Vitacore

Outline

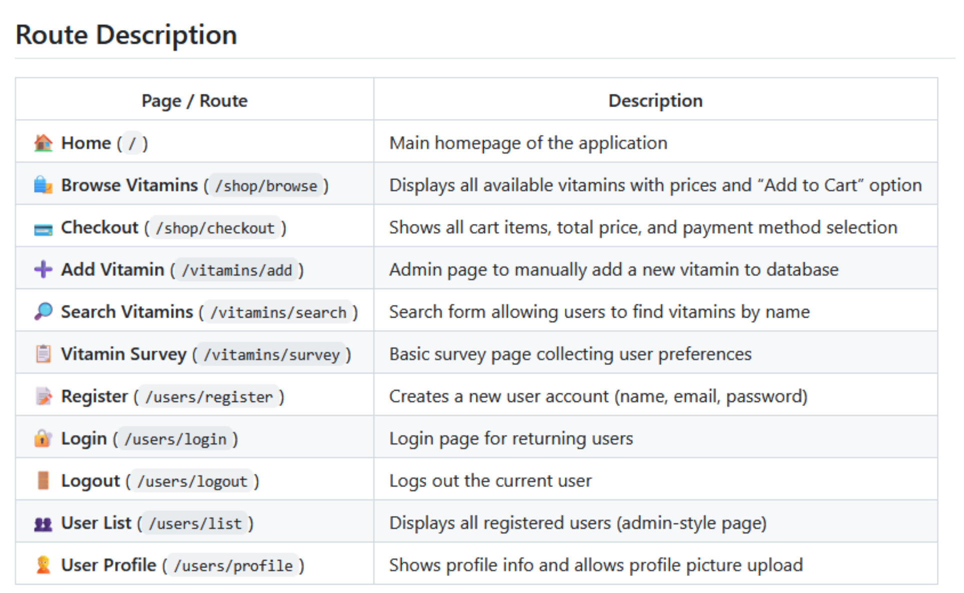
During this assignment, I decided to build a health website tailored around selling and recommending vitamins to its users. I selected this theme because it provided me with an opportunity build a more shop style application whilst expanding on my current skillset.

My project allows for users to browse, purchase and learn about different vitamins and their functionality. It provides for a dynamic product catalogue where the price, description and name are displayed in a session-based shopping cart allowing for their progress to be saved whilst logged into the browser. Alongside with this, my website aimed to provide a mock checkout system to help simulate the feeling of online shopping and browsing,

This application also allowed for me to create a full user account functionality where users can register with sanitised inputs, manage their own profiles including uploading their own chosen personalised profile picture. Additionally, session management also ensures that during the time the users are active on the website, only logged in users can access to full functionality of the website.

Overall, during this process I had the opportunity to demonstrate my skills during this module through developing my application such as through using express.js, ejs, sessions, input validation, file uploads, and database integration into a user-friendly shop.

Architecture



Route Description

Page / Route	Description
🏠 **Home** (`/`)	Main homepage of the application
🛍️ **Browse Vitamins** (`/shop/browse`)	Displays all available vitamins with prices and "Add to Cart" option
💳 **Checkout** (`/shop/checkout`)	Shows all cart items, total price, and payment method selection
➕ **Add Vitamin** (`/vitamins/add`)	Admin page to manually add a new vitamin to database
🔎 **Search Vitamins** (`/vitamins/search`)	Search form allowing users to find vitamins by name
📋 **Vitamin Survey** (`/vitamins/survey`)	Basic survey page collecting user preferences
📝 **Register** (`/users/register`)	Creates a new user account (name, email, password)
🔐 **Login** (`/users/login`)	Login page for returning users
🚪 **Logout** (`/users/logout`)	Logs out the current user
👥 **User List** (`/users/list`)	Displays all registered users (admin-style page)
👤 **User Profile** (`/users/profile`)	Shows profile info and allows profile picture upload

List of web pages from my readme file.

