Commute Buddy – System Design Document – Sprint 2

This document captures the actual behaviour of the current code base under `setup/backend` and `setup/frontend`. Every section references the source files that implement the described behaviour so future reviewers can trace requirements back to code.

Product Overview

Commute Buddy is a full-stack web application that lets York University students find commute partners with overlapping transit routes. The MVP currently provides:

- Account creation restricted to `@yorku.ca/@my.yorku.ca` (Signup.jsx:48-104) and JWT-based login with role awareness (Login.jsx:11-78, backend/controllers/userController.js:24-79).
- Profile management with commute preferences, routes, and optional interests (ProfilePage.jsx:5-400, userController.js:114-213).
- A matching screen with route/time filters (Matches.jsx, FilterBar.jsx, MatchCard.jsx) powered by a server-side scoring algorithm (matchCalculator.js, commuteController.js:63-188).
- Commute request workflow (CommuteRequests.jsx, SendRequestModal.jsx, backend/controllers/commuteController.js).
- An admin view that displays all registered users (UserTable.jsx) behind a role guard (ProtectedRoute.jsx:3-33 and `/admin` route in App.js:30-49).

Technology Stack

Layer	Technologies	Location
Frontend	React 19 + React Router 7, axios, Tailwind utility classes (CSS modules)	`setup/frontend`
Backend	Node.js (ES modules), Express 5, Mongoose 8, bcryptjs, jsonwebtoken, cors, dotenv	`setup/backend`
Data	MongoDB (Local/Atlas). TTL index for expiring commute requests	`models/User.js`, `models/CommuteRequest.js`

Testing libs (React Testing Library, Jest DOM) are present but no tests currently exist.

Backend Components (setup/backend)

1. Application Setup

- `server.js` creates the Express app, plugs in `cors` and `express.json()`, connects to Mongo via `mongoose.connect(process.env.MONGO_URI)`, and mounts route modules. Port defaults to 5001 to avoid macOS conflicts (server.js:31-38).
- Authentication is enforced either via `protect` (loads `req.user` from DB) or `verifyToken` (decodes JWT payload only) plus optional `isAdmin` (middleware/authMiddleware.js:1-43).

2. Data Models

User (`models/User.js`)

- Required fields: `name`, `email`, `password`, `role` (defaults to `user`).
- Matching metadata: `preferredRoutes[]`, `startArea`, `transportMode`, `profileImage`, `gender`, `interests[]`, `commuteWindow`.
- Index: `{ preferredRoutes: 1 }` speeds up `$in` queries in the match finder.

CommuteRequest (`models/CommuteRequest.js`)

- Links `sender` and `receiver` (ObjectId refs), status state machine (`pending/accepted/declined/expired`), free-text `message`.
- Has `expiresAt` defaulted to 1h and TTL index to auto delete stale pending requests.

3. REST APIs

Endpoint	Method	Auth	Controller	Purpose
`/api/users/register`	POST	none	`registerUser`	Creates user, hashes password, returns JWT for immediate use.
`/api/users/login`	POST	none	`loginUser`	Validates password, issues JWT containing `id` and `role`.
`/api/users/content`	GET	`protect`	`getContent`	Sample protected message (legacy).
`/api/users/all`	GET	`verifyToken` + `isAdmin`	`getAllUsers`	Admin list of users (password excluded).

`/api/users/routes`	GET	`verifyToken`	`getAllRoutes`	Returns a static list of TTC route strings (future DB).
`/api/users/preferences`	GET	`verifyToken`	`getUserRoutes`	Reads caller's `preferredRoutes`.
`/api/users/preferences`	PUT	`verifyToken`	`updateUserRoutes`	Replaces caller's `preferredRoutes` with provided `routes[]`.
`/api/users/profile`	GET	`verifyToken`	`getUserProfile`	Fetches caller profile excluding password.
`/api/users/profile`	PUT	`verifyToken`	`updateUserProfile`	Upserts profile fields with normalization and uniqueness check on email.
`/api/users/changePassword`	PUT	`verifyToken`	`changeUserPassword`	Verifies current password and saves new Bcrypt hash.
`/api/commute/send`	POST	`protect`	`sendRequest`	Creates request by receiver name or email, prevents duplicates/self.
`/api/commute/respond`	POST	`protect`	`respondRequest`	Accepts/declines pending request (receiver only).
`/api/commute/my-requests`	GET	`protect`	`getUserRequests`	Returns both sent and received requests with `type` tag for UI.
`/api/commute/matches`	GET	`protect`	`findMatches`	Computes list of compatible users using filters and scoring.

