

# Final Project | CSC337

- Overview
- Frontend
- Backend
- Timeline

*Author:* Christian P. Byrne

## Overview

This webapp will server as a wardrobe manager app. It will work similarly to how a kitchen manager app would work – by saving recipes (outfits) and suggesting recipes based on available foods (clothes) and defined rulesets/dictionaries (publicly available cookbooks).

Theoretically, the user will, over time, enter information about the items in their closet and about outfits that they wear. The extent of detail entered will depend on the user, but they will have the option to fill out a significant number of fields per item/outfit. The most important fields will be those inputs that are crucial to the app's suggestion algorithms.

The suggestion algorithms will be used to generate new outfits for the sake of convenience – in a way acting as a stylist. The algorithm will most likely work by (1) using a color matching algorithm, (2) using some ruleset derived from fashion rules I find online, or (3) use patterns found in outfits that the user already has entered and given high ratings to. Or, some combination of these.

But perhaps most helpful, the app will simply serve as a log of outfits. In theory it will be beneficial to have a list of outfits (characterized by each constituent item) and a rating system attached to those outfits – as well as other sorting parameters like temperature, formality, and so on. If someone spends a portion of their day trying to find the correct clothing items – constantly forgetting the full extent of their wardrobe and the previous combinations – this app can help them.

## Frontend

### Login / Register Page

### Add Items Page

The required fields will be indicated somehow.

For the item's colors field (the colors of the item the user is adding), I will try to use a library that can parse colors from a picture; but if that turns out to be unrealistic, I will just put a color selector input.

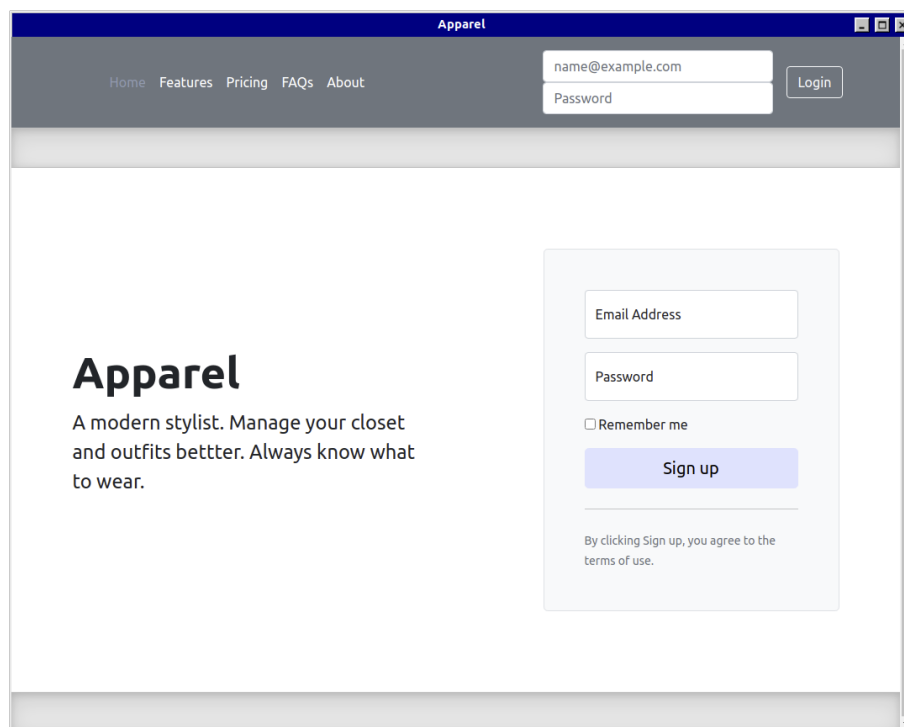


Figure 1: login page

Apparel | Setup

HomeAdd ItemGenerate OutfitWardrobeOutfitsStatsAnalyze

Apparel | Add Item

Search...