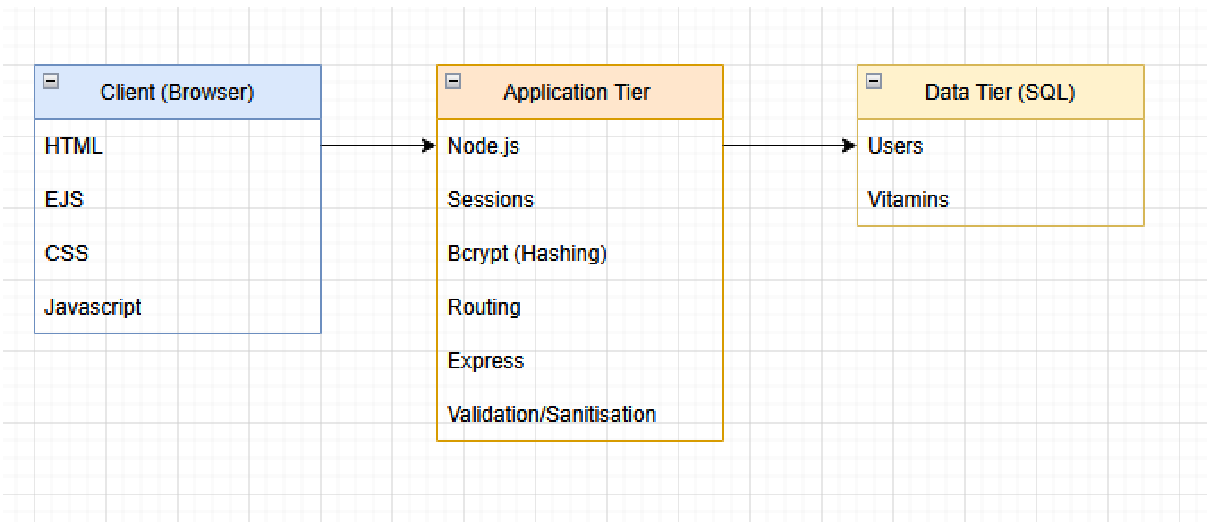
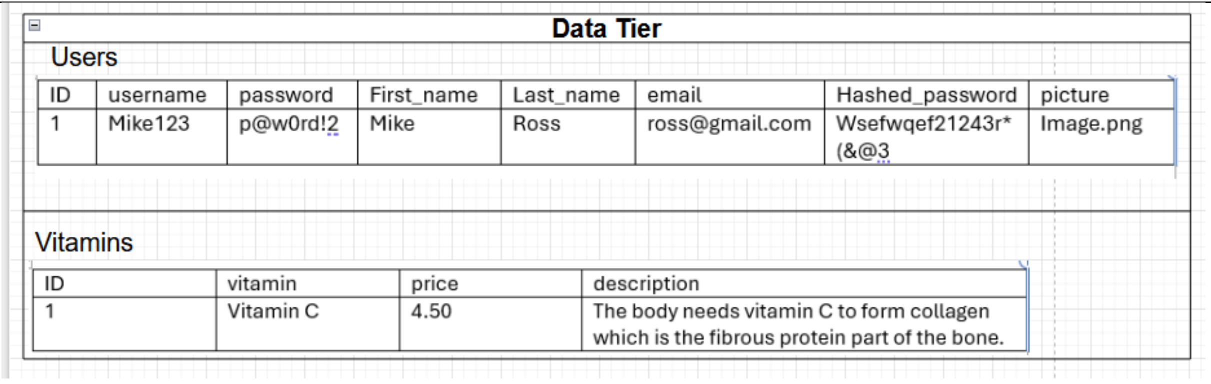


Diagram of my architecture

As you can see in my diagram, there is a client, application, and data tier. The application tie is primarily built with node.js and express.js to help with handling sessions, and routing. Alongside with this, we also utilise the bcrypt library for converting passwords into hashes. The client side of my application also manages the shopping cart using sessionStorage.

