Commute Buddy – System Design Document – Sprint 3

This document reflects the code in setup/backend and setup/frontend as of Sprint 3. It calls out what changed from Sprint 2 and the new updates that have been added.

SYSTEM DESIGN

Product Overview

Commute Buddy is a full-stack web app for York University students to find and coordinate with commute partners. Sprint 3 adds:

- Real-time notifications (requests, matches, messages) via Socket.io (server.js, App.js, Notifications.jsx).
- Persistent 1:1 chat with chat rooms and message history (models/ChatRoom.js, models/ChatMessagesTable.js, controllers/messagesController.js).
- Matching filters expanded with faculty/program/availability and richer profile data (models/User.js, commuteController.js, Matches.jsx).
- Soft session hardening: login rate limiting, idle logout, and silent token refresh while active (userRoutes.js, App.js).
- Admin dashboard scaffolding retained; user table still the primary functional admin tool (components/admin/*).

Technology Stack

- **Frontend:** React 19 + React Router 7, axios, Tailwind utility classes; Socket.io client (setup/frontend).
- **Backend:** Node.js (ESM), Express 5, Mongoose 8, Socket.io server, bcryptjs, jsonwebtoken, cors, dotenv, express-rate-limit (setup/backend).
- **Database:** MongoDB (local/Atlas) with TTL for commute requests and indexes for matching queries.

Backend Components (setup/backend)

Application Setup

- server.js wires Express, CORS, JSON parsing, Mongo connect, and mounts routes /api/users, /api/content, /api/commute, /api/messages, /api/auth.
- Socket.io server runs on the same HTTP server; tracks activeChats to suppress duplicate message notifications and supports rooms for users and chat rooms.
- Login/register routes are throttled to 5 requests/minute per IP via express-rate-limit (routes/userRoutes.js).

Data Models

- models/User.js: Core identity plus matching fields; Sprint 3 adds faculty, program, availabilityWindow, and seenMatches[] to avoid repeat match pings. Existing fields cover start area, transport mode, profile image, gender, interests, commute window. Index on preferredRoutes.
- models/CommuteRequest.js: Sender/receiver, status state machine, free-text message, 24h TTL via expiresAt + index.

- models/ChatRoom.js: Unique room per user pair (user1Id, user2Id sorted).
- models/ChatMessagesTable.js: Message text, sender, chatRoom ref, timestamp; indexed by chatRoomId.

REST & Realtime APIs

Endpoint	**Method**	**Auth**	**Controller**	**Sprint 3 Notes**
/api/users/register	POST	none	registerUser	Rate limited; returns JWT.
/api/users/login	POST	none	loginUser	Rate limited; issues 1h JWT with role.
/api/users/all	GET	verifyToken + isAdmin	getAllUsers	Admin-only table data.
/api/users/routes	GET	verifyToken	getAllRoutes	Static seed list.
/api/users/preferences	GET/PUT	verifyToken	getUserRoutes/updateUserRoutes	Manage preferredRoutes.
/api/users/profile	GET/PUT	verifyToken	getUserProfile/updateUserProfile	Profile + new faculty/program/availability fields; email uniqueness enforced.
/api/users/changePassword	PUT	verifyToken	changeUserPassword	Current-password check + bcrypt hash.
/api/commute/send	POST	protect	sendRequest	Prevents self/duplicate requests; now emits incoming-request Socket.io event.
/api/commute/respond	POST	protect	respondRequest	Accept/decline, creates chat room on accept, emits request-response.
/api/commute/my-requests	GET	protect	getUserRequests	Adds type flag for sent vs received.
/api/commute/matches	GET	protect	findMatches	Adds filters for faculty/program/availability; emits new-match for first-time pairings.
/api/messages/open-or-create/:friendId	POST	protect	openOrCreateChat	Stable room creation using sorted user IDs.
/api/messages/my-chats	GET	protect	getMyChats	Returns rooms with populated user info.

/api/messages/:chatRoomId/messages	GET	protect	getMessagesByChatId	Ordered message history.
/api/messages/:chatRoomId/send	POST	protect	sendMessage	Persists message, emits new-message to receiver with suppression if they're active.
/api/auth/me	GET	protect	inline	Returns minimal user profile for session restore.
/api/auth/refresh	GET	protect	inline	Issues new JWT cookie for active sessions.
/api/content	GET	protect	contentController.getContent	Legacy placeholder.

Matching Updates

- findMatches now filters by startArea, transportMode, commuteWindow, availabilityWindow, gender, faculty, program, and specific route query param. Excludes anyone with pending/accepted requests and tracks seenMatches to avoid repeat socket notifications.
- Sorting prioritizes common route percentage, then shared route count, then transport mode match bonus.

Commute Request Flow

- Requests expire automatically after 24h (TTL index). Send resolves receiver by email or exact-case-insensitive name, blocks self/duplicates, and notifies receiver in real time. Respond only allows the intended receiver; accepting auto-creates a chat room.

Messaging & Notifications

- Chat flows: /messages/open-or-create/:friendId -> /messages/:chatRoomId/send + Socket.io broadcast to room and receiver-specific notifications. activeChats map suppresses notifications when the receiver is already viewing the conversation.
- Global socket events used in the frontend: incoming-request, request-response, new-match, new-message.