### Add an Item

Start uploading your wardrobe by adding an Item.  
Most fields are optional.

Short Description

Something memorable

Rating

Category

Choose...

Sub-Category

...

Sub-Type

...

Add Style Tag

Fit

Choose...

Length

Choose...

Number Sizing

Letter Sizing

Materials

Brand

Prada

Purchase Details

Location

Date

\$

Condition

Wash Instructions

Submit

Clear

ShirtPantOuterwearMore

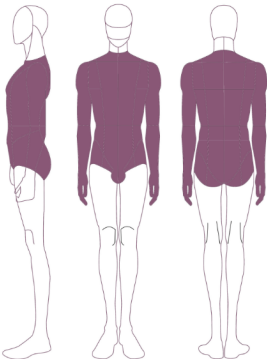


Figure 2: add-item page

## Log Outfit Page

It would be ideal to add a lot of functionality to these forms.

I will try to make sections that are collapsible so there is not so much information displayed at once.

The mannequin image should change colors based on the current selection. I have a sketch template that I am photoshopping and am just going to rotate the `<img> src` attribute in some event handler.

The list on the bottom left will update as new items are added, and there should be as much freedom as possible in terms of how many items a user can add per outfit.

The screenshot shows a web browser window with the title 'Apparel | Add Outfit'. The browser's address bar shows 'localhost:5500/public\_html/add-outfit.html#'. The page has a navigation bar with links: 'Home', 'Add Item', 'Generate Outfit', 'Wardrobe', 'Outfits', 'Stats', and 'Analyze'. Below the navigation bar is a search bar labeled 'Apparel | Log Outfit' with a 'Search...' input field. The main content area is titled 'Log an Outfit' and contains the following form elements:

- Name:** A text input field with the placeholder text 'Something short and memorable'.
- Rating:** A text input field.
- Category:** A dropdown menu with 'Choose...' selected.
- Sub-Category:** A dropdown menu with '...' selected.
- Sub-Type:** A dropdown menu with '...' selected.
- Add Style Tag:** A dropdown menu.
- Formality:** A dropdown menu.
- Setting:** A dropdown menu.
- Event:** A dropdown menu.
- Last Worn:** A text input field with the placeholder 'mm/dd/yyyy' and a calendar icon.
- Wear Count:** A text input field with the placeholder 'Times'.
- Temperature:** A text input field with the value '56' and a dropdown arrow.
- Weather:** A row of checkboxes for 'Rain', 'Snow', 'Sun', 'Windy', and 'Humid'.
- Notes:** A text area.
- Item List:** A list of three items, each with a category and an 'unset' button:
  - 1. **Upperbody** (Shirt | Outerwear | Hat | Glasses) - unset
  - 2. **Lowerbody** (Pant | Belt | Shorts | Skirt | Shoes) - unset
  - 3. **Accessory** (Watch | Necklace | Earrings | Scarf) - unset
- Buttons:** 'Submit' and 'Clear' buttons.

To the right of the form is a mannequin template with three views (side, front, back) and a color selection dropdown menu with options: 'Shirt', 'Pant', 'Outerwear', and 'More'.

Figure 3: log outfit page

## Generate Outfit Page

1. Start from one item or one color
2. Get recommendations based on color algorithm, defined rules, and relationships in settings

## Browse Wardrobe Page

Browse user's items.

## Browse Outfits Page

Browse user's outfits.

## Wardrobe Analytics Page ?

If time permits and it's realistic to implement, some analytics based on the data the user passes. For example, where the user spends most money in terms of clothing type.

## Backend

### Authentication

Authentication middleware handler that was taught in the lecture will be passed to `Express.all()` probably, authenticating all routes except `/login`, `/register`, (and `/?`).

Then store username with `sessionStorage` and create a global function to add a username/password attribute to Ajax POSTs – also allowing the user document to be updated when a new item is posted.