The data tier is where mySQL is being utilised to store users and vitamin data. These is how it relates to the frontend shop and user profile. Communication between these tiers happen through SQL queries so that the data stored within the database is being called and applied where necessary for example adding a vitamin too the SQL database and the vitamin popping up in the shop.

Data Model



Data Tier

Users

ID	username	password	First_name	Last_name	email	Hashed_password	picture
1	Mike123	p@w0rd!2	Mike	Ross	ross@gmail.com	Wsefwqef21243r*(&@3	Image.png

Vitamins

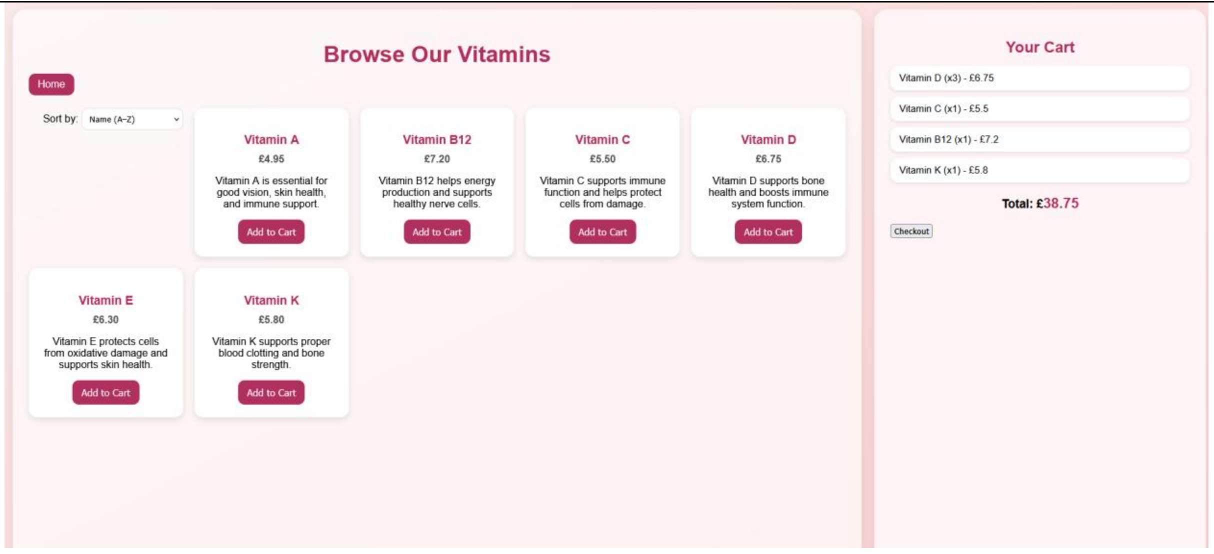
ID	vitamin	price	description
1	Vitamin C	4.50	The body needs vitamin C to form collagen which is the fibrous protein part of the bone.

Diagram of my data model

My data model primarily consists of two elements: users and vitamins. The user tables store information that is used from the register page input into the SQL database for example when users enter their username and password this is then saved into the database. The vitamins table is used to store vitamins with a price and a description. This is then used and displayed in the shop for users to browse and select vitamins to purchase.

Functionality

My application provides a complete user facing experience around vitamins through browsing and recommending users suitable vitamins. When a user first enters the webpage, they first encounter the home page with a navigation bar which would allow them to explore around the websites. The shop page which in the codebase is known as '/browse,' is utilised to display all the available vitamins which is taken from the SQL database.



