DevFolio - My Portfolio Website

CSC 363 Human-Computer Interaction Final Project

Student: Steven M Cain Jr.
Date: December 8, 2025

What I Built

I created a full-stack portfolio website called DevFolio where I can showcase my projects, share information about myself, and let people contact me. The site has a public side that anyone can view and an admin dashboard where I can manage everything through a web interface instead of having to manually edit code.

Getting Started - How to Run My Project

Backend Server

1. Open terminal and navigate to backend folder:

   ```
   cd devfolio-backend
   ```

2. Install dependencies (first time only):

   ```
   npm install
   ```

3. Start the server:

   ```
   node index.js
   ```

4. You should see: `DevFolio server listening on port 8080`

Frontend Development Server

1. Open a second terminal and navigate to frontend folder:

   ```
   cd devfolio-frontend
   ```

2. Install dependencies (first time only):

   ```
   npm install
   ```

3. Start the dev server:

   ```
   npm run dev
   ```

4. Open your browser to `http://localhost:5173` (or whatever port it shows)

Database Setup

1. I'm using MySQL Workbench with a database called `devfolio`
2. All my database tables are listed in the "Database Structure" section below
3. To create the tables, just run the SQL code shown for each table

---

My Tech Stack

Frontend (What Users See): - React 19.2.0 - for building the user interface - Redux Toolkit - for managing app state (logged in admin, projects, profile data) - React Router v7 - for navigation between pages - Tailwind CSS - for styling everything - Vite - development server with fast hot reload

Backend (Server Side): - Node.js with Express - handles API requests - MySQL - stores all my data - bcrypt - securely hashes admin password - CORS - allows frontend to talk to backend

---

Database Structure

I designed my database to store everything I need for the portfolio. Here's what each table does:

admin

Stores my login credentials so I can access the admin dashboard.

```sql
CREATE TABLE admin (
  id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(255) NOT NULL UNIQUE,
  password_hash VARCHAR(255) NOT NULL
);
```

projects

The main table for all my projects - both digital (websites, apps) and physical (engine rebuilds, etc).

```sql
CREATE TABLE projects (
  id INT AUTO_INCREMENT PRIMARY KEY,
  title VARCHAR(255) NOT NULL,
  short_description TEXT,
  long_description TEXT,
  github_url VARCHAR(255),
  live_url VARCHAR(255),
  tags TEXT,
  type TINYINT DEFAULT 0,                -- 0 = physical, 1 = digital
  is_published TINYINT DEFAULT 0,        -- 0 = draft, 1 = published
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

project_images

Each project can have multiple images. I can add them via URL or by uploading files.

```
CREATE TABLE project_images (
  id INT AUTO_INCREMENT PRIMARY KEY,
  project_id INT NOT NULL,
  image_path VARCHAR(500) NOT NULL,
  caption TEXT,
  sort_order INT DEFAULT 0,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (project_id) REFERENCES projects(id) ON DELETE CASCADE
);
```

contacts

When someone fills out the contact form, their message gets saved here.

```
CREATE TABLE contacts (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL,
  subject VARCHAR(500) NOT NULL,
  message TEXT NOT NULL,
  project_details TEXT,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

profile

Basic info about me that shows on the About page.

```
CREATE TABLE profile (
  id INT PRIMARY KEY DEFAULT 1,
  full_name VARCHAR(255),
  bio TEXT,
  philosophy TEXT,
  photo_url VARCHAR(500),
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

skills

My technical skills organized by category (programming, tools, professional).

```sql
CREATE TABLE skills (
  id INT AUTO_INCREMENT PRIMARY KEY,
  category VARCHAR(50) NOT NULL,       -- 'programming', 'tools', or 'professional'
  skill_name VARCHAR(255) NOT NULL,
  sort_order INT DEFAULT 0
);
```

experience

My work experience entries for the About page.

```sql
CREATE TABLE experience (
  id INT AUTO_INCREMENT PRIMARY KEY,
  title VARCHAR(255) NOT NULL,
  company VARCHAR(255),
  period VARCHAR(100),
  description TEXT,
  sort_order INT DEFAULT 0
);
```

education

My education history.