Use the `setInterval` sessionkeys method outlined in the lecture for security.

### Image Upload

I will use `Multer` from PA10 for image uploads. Maybe I should learn how to set permissions on images, because I have just been storing them in the public directory so far.

### Schemas

- User
- Item
- Outfit

Create relationships between **Outfit**, **Item**, and **User** documents using auto-generated `_id` value as key field.

**Outfit** document will have an `items` field that is an object that has the shape `[itemCategory: string]: _id`.

**User** document will have `items` attribute of the shape `_id[]`.

## Routers

- POST Login
- POST Register
- POST Item
- POST Outfit
- GET Wardrobe
- GET Outfits
- GET suggestion
- GET Analytics (*maybe*)

## Dependencies

- Cookie-parser
- Multer
- Node sass
- Sass loader
- Cors

## Document Shapes

```
type ItemFit = "Oversized" | "Loose" | "Casual" | "Fitted" | "Tight";
type BroadCategory = "shirt" | "tshirt" | "sweater";
type MenGeneralSize =
  | "KID"
  | "XXS"
  | "XS"
  | "S"
  | "M"
  | "L"
  | "XL"
  | "XXL"
  | "XXXL"
  | "PLUS";
type WomenGeneralSize = number;
type ShoeSize = number;
type FormalSize = [number, number];

interface ItemColors {
  colors: string[];
  weights: {
    [color: string]: number;
  };
  ordered?: string[];
}

interface ItemMaterials {
```

```

    materials: string[];
    weights: {
        [material: string]: number;
    };
}

interface ItemCondition {
    material: number;
    color: number;
}

interface Item {
    // A description that it is helpful to the user -- something that they can
    // see to help them remember what the item refers to.
    description: string;
    // Broad category.
    category: BroadCategory;
    // Type of the given category.
    subCategory: string;
    // Hihgly specific type of given category.
    type: string;
    // See styles list.
    styles: string[];
    // How the item fits the user.
    fit: ItemFit;
    // TODO: Add details
    length: string;

    color: ItemColors;
    material: ItemMaterials;
    brand: string;
    rating: number;
    size: MenGeneralSize | WomenGeneralSize | ShoeSize | FormalSize;

    // Optional fields.
    purchaseLocation?: string;
    purchaseDate?: Date;
    cost?: number;

    washType?: string;
    condition?: ItemCondition;
}

```

## Timeline

### Day 1-3

- App theme
  - Global CSS variables
  - Global page layout
- Static Resources
  - sketch images
    - \* photoshop fill
- Login / Register
  - HTML
  - Ajax
  - Routers
  - User schema
  - Authentification middleware
  - session storage
- Add Outfit & Add Item
  - connect to user
  - Interfaces and schemas
  - Routers
  - HTML forms
  - Ajax
- Info collection
  - definitions of default field parameters
  - definition for tooltips and info modals

#### Day 4-7

- Refactor routers
- Handle errors from server on client side by displaying to user
- Client color selection inputs
- Color parsing module?
- Image upload?
  - add field to schemas
- Color algorithm
- Define ruleset
  - Info tooltips/modals
- Define algorithmic suggestion
- Client script
  - form fields update based on previous selection
- Specify form required inputs
- Interactive features on generate page

#### Day 8-11

- Browse pages
  - DOM constructor functions
  - Browse wardrobe
    - \* HTML & Ajax



- Browse Items
    - \* HTML & Ajax
- Generate page features
  - Collapsible sections
  - Navbar collapse feature
  - Dark mode
- Tooltips
- Navbar and button hrefs

## **Day 12-14**

- Refactoring
- Debugging
- Bundling
- Documenting
- Pruning
- Testing deployment
- Testing users
- Testing mobile
- Final Demo Video -> Upload

---

## **Brainstorm Features**

- Similar: Dress me app
- Use Mannequin.js to model mockups
- Color selection from picture such as feature on colors.co