Tab 1

# StudentChoice Tech Handover

## Overview

Students Choice is a React-based web platform for university students to share reviews, blogs, Q&As, and helpful links about their university experiences. The frontend is built for scalability, maintainability, and a modern user experience, leveraging Redux, RTK Query, Tailwind CSS, and a modular component structure.

## Tech stack

* **React 18:** Functional components, hooks, and React Router v6 for SPA navigation.
* **Redux Toolkit:** State management, including slices for authentication, UI, and data fetching.
* **RTK Query:** Data fetching and caching for API endpoints.
* **Tailwind CSS:** Utility-first CSS framework for rapid UI development and dark mode support.
* **Framer Motion:** Animations and transitions.
* **Jest:** Unit testing.
* **Playwright:** End-to-end testing.
* **Other:** color thief for color extraction, react-icons, etc.

## Project structure



## Routing

* Uses React Router v6.
* Main routes are defined in index.js.
* Dynamic routes for universities, programs, and subjects (e.g., /universities/:idUniversity/program/:idDetail).

## State management

* Redux Toolkit for global state (auth, UI, etc.).
* RTK Query for API calls (see service).
  + Each API (users, universities, fields, etc.) is defined as a slice.
  + Example: useGetUserDetailsQuery() fetches user profile data.
* UI State: Modal visibility, selected tabs, etc., managed via Redux UI slice.

## Styling

* Tailwind CSS: All styling is utility-based. Custom classes are defined in input.css.
* Dark Mode: Supported via Tailwind’s dark mode utilities.
* Responsive Design: Layouts adapt to mobile and desktop.

## Components

* Highly Modular: Components are split by function (e.g., InterestCircle, ReviewCard, ActionButton).
* InteractionArea: Central component for displaying reviews, blogs, Q&As, and helpful links for a given entity.
* Profile Components: For user profile, editing, and stats.
* Compare Components: For comparing universities, programs, and subjects.

## API integration

* Backend: Strapi-based REST API.
* Endpoints: Defined in RTK Query services (e.g., usersAPI, universityPagesAPI).
* Authentication: JWT stored in localStorage, managed via Redux.

## Testing

* Jest: For unit tests (see jestTests).
* Playwright: For E2E tests (see playwrightTests and playwright.config.js).

## Build & Deployment

* Build: npm run build (runs Tailwind and React build).
* CI/CD: GitHub Actions and GitLab CI configured for build, test, and deploy.
* Static Output: Built files are moved to the deployment directory.

## Key files & Entry points

* index.js: App entry, routing, Redux provider.
* store.js: Redux store and middleware setup.
* service: All API endpoints.
* InteractionArea.js: Main logic for displaying and filtering posts.
* Profile: User profile logic.

## Onboarding & Maintenance Tips

* Start with index.js to understand routing and app structure.
* Review API slices in service to see how data is fetched and cached.
* Use Tailwind for all new styling; avoid custom CSS unless necessary.
* Follow component conventions: Keep components small and focused.
* Check Redux slices for UI state changes (modals, tabs, etc.).
* Test new features with Jest and Playwright before merging.
* Refer to README for team info and project background.

## Useful commands

* npm start – Start dev server
* npm run build – Build for production (includes Tailwind)
* npm test – Run unit tests

## Further Reading

* [Redux Toolkit Docs](https://redux-toolkit.js.org/)
* [RTK Query Docs](https://redux-toolkit.js.org/rtk-query/overview)
* [Tailwind CSS Docs](https://tailwindcss.com/docs/installation/using-vite)
* [React Router Docs](https://reactrouter.com/home)