Portfolio Phase 3: CosFormulator application abstract

CosFormulator is a web application designed to help cosmetic chemists to efficiently manage and develop their own formulations. The app allows users to build cosmetic formulations with real-time validation of ingredient percentages and weights, aiming to simplify their creative process of developing unique products.

Users can use this application to:

- Sign up into the system and manage only their own formulations within a secure user account
- Creating new formulations, as well as editing or deleting existing ones
- Input multiple ingredients of the users' choice
- Automatically calculate and validate total ingredient percentages to ensure the sum of all ingredients is 100%
- Dynamically recalculate ingredient sample weights based on the specified batch size for experimental mixing in the lab
- Create new ingredients dynamically during formulation creation, if the desired ingredient is not yet in the database
- Access a shared ingredient database, which is available to all users of the application

Technical approach

I followed an iterative development process to implement the core features of the application. For each feature, I began with a visual idea - a sketch or a table and then translated it into the code. Once implemented, I moved to manual testing to verify correct behavior and eliminate bugs.

1. **Database design**

I started with a sketch of the database schema to clarify which data needed to be stored and how different entities would relate. I used SQLite as the database system, which is lightweight and file-based, making it suitable for small projects and development.

I designed two main models: Ingredient and Formulation, which are connected through a many-to-many relationship. This reflects the logic that a formulation can contain multiple ingredients and a single ingredient can be used in many formulations. To manage additional data (e.g., percentage of each ingredient in a formulation), I implemented a through model, allowing more control over the relationship.

2. **Backend design**

The backend was built using the **Django framework**. I initially used the Django admin interface to test and visualise the model structure and interactions. In the end, the models included more attributes than were used, leaving room for future feature expansion.

Next, I implemented the **user authentication system** using Django's built-in User model and templating tools. I created templates for the home page, log-in, and sign-up, and manually tested the authentication workflow.

The following iteration focused on formulation functionality: creating, editing, and validating formulations. I added backend validation to ensure the sum of ingredient percentages equals 100%.

3. **Frontend design**

Basic design features were added to the system during the development of individual temples. I used Tailwind CSS and Flowbite blocks to create a responsive and modern design. Dynamic behavior was implemented using **Vanilla JavaScript**, which helped me deepen my understanding of DOM manipulation.

My biggest challenge was dynamic behavior. I used a modal form to create a new ingredient during formulation form filling, in case users cannot find a required ingredient in the database. They have the possibility to open a modal to input a new material into the ingredient database without interrupting their current progress with the formulation.

Lesson learned

Throughout this project, I had the opportunity to explore several new tools and deepen my understanding of Django. Replacing Bootstrap with Tailwind CSS and the Flowbite component library introduced me to a utility-first approach to styling, giving me more precise control over UI design. A major learning outcome was the implementation of Django formsets and the use of inlineformset_factory for managing multi-row input forms.

One of the most challenging parts of the project was managing dynamic form behavior using Vanilla JavaScript, particularly during formulation saving. I had to ensure that deleted ingredient rows (visually hidden by the user) were correctly marked so that they would not be saved to the database, which required careful synchronisation between the frontend and Django's formset handling.