`/api/content`	GET	`protect`	`contentController.getContent`	Currently returns greeting text.

4. Matching Algorithm (`controllers/commuteController.js` + `utils/matchCalculator.js`)

1. Load current user with `preferredRoutes`. If empty, respond with `userHasRoutes=false` so the UI can redirect to profile setup.
2. Build an exclusion set containing the current user and anyone with pending/accepted requests (Commuterequest query) so duplicates are avoided.
3. Build `matchQuery`:
   - `_id` not in exclusion set.
   - `preferredRoutes` contains either a user-selected `route` query param or overlaps with the caller's preferences.
   - Optional filters for `transportMode`, `startArea`, `commuteWindow` (Matches.jsx sets query params).
4. Hydrate candidate users and compute `sharedRoutes` + `commonRoutePercentage = round(shared / max(lenA,lenB) * 100)` (matchCalculator.js:7-25).
5. Filter by `minPercentage`, then sort by percentage desc, shared route count desc, and whether transport modes match.
6. Return `matches[]` with high-level profile data and the aggregated stats.

5. Commute Request Flow

- `SendRequestModal` (frontend) posts `{ receiver, message }` to `/api/commute/send`. The server resolves receiver by email or case-insensitive name match, prevents sending to self, and ensures only one pending request per sender/receiver pair.
- Requests expire automatically after 24 hours because of the TTL index; controller does not proactively prune them.
- `CommuteRequests.jsx` fetches `GET /api/commute/my-requests`, displays "received" vs "sent" tabs, and lets receivers act via `/api/commute/respond`.

6. Security & Validation

- Password hashing (bcrypt with salt rounds=10) during register and password change.
- JWT tokens expire in 1 hour on login (userController.js:51-58) and 24h when generated via `generateToken` for signup (utils/generateToken.js). Tokens are stored in `localStorage` on the client.
- Email uniqueness enforced on signup and profile update; `updateUserProfile` normalizes all fields to avoid empty strings.
- Currently missing features: HTTPS enforcement, refresh tokens, rate limiting, audit logging, and validation of route lists. These should be captured as future work.

Frontend Components (`setup/frontend/src`)

1. **Application Shell & Routing**
   - `App.js` wires up `BrowserRouter`, renders `Navbar` globally, and defines routes for `/home`, `/signup`, `/login`, `/matches`, `/content`, `/profile`, `/requests`, `/admin`, `/403`, and `/` redirect to Home. Protected screens are wrapped in `ProtectedRoute` which verifies `localStorage.token` and optional `requiredRole` before rendering children.
   - `Navbar.jsx` reads the stored user role to adjust menus, fetches `/api/users/profile` on render to show the profile avatar, and hides itself on the admin dashboard (lines 11-89). Logout simply clears `localStorage` and navigates to `/login`.

2. **Data Flow & State Management**
   - All pages use React hooks (`useState`, `useEffect`, `useMemo`) for local state; there is no global store. API calls use axios where convenient; some components use `fetch` directly.
   - JWT is read from `localStorage` before each request. Errors that return `401` trigger redirects to `/login` in `CommuteRequests.jsx` and `Matches.jsx`.

3. **Core Screens**

   Home (`pages/Home.jsx`)

   - Marketing landing page; CTA buttons navigate to `/matches`/`/profile` for logged-in users or `/signup`/`/login` otherwise.

   Signup (`pages/Signup.jsx`)

   - Plain form with local validation for York email and password confirmation. On success, shows alert and redirects to login.

   Login (`pages/Login.jsx`)

   - Submits credentials to `/api/users/login`. Stores token and user JSON, then routes either to `/admin` (role `admin`) or `/matches`.