Browse Our Vitamins

Home

Sort by: Name (A-Z)

Vitamin A
£4.95
Vitamin A is essential for good vision, skin health, and immune support.
Add to Cart

Vitamin B12
£7.20
Vitamin B12 helps energy production and supports healthy nerve cells.
Add to Cart

Vitamin C
£5.50
Vitamin C supports immune function and helps protect cells from damage.
Add to Cart

Vitamin D
£6.75
Vitamin D supports bone health and boosts immune system function.
Add to Cart

Vitamin E
£6.30
Vitamin E protects cells from oxidative damage and supports skin health.
Add to Cart

Vitamin K
£5.80
Vitamin K supports proper blood clotting and bone strength.
Add to Cart

Your Cart

Vitamin D (x3) - £6.75

Vitamin C (x1) - £5.5

Vitamin B12 (x1) - £7.2

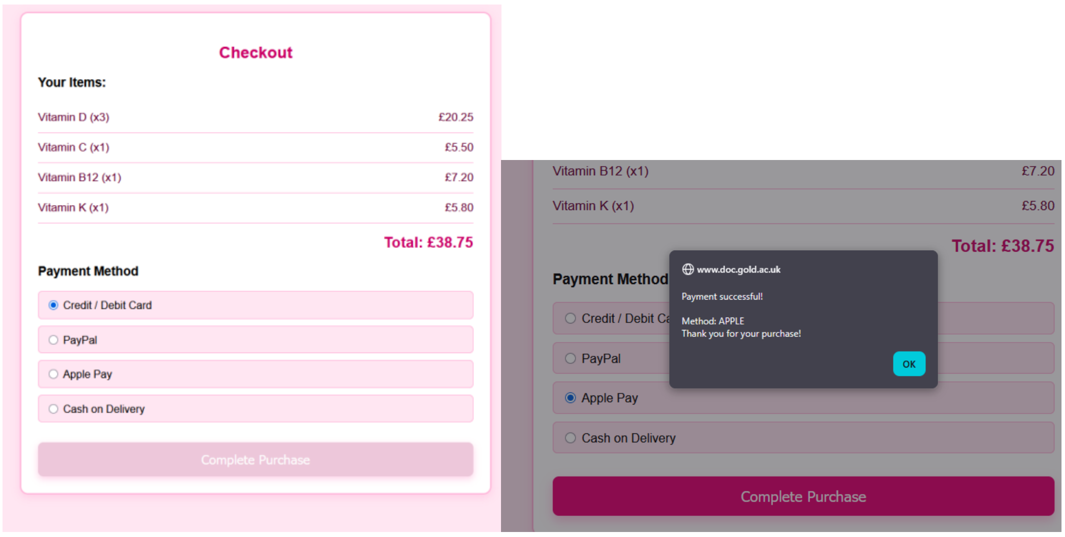
Vitamin K (x1) - £5.8

Total: £38.75

Checkout

Screenshot of my shopping page

Each vitamin with its name, price and description is presented on the screen allowing for users to add an item to their cart during a session storage-based system so that they do not lose their cart progress whilst they are logged in and browsing around the website. The total price is then accumulated, and the user can checkout their basket for purchase. Id also like to note that due to the nature of the assignment, I did not add a real checkout function and decided to simulate the act of checking out for the user. The checkout feature allows for users to choose their payment method and then the cart is cleared, and the user returns to the homepage once their purchase has been competed.


Checkout

Your Items:

Vitamin D (x3)	£20.25
Vitamin C (x1)	£5.50
Vitamin B12 (x1)	£7.20
Vitamin K (x1)	£5.80

Total: £38.75

Payment Method

[✓ Credit / Debit Card]
PayPal
Apple Pay
Cash on Delivery

Complete Purchase

Vitamin B12 (x1)	£7.20
Vitamin K (x1)	£5.80

Total: £38.75

Payment Method

Credit / Debit Ca
PayPal
[✓ Apple Pay]
Cash on Delivery

www.doc.gold.ac.uk

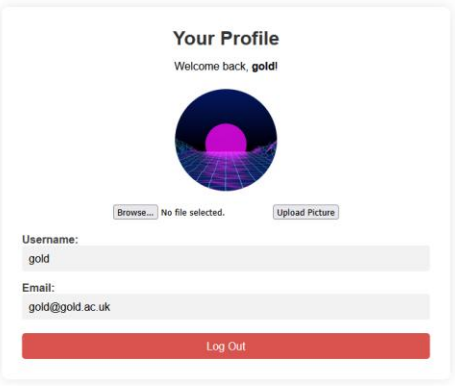
Payment successful!

