TechNest – eCommerce Web Application

Week 1 Planning Document

Muhammad Saleem | L00196822 | ATU

This document covers the planning work done in Week 1 of the TechNest project. It includes the full list of requirements, the architecture plan for how the code will be structured, and written descriptions of the wireframes for each page of the application.

1. Requirements

Before writing any code, it is important to clearly define what the application needs to do. The requirements below are split into two categories: functional requirements (what the app must actually do) and non-functional requirements (how it should behave in terms of quality, performance, and usability).

1.1 Functional Requirements

Functional requirements describe the specific features and actions the application must support. Each requirement has been given a unique ID so it can be tracked and referenced throughout the project.

ID	Requirement	Type
FR01	The application shall display a list of products loaded from a JavaScript data file.	Must
FR02	Each product entry shall show the product name, image, price, and category.	Must
FR03	The user shall be able to filter products by category using buttons or a dropdown.	Must
FR04	The user shall be able to search for products by name using a search input.	Must
FR05	The user shall be able to sort products by price (low to high, high to low).	Must
FR06	Clicking on a product shall navigate the user to an individual product detail page.	Must
FR07	The product detail page shall load the correct product using a URL parameter (e.g. ?id=3).	Must
FR08	The product detail page shall show the full description, image, price, and an add-to-basket button.	Must

FR09	The user shall be able to add a product to the basket from the product detail page.	Must
FR10	The user shall be able to view all items currently in their basket.	Must
FR11	The user shall be able to update the quantity of an item in the basket.	Must
FR12	The user shall be able to remove an item from the basket.	Must
FR13	The basket shall automatically recalculate and display the total price when items change.	Must
FR14	Basket contents shall be saved to localStorage so they persist if the page is refreshed.	Must
FR15	The checkout page shall include a form with fields for name, email, address, and card details.	Must
FR16	The checkout form shall validate all fields before allowing submission.	Must
FR17	A confirmation message shall be shown to the user after a successful checkout.	Must
FR18	The navigation bar shall display the current number of items in the basket.	Want
FR19	The user shall be able to continue shopping from the basket page without losing basket contents.	Want

1.2 Non-Functional Requirements

Non-functional requirements describe the quality standards the application should meet. These are not specific features but they are just as important for making sure the app works well for the end user.

ID	Requirement	Type
NFR01	The application shall be fully responsive and usable on desktop, tablet, and mobile screen sizes.	Must
NFR02	All pages shall load quickly in a modern browser without any build tools or server required.	Must
NFR03	The codebase shall be organised into separate modules (data, models, pages) to keep concerns separated.	Must
NFR04	All JavaScript shall use ES6+ syntax including classes, modules, arrow functions, and template literals.	Must
NFR05	The application shall work correctly in Google Chrome and Mozilla Firefox at minimum.	Must
NFR06	Variable and function names shall be descriptive and consistent throughout the codebase.	Must
NFR07	The visual design shall be consistent across all pages, using the same fonts, colours, and spacing.	Must
NFR08	Error messages and validation feedback shall be clear and easy for the user to understand.	Must

NFR09	The application shall not rely on any back-end server or database — everything runs client-side.	Must
NFR10	The project shall use Git for version control, with regular commits after each meaningful change.	Must

2. Architecture Plan

This section describes how the project will be structured before any code is written. Planning the architecture early means that the code should be easier to manage, read, and build on week by week. The main idea is to keep different types of code in separate files so that each file has one clear job.

2.1 Technology Stack

The following tools and languages will be used throughout the project:

- HTML5 – for building the structure and content of each page
- CSS3 – for custom styling on top of Bootstrap
- Bootstrap 5 – for the responsive grid layout and ready-made UI components like buttons and cards
- JavaScript (ES6+) – for all application logic, using classes, modules, and DOM manipulation
- localStorage API – for saving the basket contents in the browser so it is not lost on page refresh
- Visual Studio Code – the code editor used for development
- Git and GitHub – for version control and tracking progress week by week
- Chrome Developer Tools – for testing, debugging, and checking responsiveness

2.2 Folder and File Structure

The project will be arranged into the following folders. Each folder has a clear purpose so it is easy to find the right file when making changes.

Path	Purpose
`index.html`	Home / landing page
`products.html`	Product listing page with filtering and search
`product-detail.html`	Individual product detail page
`basket.html`	Basket page showing all added items
`checkout.html`	Checkout form and confirmation page
`css/style.css`	All custom CSS styles for the project
`js/data/products.js`	The product data stored as JavaScript objects
`js/models/Product.js`	ES6 class representing a single product
`js/models/Basket.js`	ES6 class handling all basket logic (add, update, remove, total)
`js/models/Storage.js`	ES6 class handling read/write to localStorage
`js/pages/`	Folder for page-specific JavaScript files (one per page)
`assets/images/`	Folder for product images and any other image assets

2.3 Module Responsibilities

The JavaScript code will be split across three main classes. Each class has one specific job, which helps keep the code clean and makes it easier to fix bugs or add new features later.