   Matches (`pages/Matches.jsx`)

   - Loads available routes from `/api/users/routes` and user matches from `/api/commute/matches`.
   - Maintains two copies of filters: working `filters` bound to UI and `appliedFilters` mirrored into the query string parameters so "Show Matches" only re-fetches when the user clicks the button.
   - Shows `FilterBar` (component that emits filter events) and `MatchCard` entries. If backend returns `userHasRoutes=false`, prompts user to complete profile.

   Profile (`pages/ProfilePage.jsx`)

- Fetches profile details, all routes, and existing preferences on mount (three effects).
- Lets users select `preferredRoutes` via multi-select list, update profile metadata, manage interests, and change password.
- Enforces York email addresses client-side before hitting `/api/users/profile`.

Commute Requests (`pages/CommuteRequests.jsx`)

- Displays sent/received requests, provides inline actions for accept/decline, and a form to send new requests (mirrors `SendRequestModal` behaviour).

Admin (`pages/AdminPage.jsx`)

- Wraps `AdminDashboard` in `ProtectedRoute requiredRole="admin"`. AdminDashboard is currently a placeholder; admin tooling lives in reusable components (`UserTable`, `AdminNav`, `ReportsTable`, `ActivityLog`). `UserTable` fetches `/api/users/all` and renders a table.

Misc Pages

- `Content.jsx` is a placeholder for protected content.
- `Forbidden.jsx` shows a 403 page when a user lacks the required role.

4. **UI Components**
   - `ProtectedRoute.jsx` centralizes route guards by checking the token and verifying `requiredRole` before rendering.
   - `FilterBar.jsx` renders dropdowns for route, start area, commute window, and min percentage; it is stateless besides callback props.
   - `MatchCard.jsx` displays match metadata and launches `SendRequestModal` to send targeted requests.
   - `SendRequestModal.jsx` is a controlled overlay used both from MatchCard and the Requests page for message composition.
   - `AdminNav`, `UserTable`, `ReportsTable`, `ActivityLog` provide a foundation for future admin workflows.

5. **Authentication & Authorization Flow**
   1. **Signup** (`/api/users/register`): stores hashed password and immediately returns a 24h JWT (currently unused by frontend).
   2. **Login** (`/api/users/login`): returns `{ token, user }` with role. Frontend persists this data in `localStorage`.
   3. **Route Guards**: ProtectedRoute redirects unauthenticated users to `/login` and unauthorized roles to `/403`.
   4. **API Access**: Frontend attaches `Authorization: Bearer <token>` header to each axios/fetch call. Backend `protect` middleware verifies JWT signature via `JWT_SECRET` then loads the user document, while `verifyToken` just decodes payload for lightweight checks.
   5. **Admin Access**: `/api/users/all` uses `verifyToken` + `isAdmin` so tokens must include `role: "admin"`.

Security gaps to consider next:

- Tokens never refresh; user stays logged in until manual logout or token expiry (error handling for expired tokens should be improved).
- Tokens are kept in `localStorage`, making XSS a risk; consider HttpOnly cookies.
- Register/login lack rate limiting and email verification.
- No CSRF protection is required today because APIs depend on bearer tokens, but future cookie storage would need CSRF tokens.

6. Key User Flows

1. **Onboarding**
    - User lands on `/home`, clicks Sign Up → `/signup`.
    - Signup validates York email and matching passwords, hits `/api/users/register`, sees confirmation, then goes to `/login`.
2. **Profile Completion**
    - After login, user is redirected to `/matches` but a missing route list triggers the "Add routes" prompt, sending them to `/profile`.
    - Profile page fetches `/profile`, `/preferences`, `/routes`. User selects preferred routes and commute metadata, saves, then can view matches.
3. **Matching & Requests**
    - Matches page fetches `/api/commute/matches` with optional filters (route/startArea/commuteWindow/minPercentage). Each MatchCard can open a modal to send a request to the displayed user (receiver name auto populated and disabled to avoid tampering).
    - CommuteRequests page allows direct sending by typing receiver email/name, and handles responses for received requests.
4. **Admin Oversight**
    - Admin logs in, `Navbar` hides itself (Admin view will render a bespoke navigation later). `UserTable` fetches all users without passwords for monitoring.