Method: APPLE
Thank you for your purchase!

Screenshot of checkout page

In addition to the shopping feature, users are also able to create an account on the register page by providing their personal details as we see in the the data tier model. Once the details are received my codebase then sanitises the inputs as well as the passwords being hashed using bcrypt before being stored in MySQL. After registration, when the user wants to login, the login page then checks these credentials and then grants the user access whilst also starting a session, this also includes restricted pages also becoming accessible e.g. Profile.

Once the user is logged in, they also have the option to view their profile which displays their account information alongside with the ability to log out or upload their own profile picture which you can see in the screenshot below.


Your Profile

Welcome back, **gold**!

Browse... No file selected.

Username:
gold

Email:
gold@gold.ac.uk

Screenshot of profile page

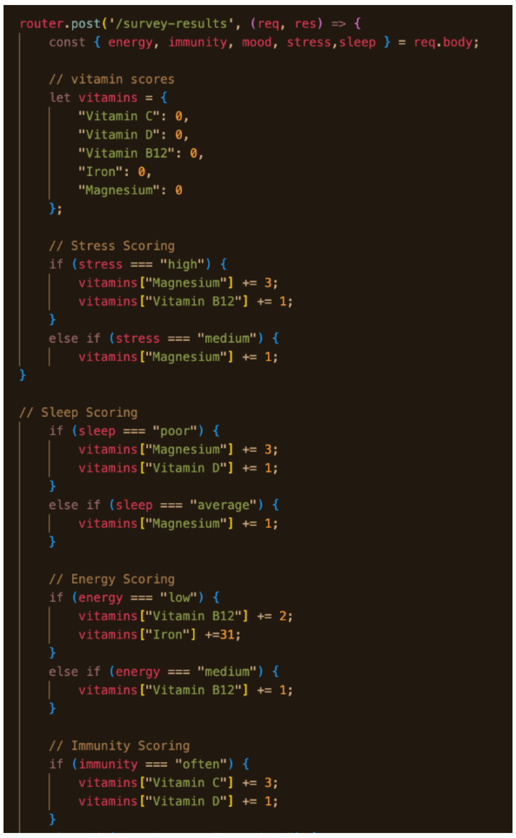
In addition to some of the core features, another functionality would be the vitamin search interface where users are allowed to search for their own vitamin by name and read a description on what that vitamin specialises in.

My website also includes a survey page, which includes a weighted quiz that calculates a result based off the user's input. The user answers a series of quiz questions which then generate a final vitamin recommendation supplement for them to take, for example the user may be recommended Vitamin C. Once the user is taken to the quiz results page, they are also able to see the weighted score between all the vitamins as well as the final recommendation.

Optional: Advanced Techniques

Weighted Quiz

One advanced technique in which I incorporated into my project was a vitamin survey that consisted of a simple weighted scoring system that recommended a vitamin based off the user's response. Each answer contributed to a score and through this the highest school is then finally suggested to the user.



```
router.post('/survey-results', (req, res) => {
    const { energy, immunity, mood, stress,sleep } = req.body;

    // vitamin scores
    let vitamins = {
        "Vitamin C": 0,
        "Vitamin D": 0,
        "Vitamin B12": 0,
        "Iron": 0,
        "Magnesium": 0
    };

    // Stress Scoring
    if (stress === "high") {
        vitamins["Magnesium"] += 3;
        vitamins["Vitamin B12"] += 1;
    }
    else if (stress === "medium") {
        vitamins["Magnesium"] += 1;
}

// Sleep Scoring
    if (sleep === "poor") {
        vitamins["Magnesium"] += 3;
        vitamins["Vitamin D"] += 1;
    }
    else if (sleep === "average") {
        vitamins["Magnesium"] += 1;
    }

    // Energy Scoring
    if (energy === "low") {
        vitamins["Vitamin B12"] += 2;
        vitamins["Iron"] +=31;
    }
    else if (energy === "medium") {
        vitamins["Vitamin B12"] += 1;
    }

    // Immunity Scoring
    if (immunity === "often") {
        vitamins["Vitamin C"] += 3;
        vitamins["Vitamin D"] += 1;
    }
```

