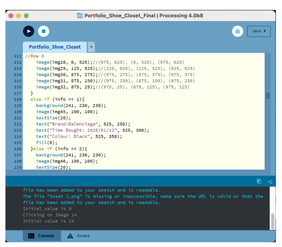
HSCI_234_1



Concept Description —
Virtual Shoe Closet

For this project, I am developing a Virtual Shoe Closet, inspired by a client project I worked on back in Grade 11 computer science. The original brief asked me to design a system that could help a client organize and manage their growing sneaker collection. For this final assignment, I'm reimagining that idea through an interaction-focused, p5.js-based experience.

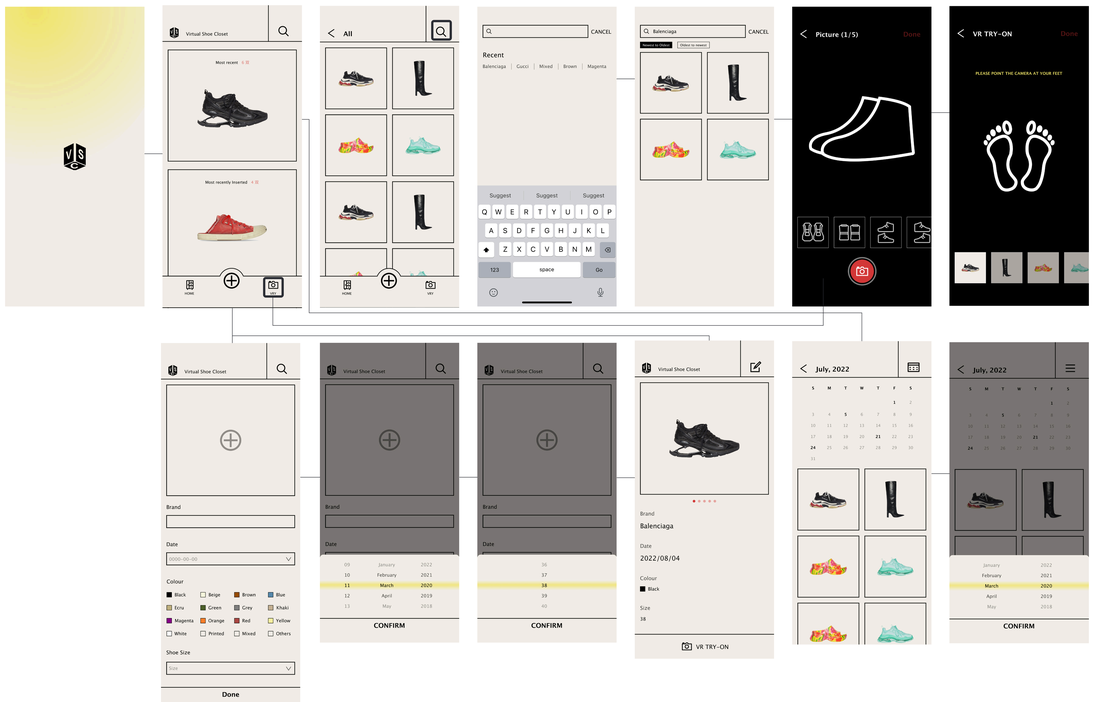
Client Profile

My client is someone who owns many pairs of shoes and struggles to keep track of what she has. She often forgets when she bought certain pairs, whether she already owns something similar, or where a specific pair is stored. She wanted a digital system that feels simple, visual, and personal, something that behaves like a virtual closet she can interact with.

Focus

For this course, I will simplify and reinterpret these features into a p5.js interaction prototype that focuses on:

- a clean screen-based interface,
- user-triggered interactions (click, input, hover),
- visual feedback for each action,
- a functional but minimal data system stored in arrays or objects.



Client Needs

After multiple conversations with her, I identified four core needs:

1. **Add New Shoes**
   She should be able to input a new pair of shoes into her virtual closet, including name, type, colour, and purchase date.
2. **View Closet Inventory**
   She wants a quick way to see all the shoes she currently owns, either as a list or through simple visuals.
3. **Search for a Specific Shoe**
   She should be able to type in a keyword (e.g., "black boots," "Nike," etc.) and immediately check if she already owns it.
4. **Track Purchase Dates**
   For budgeting and wardrobe planning, she wants to remember when each shoe was bought.

HSCI_234_2

Core features:

1. Add New Shoes (Form Input) – **Must-have**
   - User can enter shoe name, type, colour, and purchase date.
   - Upload / select a thumbnail image for that pair.
   - Validation for empty fields (cannot add a blank shoe).
2. View Closet Inventory – **Must-have**
   - All shoes are displayed as cards in a grid/list.
   - Each card shows image, name, type, colour, and date purchased.
   - Basic sorting (e.g., newest first).
3. Search / Filter – **Must-have**
   - User can type a keyword (brand, colour, type) to filter the visible shoes.
   - Real-time feedback: as the user types, the list updates.
4. View Shoe Detail – **Nice-to-have**
   - Clicking a card reveals a detail view with larger image and notes (ex: "rainy day boots").
   - Simple back button to return to the closet view.
5. Responsive Layout / Small Screen Support – **Nice-to-have**
   - On larger screens: multi-column layout.
   - On smaller screens: stack cards in a single column and simplify the header.
   - Implemented through CSS **flex-box** + media queries.