```sql
CREATE TABLE education (
  id INT AUTO_INCREMENT PRIMARY KEY,
  school VARCHAR(255) NOT NULL,
  degree VARCHAR(255),
  period VARCHAR(100),
  coursework TEXT,
  sort_order INT DEFAULT 0
);
```

interests

A single text field for my interests/hobbies section.

```sql
CREATE TABLE interests (
  id INT PRIMARY KEY DEFAULT 1,
  content TEXT
);
```

API Endpoints - How Frontend Talks to Backend

I organized my API into logical sections. All routes start with `/api/`.

Projects

- `GET /api/projects` - Gets all projects with their first image
- `POST /api/projects` - Creates a new project
- `PUT /api/projects/:id` - Updates an existing project
- `PUT /api/projects/:id/publish` - Toggles published status (draft/published)
- `DELETE /api/projects/:id` - Deletes project and all its images

Project Images

- `GET /api/projects/:projectId/images` - Gets all images for a specific project
- `POST /api/projects/:projectId/images` - Adds image to project (by URL)
- `DELETE /api/project-images/:imageId` - Deletes a specific image

Admin Authentication

- `POST /api/admin/signin` - Login with username/password

Contact Form

- `POST /api/contacts` - Saves contact form submission

Profile (About Page Data)

- `GET /api/profile` - Gets all about page content (profile, skills, experience, education, interests)
- `PUT /api/profile` - Updates profile information
- `POST /api/skills` - Adds a new skill
- `DELETE /api/skills/:id` - Removes a skill
- `POST /api/experience` - Adds work experience
- `PUT /api/experience/:id` - Updates work experience
- `DELETE /api/experience/:id` - Removes work experience
- `PUT /api/education/:id` - Updates education entry
- `PUT /api/interests` - Updates interests text

Frontend Routes - What URLs Do

Public Pages (Anyone Can View)

- `/` - Home page with intro and my photo
- `/projects` - Gallery of all published projects (drafts hidden)

- `/about` - About me page with bio, skills, experience, education
- `/contact` - Contact form

Private Pages (Admin Only)

- `/admin/login` - Login page (has easter egg: "You name better be Steven if you're on this page…")
- `/admin` - Admin dashboard where I manage projects and images

Key Components I Built

Navigation

- **SideNav** - The cool sliding sidebar that appears on desktop (hidden on mobile)
- **MobileMenu** - Hamburger menu for mobile devices
- **ProtectedRoute** - Wraps admin routes, redirects to login if not authenticated

Project Management

- **ProjectForm** - Form to create/edit projects with all fields
- **ProjectList** - Shows all projects with edit/delete/publish buttons
- **ProjectCard** - Individual project card display
- **ProjectDetailModal** - Popup modal showing full project details

Image Management

- **ImageUploader** - Upload images by pasting a URL (file upload removed to simplify)
- **ImageList** - Displays project images with delete buttons

Other

- **SocialLinks** - Footer with my social media links

How Redux State Management Works

I use Redux Toolkit to manage app state. Here are my slices:

authSlice

Handles admin login state. - **State:** `admin` (logged in user), `status` (loading/idle/error) - **Actions:** - `signIn` - Login with credentials - `signOut` - Logout and clear state - `fetchMe` - Get current admin (not implemented on backend yet)

projectsSlice

Manages project data. - **State:** `items` (array of projects), `status` - **Actions:** - `fetchProjects` - Load all projects - `createProject` - Add new project - `updateProject` - Edit project - `deleteProject` - Remove project - `togglePublishProject` - Switch between draft/published

mediaSlice

Handles project images. - **State:** `imagesByProject` (object mapping projectId to array of images), `status` - **Actions:** - `fetchImages` - Load images for a project - `uploadImage` - Add image by URL - `deleteImage` - Remove image

profileSlice

Manages About page data. - **State:** `profile`, `skills`, `experience`, `education`, `interests`, `status` - **Actions:** - `fetchProfile` - Load all about page data from database

---

Project Structure

```
devfolio/
  devfolio-backend/
      index.js                  # Main server file with all API routes
      package.json              # Backend dependencies
      uploads/                  # Folder for project images
          me.png                # My profile photo
          12Abucket.jpg        # Project images
          ...