Screenshot of weighted quiz
Source : routes/vitamins.js line 49

As you can see here, the algorithm begins by defining a score object and with each response the score of the vitamins increase this is implemented using if statements.

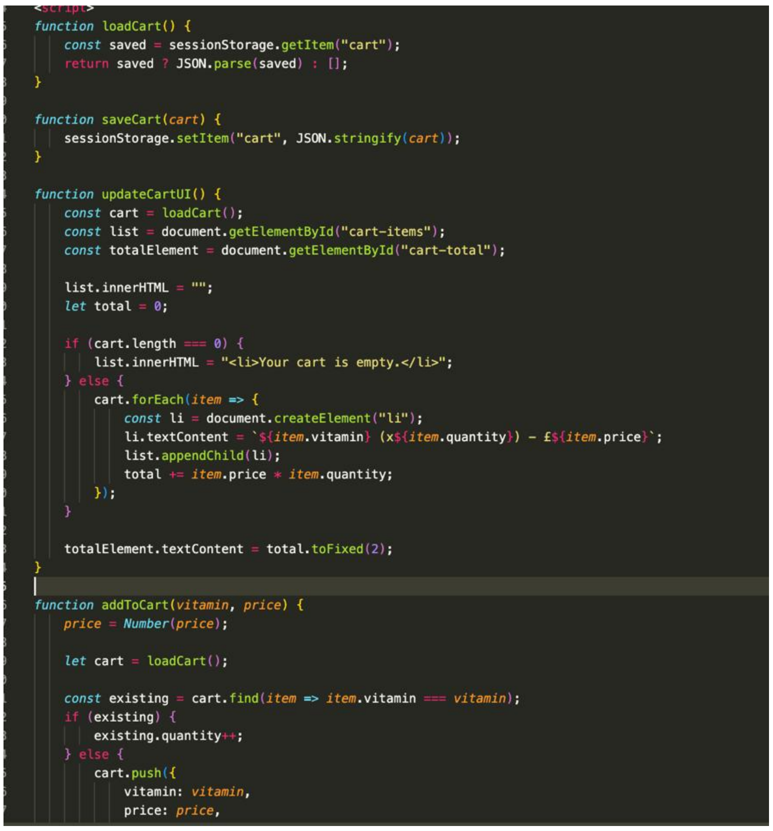
The advantage of using weighted scores is that it can increase and decrease the results therefore producing an outcome that is more tailored and specific to the user.

Once all the answers are processed the vitamins with the highest score is then use an output and on the results page. The results page also returns all the values of the other vitamins as well so user can see the scoring.

Sessions

Within the code base there are two types of sessions that are used, firstly we utilise session-based authentication, this helps maintains the login state of the user on the browser. Secondly, we also use sessions storage functionality which allows the user to add items to their basket and saved their basket however an evaluation of this would be that when a user logs out their basket is no longer saved. The code snippet below explains in more detail the logic behind the cart and its functionality in detail with the source of the code underneath the screenshot.