Features I'm intentionally simplifying or cutting for this class project:

- No login / multi-user system.
- No real database or long-term storage (data lives in the browser session only).
- No advanced analytics, recommendations, or "auto-suggest" features.

Implementation Strategy

Problems to solve

Regardless of which route I go, I need to:

- Represent shoes as data
  Use an array of objects (name, type, colour, date, imagePath) to store the closet contents.
- Connect UI to data
  When the user submits the form, a new shoe object is pushed into the array and the UI re-renders.
- Filter/search the data
  On every change in the search field, filter the array and only display matching shoes.
- Give clear feedback
  Error states (missing fields), empty search results ("No shoes found"), hover/tap states for interactive elements.

Plan A – p5.js Sketch (Single Canvas App)

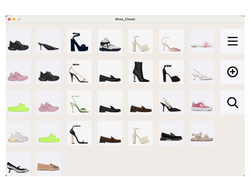
Build the entire thing inside a p5.js file. The file becomes the interface where cards, buttons, and text are shown.

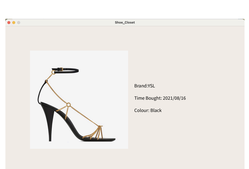
How it would work technically

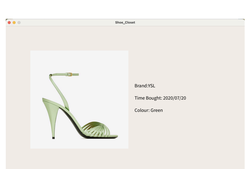
- Use p5.js for:
  - Drawing shoe cards (rectangles + text + possibly images via loadImage() / image()).
  - Handling input events (mousePressed, mouseMoved, and text input via createInput() elements).
  - Animations and small transitions when a new shoe appears or when filters change.
- Use arrays & objects in plain JS to store and update the shoe data.
- Use p5.js layout logic to fake "rows" and "columns" for the grid.

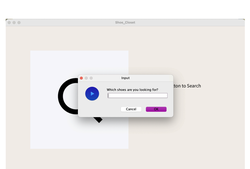
Problems / challenges

- I'd have to manually manage layout math (card widths, wrapping, scrolling if there are many shoes).
- Accessibility and responsiveness are harder because everything is drawn on a there.
- Recreating standard UI patterns (form fields, scrollable lists) is more work.









Plan B – HTML/CSS/JS Website

Turn the virtual closet into a small web app: HTML structure, CSS for layout and responsiveness, and JavaScript controlling the closet logic. P5.JS can still be used, but the main UI is standard HTML.

How it would work technically

- HTML
  - Structure: header (logo + "Virtual Shoe Closet"), main area with:
    - Add Shoe form
    - Search bar
    - Closet grid section (cards)
- CSS
  - Use grid to lay out shoe cards in multiple columns on larger screens.
  - Use CSS media queries to switch layout at certain breakpoints
  - This approach follows the responsive media query pattern from W3Schools, where styles change based on viewport width using the @media rule. W3Schools+3W3Schools+3W3Schools+3
- JavaScript
  - Handle form submission, create shoe objects, push to an array, and re-render the grid.
  - Handle search input and filter the array.
  -

Problems / challenges

- I need to manage more files (HTML, CSS, JS, and image assets).
- Setting up responsive breakpoints and testing across screen sizes.

Interaction Specification

1. Home / Closet View
   - Input: User lands on page.
   - Interaction:
     - Scroll through shoe cards.
     - Hover / tap on a card to see elevation / highlight.
     - Click card → opens shoe detail (if implemented).
   - Feedback: Smooth hover state, card shadow, maybe a mild scaling animation.

2. Add Shoe Form
   - Input: User fills in fields and clicks "Add Shoe".
   - System behaviour:
     - Validate required fields.
     - If valid, create a new shoe object, push to closet array, clear the form, and scroll to the new card.
     - If invalid, show inline error messages.
   - Feedback: Success message ("New pair added to your closet") and visual highlight on the new card.

3. Search / Filter
   - Input: User types in the search bar (brand / colour / type keywords).
   - System behaviour:
     - On every keystroke, filter the closet array and redraw only matching cards.
   - Feedback:
     - Instant update of visible cards.
     - Show a "No shoes found" state when nothing matches.

4. Shoe Detail (optional screen)
   - Input: User clicks a shoe card.
   - System behaviour:
     - Overlay or side panel opens showing larger image and full details.
   - Feedback:
     - Dimmed background, clear close button, simple fade-in animation.

5. Responsive Behaviour
   - Desktop: Multi-column grid of cards, form and search visible side-by-side (if space allows).
   - Tablet / Mobile:
     - Single-column list of cards.
     - Form stacked above the list.
     - Navigation simplified into a single header row.

Also refer to old codes for reference and remake the framework