  devfolio-frontend/
      src/
          main.jsx              # App entry point
          App.jsx               # Route configuration
          index.css             # Global styles + custom animations

          api/
              apiClient.js     # Axios setup for API calls

          pages/
              HomePage.jsx     # Landing page with animated photo
              ProjectsPage.jsx # Project gallery with filters
```

```
            AboutPage.jsx       # About me with database content
            ContactPage.jsx     # Contact form
            admin/
                AdminLogin.jsx  # Login page
                AdminDashboard.jsx  # Admin panel

        components/
            SideNav.jsx
            MobileMenu.jsx
            ProjectCard.jsx
            ProjectList.jsx
            ProjectForm.jsx
            ProjectDetailModal.jsx
            ImageUploader.jsx
            ImageList.jsx
            SocialLinks.jsx
            ProtectedRoute.jsx

        store/
            store.js            # Redux store setup
            slices/
                authSlice.js
                projectsSlice.js
                mediaSlice.js
                profileSlice.js

    package.json
    vite.config.js
    tailwind.config.js          # Custom colors (olive, beige, cream)
```

Challenges I Faced

1. Profile Data Not Loading from Database

What went wrong: My About page was showing blank even though I had data in the database.

The cause: My Redux profileSlice was storing the entire API response under `state.profile` instead of breaking it apart. The backend returns an object like:

```
{
  profile: { full_name: "...", bio: "..." },
  skills: { programming: [...], tools: [...] },
  experience: [...],
  education: [...],
  interests: "..."
}
```

But my slice was just doing `state.profile = action.payload`, so everything got nested too deep.

How I fixed it: Changed the `fetchProfile.fulfilled` reducer to destructure:

```
state.profile = action.payload.profile;
state.skills = action.payload.skills;
state.experience = action.payload.experience;
// etc...
```

Where I found help: Redux Toolkit documentation on handling API responses

---

2. CV/Resume Management - Over-Engineering

What went wrong: I initially designed a whole database system for my resume with tables for resume sections, admin upload forms, CRUD endpoints, etc.

The problem: I realized I was making it way too complicated. I just needed a simple link to my resume that I could update easily.

How I fixed it: Deleted all the resume database stuff and just added a hardcoded Google Drive link as a button on the About page. Now I can update my resume by just replacing the file in the Google Drive folder - the link stays the same, no code changes needed!

Link I used: `https://drive.google.com/drive/folders/1h4Rwm1hvixSMiHQ9ve3nKqxOWo3IPMw8`

Lesson learned: Don't over-engineer simple features. Sometimes the simple solution is better.

Where I found help: Talked through it and realized database storage was overkill for a single file reference.

---

3. Code Organization and Cleanup

What went wrong: My code had a bunch of messy stuff left over from development - verbose comments, console.log() debug statements everywhere, inconsistent variable naming.

Examples of issues: - Using `(s)` vs `(state)` in Redux selectors - Multi-line comment blocks like `// ==================\n// PROJECTS\n// ==================` - Debug logs like `console.log("ProtectedRoute -`

`admin:", admin);` - Old files that weren't used anymore (AuthContext.jsx, hashPassword.js)

How I fixed it: - Deleted obsolete files (AuthContext was replaced by Redux, hashPassword was one-time use) - Simplified all backend comments to single-line style - Removed debug console.logs from components - Standardized naming conventions throughout

Where I found help: Just went through file by file and cleaned things up

---

4. Draft Projects Showing on Public Page

What went wrong: All my projects were showing on the `/projects` page even when I marked them as drafts.

How I fixed it: Added a filter to only show published projects:

