeRates } = useContext(CurrencyContext);

  // Calculate price in selected currency
  const priceInSelectedCurrency = (
    product.price * exchangeRates[currency]
  ).toFixed(2);

  // Define currency symbols
  const currencySymbols = {
    USD: '$',
    EUR: '€',
    MXN: 'Mex$',
  };

  return (
    <Card>
      <Link to={`/product/${product._id}`}>
        <CardMedia
          component="img"
          height="300"
          image={`http://localhost:5000${product.image}`}
          alt={product.name}
        />
      </Link>
      <CardContent>
        <Typography variant="h6" gutterBottom>
          {product.name}
        </Typography>
        <Typography variant="body2" color="text.secondary">
```

```jsx
return (
  <Card>
    <Link to={`/product/${product._id}`}>
      <CardMedia
        component="img"
        height="300"
        image={`http://localhost:5000${product.image}`}
        alt={product.name}
      />
    </Link>
    <CardContent>
      <Typography variant="h6" gutterBottom>
        {product.name}
      </Typography>
      <Typography variant="body2" color="text.secondary">
        {product.description}
      </Typography>
      <Typography variant="h6" color="primary">
        {currencySymbols[currency]}
        {priceInSelectedCurrency}
      </Typography>
    </CardContent>
    <CardActions>
      <Button
        variant="contained"
        color="primary"
        component={Link}
        to={`/product/${product._id}`}
      >
        View Details
      </Button>
    </CardActions>
  </Card>
);
};

export default ProductCard;
```

Frontend-kansiossa on myös CartContext.jsx -scripti, joka mahdollistaa toimivan ostoskori-mahdollisuuden. Cartpage.jsx -tiedostossa on käytetty hyödyksi tätä scriptiä, jotta ostoskori-sivu toimisi mahdollisimman sulavasti. Cartpage.jsx -tiedostossa on myös määritelty funktiot esimerkiksi kokonaissumman laskemiselle ja jälleen CurrencyContext -scriptiä hyödynnetään, jotta saadaan oikea summa tiettyä valuuttaa käyttäen.

```jsx
import React, { useContext } from 'react';
import { Box, Typography, Button, Grid } from '@mui/material';
import { Link } from 'react-router-dom';
import { CartContext } from '../context/CartContext';
import { CurrencyContext } from '../context/CurrencyContext';

const CartPage = () => {
  const { cartItems, removeFromCart } = useContext(CartContext);
  const { currency, exchangeRates } = useContext(CurrencyContext);

  // Define currency symbols
  const currencySymbols = {
    USD: '$',
    EUR: '€',
    MXN: 'Mex$',
  };

  // Calculate total price
  const totalPrice = cartItems.reduce((total, item) => {
    console.log(item)
    return total + item.price * item.qty;
  }, 0);


  // Convert total price to selected currency
  const totalPriceInSelectedCurrency = (
    totalPrice * exchangeRates[currency]
  ).toFixed(2);

  return (
    <Box sx={{ padding: 2 }}>
      <Typography variant="h4" gutterBottom>
        Shopping Cart
      </Typography>

      {cartItems.length === 0 ? (
        <Typography variant="body1">Your cart is empty.</Typography>
      ) : (
        <Box>
          {cartItems.map((item) => {
            const priceInSelectedCurrency = (
              item.price * exchangeRates[currency]
```

```jsx
        return (
          <Grid container spacing={2} key={item._id}>
            {/* Product details */}
            <Grid item xs={12} sm={6}>
              <Typography variant="h6">{item.name}</Typography>
              <Typography variant="body2">
                Quantity: {item.qty}
              </Typography>
              <Typography variant="body2">
                Price: {currencySymbols[currency]}
                {priceInSelectedCurrency}
              </Typography>
            </Grid>
            {/* Remove button */}
            <Grid item xs={12} sm={6}>
              <Button
                variant="outlined"
                color="secondary"
                onClick={() => removeFromCart(item._id)}
              >
                Remove
              </Button>
            </Grid>
          </Grid>
        );
      })}

      {/* Total Price */}
      <Typography variant="h6" sx={{ marginTop: 2 }}>
        Total: {currencySymbols[currency]}
        {totalPriceInSelectedCurrency}
      </Typography>

      {/* Checkout Button */}
      <Button
        variant="contained"
        color="primary"
        component={Link}
        to="/checkout"
        sx={{ marginTop: 2 }}
```

```
import React, { createContext, useState, useEffect } from 'react';

export const CartContext = createContext();

const CartProvider = ({ children }) => {
  const cartFromStorage = JSON.parse(localStorage.getItem('cartItems')) || [];
  const [cartItems, setCartItems] = useState(cartFromStorage);

  useEffect(() => {
    localStorage.setItem('cartItems', JSON.stringify(cartItems));
  }, [cartItems]);

  const addToCart = (product, qty = 1) => {
    const existItem = cartItems.find((x) => x._id === product._id);
    if (existItem) {
      setCartItems(
        cartItems.map((x) =>
          x._id === product._id ? { ...x, qty: x.qty + qty } : x
        )
      );
    } else {
      setCartItems([...cartItems, { ...product, qty }]);
    }
  };

  const removeFromCart = (productId) => {
    setCartItems(cartItems.filter((x) => x._id !== productId));
  };

  return (
    <CartContext.Provider value={{ cartItems, addToCart, removeFromCart }}>
      {children}
    </CartContext.Provider>
  );
};

export default CartProvider;
```

Näiden tiedostojen lisäksi frontend-kansioon kuuluu myös olennaiset tiedostot jotka mahdollistavat sivun toimivuuden Reactia käyttäen, kuten app.js ja muut välttämättömät React-tiedostot.

Tiivistettynä harjoitustyön tekeminen mahdollisti itselleni syvemmän oppimisen liittyen Full Stack -ohjelmointiin. Vahvuuksinani harjoitustyössäni koen sen että se on suhteellisen laaja ja siinä on myös hyödynnetty erilaisia teknologioita kuten MongoDB -tietokantaa ja ExchangeRate-APIa. Heikkouksina taas on esimerkiksi maksuteknologina puuttuminen, mutta tämä on tarkoitus kehittää myöhemmin. Työ oli haastava ja olenkin itse arvioinut sen olevan kurssiarvosanan 5 arvoinen. Työhön on panostettu ja siihen on myös käytetty aikaa. Sivusto ei vielä ole valmis, mutta omasta mielestäni tällä hetkellä ihan toimiva ratkaisu kauppapaikkasivulle. Seuraavaksi ajattelin kehittää kirjautumismahdollisuuden sivulle ja kun sivustoa aletaan oikeasti hyödyntämään, pitää myös maksumahdollisuudet implementoida.