**Design, Analysis and Implementation Decisions for Nebula - Junior Coding Community**

**My Initial Designs**

The goal of this project was to design a visually appealing, responsive web app aimed at beginner and junior coders. I wanted to build something that felt fun, encouraging, and community-driven — a space where users could explore content, connect, and grow.

Inspired by the infinite nature of learning to code, I leaned into a space-themed aesthetic. The app needed to look modern and work seamlessly across desktop, tablet, and mobile devices.

The web app includes the following pages:

**Login**

* A clean login form allows users to securely access blog posts and other features.
* The NEBULA logo appears at the top left for branding consistency and in the middle on smaller mobile devices.
* Navigation buttons use JavaScript enhancements for interactivity and responsiveness.
* Flask-WTF is used for form validation and CSRF protection.
* DumbShelf is used for user storage in this small-scale app.

**Create An Account**

* The registration page’s design is similar to the login page.
* Flask-WTF handles input validation for user creation (password matching & email validation).
* Navigation buttons use JavaScript enhancements for interactivity and responsiveness.

**Home Page**

• The web app’s main page after login - an area to share posts, ideas, and encouragement.

• Users can create posts using Flask-WTF, which are saved using DumbShelf.

• Posts include author, timestamp, content, like count, and optional comments.

• Like and comment functionality uses both Python and AJAX (via JavaScript) to keep things time-efficient.

**Merchandise Page**

• Responsive product grid showcasing NEBULA-themed items (shirts, mugs, etc.).

• “Add to Cart” buttons allow for quick selection. LocalStorage is used to persist the cart between page loads.

• A cart dropdown shows selected items, quantity, and total price.

* Pricing is displayed dynamically, formatted to two decimal places using JavaScript.
* As no database is in place, when the checkout button is clicked, it displays a custom error 404 page.

**Resources Page**

**•** A list of beginner-friendly coding resources, organised for quick access, along with two featured tutorial videos.

• This page is only accessible to logged-in users, keeping the content focused and relevant to engaged learners.

* The goal is to give junior developers a helping hand— from documentation to tutorials and interactive platforms — all in one place.

**Main design features**

For NEBULA, I went with a dark, space-inspired theme to capture the idea of exploration and growth. The main background features a starry Milky Way image, with deep purples, pinks, and navy tones. To keep things readable and visually balanced, I used soft whites and brighter purples for text, icons, and borders.

The “Login” and “Create An Account” page have a simple design and form layout, using Javascript and Python for functionality and validation. Clean, simplified buttons allow for easy user navigation.

Action buttons like “Post,” “Comment,” and “Add to Cart” use bold accent colours such as electric blue and violet, helping them stand out without overwhelming the design. These choices were made with both accessibility and aesthetics in mind, making it easy for users to navigate and interact with the site.

I designed the NEBULA logo myself using Canva, and created separate versions for desktop and mobile to make sure it stays sharp and clear across all devices. It sits in the top-left corner of the page, with navigation links to the right — a layout that feels clean and balanced, especially when paired with the responsive hamburger menu on smaller screens.

The Resources page was designed to be clean and easy to scan, with each resource presented in its own card-style layout. The goal here was to give junior developers a quick and organised way to find helpful tools and tutorials, without feeling overwhelmed.

The Merchandise page uses a responsive grid layout to display products like t-shirts, mugs, and hats. Each product is shown with an image, name, and price. Images were created by myself using an online image generator. The “Add to Cart” button on each product is styled consistently across all items, with cart updates handled smoothly through JavaScript. A cart dropdown in the header gives users a quick view of their selected items and total price. The design allows full functionality whilst being visually appealing to the user across all devices.

**My Development Process**

When I started working on the NEBULA project, my idea was to create a web app where beginner and junior coders could be supported, find useful content, and connect with each other. I wanted the app to feel fun and engaging but still functional and clean. Since I already had a Canva account, I started by designing a logo to help me visualise the web app’s theme. Once I had a version I was happy with, I moved into VS Code and began experimenting with dark-themed colour palettes that complemented the logo.

I decided to position the logo in the top-left corner of the site with the navigation bar to its right — this layout felt more visually appealing compared to a centre-aligned logo and menu. I also wanted to ensure responsiveness, so I implemented alternate logo sizes, positions and a hamburger menu for tablets and mobile devices.