```
const filteredProjects = projects.filter((p) => {
  if (!p.is_published) return false;  // Hide drafts
  // rest of filtering logic...
});
```

Now drafts only show in the admin dashboard!

What I Liked About This Project

Redux makes state management so clean - Once I understood the pattern, it was easy to add new features

Tailwind CSS is really fast - I could style things quickly with utility classes instead of writing CSS files

Component-based architecture - Being able to reuse components saved tons of time

Admin dashboard is super useful - I can update my portfolio without touching code

Vite is blazing fast - Hot reload happens instantly, way better than Create React App

MySQL is reliable - Having a real database means my data persists and I can query it easily

---

What I Didn't Like / Found Difficult

Initial Redux setup was confusing - Lots of boilerplate and concepts (thunks, slices, extraReducers) to learn

Managing image paths - Dealing with relative vs absolute URLs and making sure images display from the backend was tricky

CORS configuration - Had to figure out how to allow credentials and multiple ports for dev

Debugging Redux state - When something wasn't working, it was hard to tell if the problem was in the component, the slice, or the API

No database migrations - Had to manually run SQL commands to create tables, would be nice to have automated migrations

Type safety - Without TypeScript, I made mistakes with property names that would've been caught at compile time

---

Cool Features I Added

Animated Profile Photo on Home Page

I added my LinkedIn profile photo that pops in with a bouncy animation next to "I'M STEVEN!" using CSS animations:

```css
@keyframes popIn {
  0% { opacity: 0; transform: scale(0); }
  100% { opacity: 1; transform: scale(1); }
}
```

Easter Egg on Login Page

If someone accidentally lands on `/admin/login`, they see: *"You name better be Steven if you're on this page…"*

Triple-Click Secret on Home Page

If you click "I'M STEVEN!" three times fast, it takes you to the admin login page. Shows a click counter (1/3, 2/3, 3/3).

Dynamic About Page

Everything on the About page loads from the database - I can update my bio, skills, experience in MySQL Workbench and it shows up immediately.

Company Logo Integration

I added the Great West Casualty Company logo next to my internship entry on the About page.

Background Image on About Page

Used my LinkedIn banner image as a subtle background with opacity on the About page header.

Project Type Filtering

On the projects page, I can filter between "All", "Digital", and "Physical" projects.

Draft/Published System

Projects can be drafts (only visible in admin) or published (shown to everyone).

Future Improvements I'd Like to Make

If I had more time, here's what I'd add:

- Implement file upload for images instead of just URL input
- Add `GET /api/admin/me` endpoint for persistent sessions
- Create admin UI to edit About page content (instead of using MySQL Workbench)
- Add JWT tokens for better authentication security
- Email functionality so contact form submissions email me
- Image optimization/compression when uploading
- Search functionality for projects
- Analytics to see how many people visit
- Dark/light mode toggle
- More animations and transitions

How to Use the Admin Dashboard

1. Go to `http://localhost:5173/admin/login`
2. Enter username and password (stored in `admin` table)
3. Once logged in, you can:
    - **Create projects:** Fill out the form and click "Create Project"
    - **Edit projects:** Click "Edit" on any project
    - **Delete projects:** Click "Delete" (asks for confirmation)
    - **Toggle publish status:** Click "Unpublished" to make it public or "Published" to make it a draft

- **Manage images:** Click "Manage Images" to add/delete images for a project
- **Add images:** Paste image URL and click "Add Image"

Testing My Site

What I tested:

- All public pages load without errors
- Contact form submits and saves to database
- Admin login works with correct credentials
- Admin login rejects wrong password
- Can create/edit/delete projects
- Can add/delete project images
- Draft projects don't show on public page
- Published projects appear on public page
- About page loads data from database
- Navigation works on mobile and desktop
- Side nav only shows on large screens
- Mobile menu only shows on small screens

Conclusion

This was a challenging but rewarding project! I learned a ton about full-stack development, database design, state management, and building a real-world application. The hardest parts were getting Redux set up correctly and managing the relationship between frontend and backend, but once I got past those hurdles, adding new features became much easier.

I'm proud of how the site turned out - it's functional, looks professional, and I can easily maintain it through the admin dashboard. The best part is that I can keep adding projects and updating my information without having to touch any code!

Resources I Used

- **React Documentation** - https://react.dev/
- **Redux Toolkit Docs** - https://redux-toolkit.js.org/
- **Tailwind CSS Docs** - https://tailwindcss.com/docs
- **Express.js Guide** - https://expressjs.com/
- **MySQL Documentation** - https://dev.mysql.com/doc/
- **Stack Overflow** - For debugging specific errors

- **GitHub Copilot -** Helped with code suggestions and debugging

---

This documentation was created as part of my CSC 363 Human-Computer Interaction final project at Wayne State College.