Storage



```
<script>
function loadCart() {
    const saved = sessionStorage.getItem("cart");
    return saved ? JSON.parse(saved) : [];
}

function saveCart(cart) {
    sessionStorage.setItem("cart", JSON.stringify(cart));
}

function updateCartUI() {
    const cart = loadCart();
    const list = document.getElementById("cart-items");
    const totalElement = document.getElementById("cart-total");

    list.innerHTML = "";
    let total = 0;

    if (cart.length === 0) {
        list.innerHTML = "<li>Your cart is empty.</li>";
    } else {
        cart.forEach(item => {
            const li = document.createElement("li");
            li.textContent = `${item.vitamin} (x${item.quantity}) - £${item.price}`;
            list.appendChild(li);
            total += item.price * item.quantity;
        });
    }

    totalElement.textContent = total.toFixed(2);
}

function addToCart(vitamin, price) {
    price = Number(price);

    let cart = loadCart();

    const existing = cart.find(item => item.vitamin === vitamin);
    if (existing) {
        existing.quantity++;
    } else {
        cart.push({
            vitamin: vitamin,
            price: price,
```

Source: views/browse.ejs line 55

Function loadCart() {

The first function we see is the load car function this retrieves the cart from the session storage and if it exists then it converts the JSOn string into a JavaScript array otherwise it stays empty. This is used to help maintain the carts' structure.

}

Function saveCart(){

The save cart function converts the caught array into a JSON string and store there under the key cart in session storage this preserves the cart during the session.

}

Function updateCartUI() {

In the update cart UI function, it retrieves the call and updates the HTML list and total price dynamically it then loops through and creates a list for each item in the cart.

}

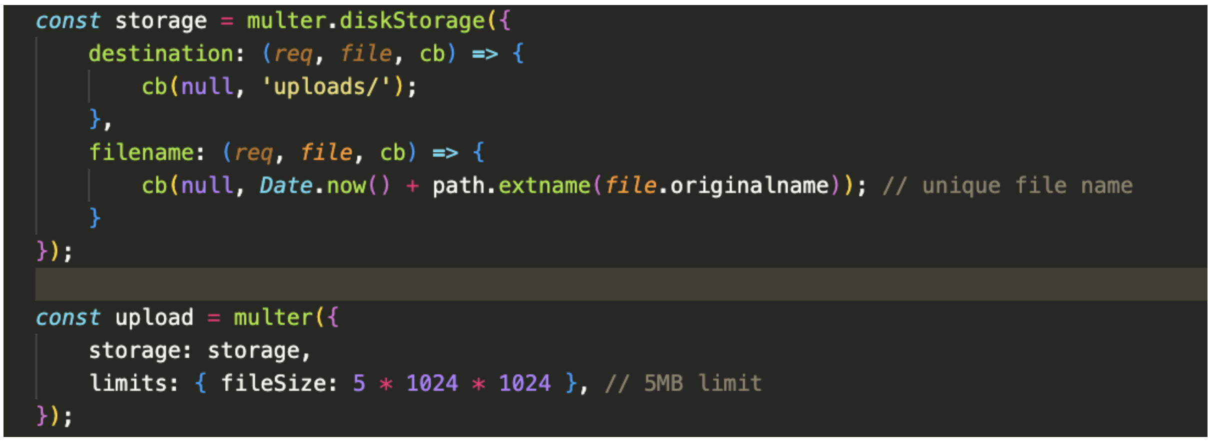
Function AddToCart(){

Finally, the add to cart function ensures the prices and number and loads the existing court it also checks if an item already exists in the cart an increase its quantity to avoid having multiple items the all the same.

}

File upload handling

Within my website users also can upload profile pictures through an upload form their profile once they have looked in. This is done through using Multer and it consists of storage rules and filename handling to prevent collisions.