As I planned out the app’s features, such as login forms, create account forms, comments, posts etc, I saw took the opportunity to incorporate Flask and Python as much as possible — for data handling and form validation, not just routing. I used Flask-WTF to manage login, registration, and posting forms, and introduced DumbShelf, a lightweight shelf-based storage wrapper, to handle user and blog post data without needing a full database for this version of the app. Using JSON files was an initial idea, however this came with a lot of challenges.

For the blog page, I focused on creating an area where users could write posts, like others’ content, and leave comments. At first, I was unsure how I’d structure the comment system, but after researching a few forums, I decided on a simple setup that looked clean and consistent — using Python for storing data and JavaScript to toggle comment visibility.

The resources page was one of the more straightforward pages to develop. I wanted it to be simple and useful — just a curated list of links to help junior coders get started. I styled each resource with its own card and kept the layout flexible using Bootstrap’s grid system so it would adapt nicely to different devices.

To add a bit of fun to this page, I added in a joke restAPI which generates a new programming joke each time the page is loaded.

One of the more interactive parts of the app is the merchandise page, which was a challenge to build. I used JavaScript to handle the “Add to Cart” functionality and localStorage to persist cart data between sessions. Originally, I had a simple list view for the products, but after experimenting with a responsive grid and adding hover effects, it felt much more like a modern shop section. The cart dropdown at the top updates in real time, showing the selected items and total. As mentioned above, in order to showcase on of my custom error pages, I included a checkout button in the dropdown cart which generates a 404 error.

After I had the core features of the web app working, I focused on responsiveness and finalised styles. I added media queries to make sure layouts worked across screen sizes and changedimage assets for .webp formats to help improve loading times. I also ran Lighthouse audits and addressed issues like image scaling and cookie security flags. In making small changes, the scores on the lighthouse report improved dramatically.

Throughout the project, I constantly tested and adjusted — from small tweaks such as adjusting button styles to larger tasks like restructuring the merchandise page. I learned a lot in the development of this project, especially about how to merge functionality with visual consistency and how to structure my code so it’s easier to maintain and grow in future course projects.

**Challenges faced during development**

One of the first challenges I ran into was working without a standard database, as this is something we have not yet studied in our coursework. As a result, my only options were to use shelve or JSON files to store data. While shelve is great for saving Python objects easily, I quickly discovered that it relies on back-end systems like dbm.gnu, which aren’t supported on Render.com. To get around this, I built a DumbShelf wrapper that forces shelve to use dbm.dumb, which is much more widely supported. This small change made a big difference and allowed the app’s data — like users, posts, and cart items — to save and load reliably both locally and online. Originally I was using JSON files to store posts and user data however I then ran into issues as JSON files require manual saving.

Another challenge was working with forms in Flask. I used Flask-WTF to handle all forms in the app, including login, registration, blog posts, and comments. Flask-WTF made things a lot easier by including built-in validation and CSRF protection. It did take some time to understand how CSRF tokens work and how to make sure they were properly included in every form, as I originally was using javascript/standard python validation. Once I understood it, using validate\_on\_submit() helped keep everything secure and user-friendly.

Getting JavaScript to work smoothly with my Flask backend took a bit of trial and error. I wanted certain features — like post likes and the shopping cart — to update without reloading the page. For the cart, I used localStorage to save items on the client side and for the blog posts, I used POST requests to update likes and comments through Flask routes.

Styling the site to be responsive on all devices was another area that took time and testing. Things looked fine on desktop from the start, but I had to write quite a few media queries to make sure it looked just as clean and functional on tablets and phones. The mobile hamburger menu and cart dropdown both needed extra updates to behave properly on smaller screens.

Deploying the app to Render brought its own set of challenges. I had to set up a Procfile, update my requirements.txt file and define environment variables like SECRET\_KEY to keep sessions and CSRF protection working. Testing the deployed app revealed small issues throughout this process that didn’t appear during local development — like cookie handling and file path mismatches — which I had to go back and fix in order to have the web app working properly.

Overall, this project was a massive learning opportunity and has given me great insight into how the backend works and blends in seamlessly with frontend code.