---

7. Non-Functional Considerations & Future Work

Area	Current State	Recommended Improvements
Validation	Minimal server-side validation beyond email uniqueness	Use schema validators or express middleware (Joi/Zod) for field length, route membership, etc.
Testing	No automated tests run (`npm test` placeholder).	Add Jest/Supertest for controllers and React Testing Library for pages.

Observability	Console logs only.	Integrate Winston/Morgan logging and store request IDs.
Deployment	Local dev only.	Add scripts/docker-compose for multi-service dev, environment config for production domains.
Performance	Matching queries scan Users with `$in`.	Add compound indexes on `preferredRoutes`, `startArea`, `transportMode`. Cache route list.
Security	JWT in localStorage, no rate limiting or HTTPS mention.	Introduce Helmet, rate limiting, secure cookie storage, password policies, email verification.

8. File Map

Module	**Description**
`setup/backend/server.js`	Express app entry point, Mongo connection, route mounting, CORS rules.
`setup/backend/controllers/*.js`	Business logic for users, content placeholder, and commute workflow.
`setup/backend/routes/*.js`	Route definitions mapping HTTP endpoints to controllers.
`setup/backend/models/*.js`	Mongoose schemas for `User` and `CommuteRequest`.
`setup/backend/middleware/authMiddleware.js`	Auth helpers (`protect`, `verifyToken`, `isAdmin`).
`setup/backend/utils/generateToken.js`	Helper for JWT creation.
`setup/backend/utils/matchCalculator.js`	Shared route percentage calculation.
`setup/frontend/src/App.js`	Router shell and route protection wiring.
`setup/frontend/src/components/common/*`	Shared UI (Navbar, FilterBar, MatchCard, ProtectedRoute, SendRequestModal).
`setup/frontend/src/components/admin/*`	Admin nav + tables.

`setup/frontend/src/pages/*`	Page-level screens described in section 5.

This document reflects the code checked in on May 20, 2025. Update it whenever new endpoints, database fields, or user flows are introduced to keep architecture knowledge accurate.

System Architecture

```
┌──────────────────────────────────────────────────────────────────────┐
│                              Frontend (React 19)                      │
│                                                                       │
│ App.js: BrowserRouter + Navbar + Pages via <Routes>                   │
│                                                                       │
│   ├─ Auth pages: Signup, Login                                        │
│                                                                       │
│   ├─ Protected pages: Matches, Profile, CommuteRequests, Content      │
│                                                                       │
│   └─ Admin view: AdminPage → AdminDashboard (future)                  │
│                                                                       │
└────────────────▲─────────────────────────────────────────────────────┘
                 │ HTTPS (axios + fetch, JWT in Authorization header)
┌────────────────┴─────────────────────────────────────────────────────┐
│                       Backend (Node 20 + Express 5)                   │
│                                                                       │
│ server.js → express app + CORS + JSON + Mongo connection              │
│                                                                       │
│   ├─ /api/users       userRoutes.js → userController.js               │
│                                                                       │
│   ├─ /api/commute     commuteRoutes.js → commuteController.js         │
│                                                                       │
│   └─ /api/content     contentRoutes.js → contentController.js         │
│                                                                       │
│ Middleware: authMiddleware.js (JWT verification, role checks)         │
│                                                                       │
└────────────────┬─────────────────────────────────────────────────────┘
                 │ mongoose ODM
┌────────────────▼─────────────────────────────────────────────────────┐
│                            MongoDB Collections                        │
│                                                                       │
│  users: models/User.js (profile + route prefs + metadata)             │
│                                                                       │
│  commuterequests: models/CommuteRequest.js (status, TTL expiry)       │
│
```

```
└────────────────────────────────────────────────────────────────────
┘
```

Deployment assumptions:

- Backend expects `MONGO_URI` and `JWT_SECRET` in `.env` loaded by `dotenv` (server.js:4-15, generateToken.js, userController.js).
- CORS currently allows `http://localhost:3000` (server.js:18-28).
- The repo does not contain build/deployment manifests yet; local dev runs two npm projects under `setup/frontend` and `setup/backend`.