Product Class (js/models/Product.js)

This class will represent a single product in the shop. When the application loads, each product from the data file will be turned into a Product object. The class will store properties like the product ID, name, description, price, category, and image path. It may also include a method to format the price nicely for display (e.g. adding the euro sign and two decimal places).

Basket Class (js/models/Basket.js)

This is the most important class in the project. It will manage everything to do with the shopping basket. Its responsibilities include adding a product to the basket, updating the quantity of an existing item, removing an item completely, calculating the total price of all items, and returning the current list of items. It will work closely with the Storage class to make sure the basket is saved and loaded from localStorage.

Storage Class (js/models/Storage.js)

This class will act as a wrapper around the browser's localStorage API. Rather than writing localStorage.getItem() and localStorage.setItem() all over the codebase, these calls will be handled in one place. The Storage class will have methods for saving the basket, loading the basket, and clearing it. This makes it easier to change the storage method later if needed.

2.4 Page Flow

The diagram below describes how a user would typically move through the application. Each arrow represents a navigation action.

Home Page → Products Page → Product Detail Page → Basket Page → Checkout Page → Confirmation Message

The navigation bar will always be visible on every page, so the user can also jump directly to the basket or back to the product listing at any time.

3. Wireframe Descriptions

The wireframes below describe what each page of the application will look like. These are text-based descriptions that can be used as a guide when drawing the actual wireframes on paper or in a tool like Excalidraw. Each page includes a description of its layout and the key elements it needs to contain.

3.1 Home Page (index.html)

Page: Home	
Layout	Full-width header at the top. A large hero/banner section in the middle. A row of category buttons below the hero. A footer at the bottom.
Key Elements	Navigation bar with the TechNest logo on the left, links to Products and Basket on the right, and a basket item count badge. Hero section with a welcome heading, a short line of text describing the shop, and a 'Shop Now' button that links to products.html. A row of 3–4 category buttons (e.g. Laptops, Phones, Accessories, All Products). A simple footer with the project name and student number.

3.2 Products Page (products.html)

Page: Product Listing	
Layout	Navigation bar at the top. A filter/search bar below the nav. A grid of product cards filling the main content area. Footer at the bottom.
Key Elements	Navigation bar (same on all pages). A search input on the left and a sort dropdown on the right. A row of category filter buttons below the search bar. A grid of product cards — each card has a product image at the top, the product name, the category label, the price, and a 'View Details' button. The grid should be 3 columns on desktop, 2 on tablet, 1 on mobile.

3.3 Product Detail Page (product-detail.html)

Page: Product Detail	
Layout	Navigation bar at the top. A two-column layout in the main content area — image on the left, product info on the right. Footer at the bottom.
Key Elements	Navigation bar. Left column: a large product image. Right column: the product name as a heading, the category shown as a small badge, a paragraph of product description text, the price shown in large bold text, a quantity input (number field defaulting to 1), and an 'Add to Basket' button. A small 'Back to Products' link above the two-column section.

3.4 Basket Page (basket.html)

Page: Basket	
Layout	Navigation bar at the top. A two-column layout — basket items list on the left (wider), order summary on the right (narrower). Footer at the bottom.
Key Elements	Navigation bar. Left side: a table or list of basket items. Each row shows the product image (small), product name, unit price, a quantity input, a remove button (X), and the line total. Right side: an order summary box showing the subtotal, and a 'Proceed to Checkout' button. If the basket is empty, a message saying 'Your basket is empty' and a link back to the products page.

3.5 Checkout Page (checkout.html)

Page: Checkout	
Layout	Navigation bar at the top. A two-column layout — checkout form on the left, order summary on the right. Footer at the bottom.
Key Elements	Navigation bar. Left side form with three sections: (1) Personal Details — fields for First Name, Last Name, Email Address. (2) Delivery Address — fields for Address Line 1, Address Line 2 (optional), City, Postcode. (3) Payment Details — fields for Card Number, Expiry Date, CVV. A 'Place Order' button at the bottom of the form. Right side: a small summary of the basket items and the final total. After the form is successfully submitted, the main content area is replaced with a confirmation message: 'Thank you for your order!' and the basket is cleared.

4. Week 1 Summary

The following tasks were completed during Week 1:

- Defined all functional and non-functional requirements for the TechNest application
- Identified the five main pages of the application and described the layout and elements for each
- Planned the modular folder and file structure for the project
- Defined the responsibilities of the three main JavaScript classes: Product, Basket, and Storage
- Created a GitHub repository and set up the initial folder and file structure with an initial commit

Next week (Week 2) the focus will be on building the semantic HTML structure for all five pages, including a shared navigation bar and footer.