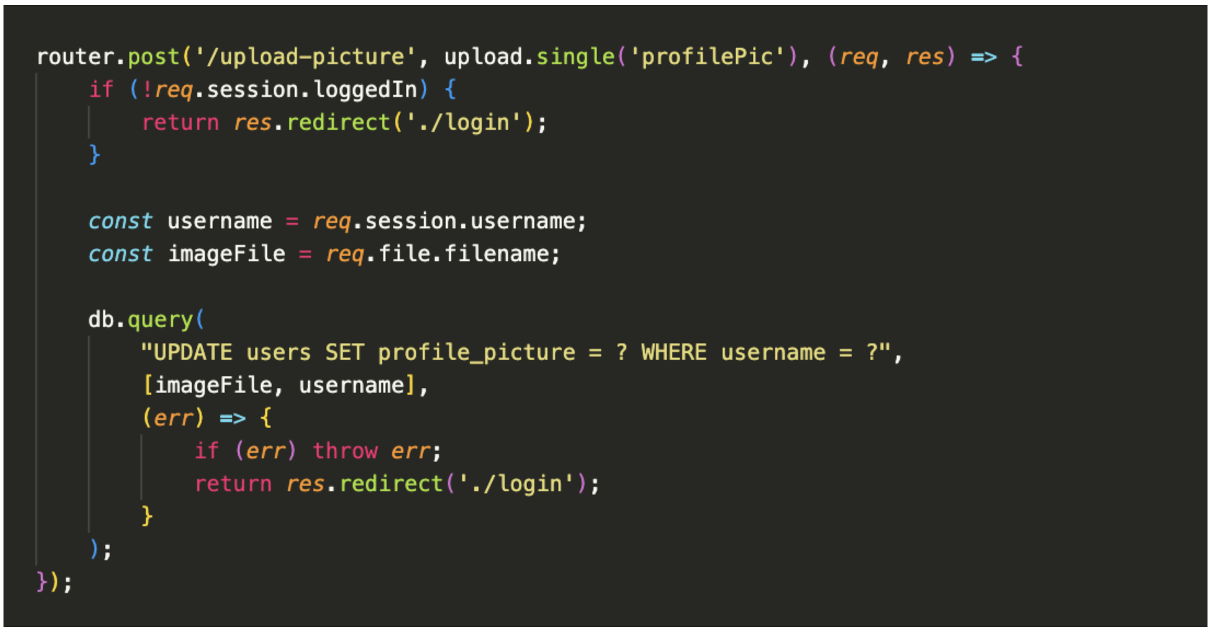
```
const storage = multer.diskStorage({
    destination: (req, file, cb) => {
        cb(null, 'uploads/');
    },
    filename: (req, file, cb) => {
        cb(null, Date.now() + path.extname(file.originalname)); // unique file name
    }
});

const upload = multer({
    storage: storage,
    limits: { fileSize: 5 * 1024 * 1024 }, // 5MB limit
});
```

Source: routes/vitamins.js line 17

Firstly, in this storage variable it specifies where the uploaded files would be located on the server which is in '/uploads'. After this it also specialises a unique file name for each upload and uses the current date an original filename to generate a unique file name this is done to prevent overwriting files with the same name and avoiding collision.

Secondly, we are utilising multiple upload middleware, what Malta does is that it connects Multer to your computer storage session and adds a size constraint for the files in this case 5MB, therefore allowing Multer to reject it if the file is too large.



```
router.post('/upload-picture', upload.single('profilePic'), (req, res) => {
    if (!req.session.loggedIn) {
        return res.redirect('./login');
    }

    const username = req.session.username;
    const imageFile = req.file.filename;

    db.query(
        "UPDATE users SET profile_picture = ? WHERE username = ?",
        [imageFile, username],
        (err) => {
            if (err) throw err;
            return res.redirect('./login');
        }
    );
});
```

Source: routes/vitamins.js line 131

In this code we created a post route for uploading the profile picture this is done using the ‘upload.single(‘profilePic’), and it tells Multer to accept one file from a form labelled as ‘profilePic’.

After this it then links the uploaded picture to the correct user and updates the database so that the picture uploaded correlates with the correct user profile so that every time the user logs into the database their profile picture will remain.

AI Declaration

Throughout this project, I used ChatGPT as a learning aid to clarify concepts I was unsure about, such as explaining JavaScript logic, Express routes, session handling, Multer file uploads, and general debugging assistance.

I also requested help generating and refining CSS styling for some user interface elements. All code implementation, integration, decision-making, and final development of the application were completed by me.