Security & Validation

- JWT verification via protect; admin enforcement via verifyToken + isAdmin.
- Login/register rate limiting, idle logout after 1h of inactivity, and silent token refresh every 5 minutes while active (frontend) reduce stale sessions.
- Password hashing with bcrypt; email uniqueness and York-domain validation are enforced client-side and server-side uniqueness check.

Frontend Components (setup/frontend/src)

Application Shell & Routing

- App.js sets up routes for /home, /signup, /login, /matches, /profile, /requests, /messages, /messages/:chatRoomId, /content, /admin, /403. ProtectedRoute checks token + optional role; admin route requires role === "admin".
- Idle logout + silent token refresh implemented in App.js (hooks) to clear tokens after inactivity while refreshing active sessions.
- Socket listeners are centralized in SocketListeners (inside App.js) and feed the notification system.

Data Flow & State Management

- Local state via hooks; contexts/NotificationsContext.jsx provides a global notification store used by Notifications.jsx.
- API helpers in api/chatApi.jsx attach bearer tokens via axios interceptor.

Core Screens

- **Matches (pages/Matches.jsx + components/common/FilterBar.jsx, MatchCard.jsx):** Adds filters for faculty/program/availability; clears "new matches" flag on load; prompts users without routes to complete profile.
- **Commute Requests (pages/CommuteRequests.jsx + SendRequestModal.jsx):** Tabbed sent/received view, inline respond, request composer. Integrates with socket notifications.
- **Messages (pages/Messages.jsx, pages/ChatWindowPage.jsx + ChatList.jsx, ChatWindow.jsx, MessageBubble.jsx, MessageInput.jsx):** Lists chat rooms, tracks unread via socket new-message, joins rooms, and streams messages in real time.
- **Profile (pages/ProfilePage.jsx):** Adds faculty, program, availability, interests chips, profile image preview, and password change UX enhancements.
- **Admin (pages/AdminPage.jsx, components/admin/*):** AdminNav, UserTable (functional), ReportsTable/ActivityLog placeholders for future sprint.
- **Auth/Navigation:** Navbar shows avatar pulled from /api/users/profile and hides on admin view; Forbidden page for role failures.

Authentication Flow

- Login saves {token, user, userId} to localStorage; refresh endpoint keeps cookies alive for active users. Navigation guards via ProtectedRoute and server-side middleware.

Key User Flows

- **Match → Request → Chat:** User filters matches, sends commute requests (modal or Requests page). Receiver gets real-time alert, accepts, which auto-creates chat; both can then message with live delivery and unread badges.
- **Session Management:** Inactivity triggers auto-logout after 1 hour; active sessions refresh JWT silently every 5 minutes; socket reconnect restores room subscriptions after /api/auth/me succeeds.

- **Admin Oversight:** Admin signs in, views all users in UserTable; other admin tabs are placeholders for Sprint 4.

Non-Functional Considerations & Sprint 3 Improvements

Area	Sprint 2	Sprint 3 delta
Validation	Basic auth checks	Added login/register rate limiters; stricter email uniqueness on profile update.
Security	JWT in localStorage only	Idle logout + silent refresh, Socket.io rooms per user, notification suppression to reduce spam.
Matching performance	Route index only	Added exclusion set + seenMatches tracking; same index retained.
Observability	Console logs	Socket connect/disconnect logs; no centralized logging yet.
UX/Engagement	Request cards only	Real-time notifications, unread chat badges, and auto room creation on accept.

File Map Highlights

- Backend: server.js, routes/*, controllers/*, models/*, middleware/authMiddleware.js, utils/matchCalculator.js, utils/generateToken.js.
- Messaging: controllers/messagesController.js, routes/messageRoutes.js, models/ChatRoom.js, models/ChatMessagesTable.js.
- Frontend routing/shell: src/App.js, src/components/common/ProtectedRoute.jsx, src/components/common/Navbar.jsx.
- Notifications: src/contexts/NotificationsContext.jsx, src/components/common/Notifications.jsx.
- Chat UI: src/pages/Messages.jsx, src/pages/ChatWindowPage.jsx, src/components/common/ChatList.jsx, ChatItem.jsx, ChatWindow.jsx, MessageBubble.jsx, MessageInput.jsx.
- Matching/Requests: src/pages/Matches.jsx, components/common/FilterBar.jsx, MatchCard.jsx, SendRequestModal.jsx, pages/CommuteRequests.jsx.

DOCUMENTATION

Installation & Running (no change from Sprint 2, reiterating)

1. Clone the repo and install dependencies in setup/backend and setup/frontend via npm install.
2. Create setup/backend/.env from .env.example with MONGO_URI and JWT_SECRET. Backend runs on port 5001 (also hosts Socket.io).
3. Start backend: cd setup/backend && npm start.
4. Start frontend: cd setup/frontend && npm start (expects backend at http://localhost:5001).

New/Updated Endpoints (since Sprint 2)

Endpoint	Method	Auth	Purpose
/api/messages/open-or-create/:friendId	POST	Bearer	Create or fetch a 1:1 chat room between the logged-in user and friendId.
/api/messages/my-chats	GET	Bearer	List chat rooms for the logged-in user with participant info.
/api/messages/:chatRoomId/messages	GET	Bearer	Fetch ordered message history for a room.
/api/messages/:chatRoomId/send	POST	Bearer	Persist a message and notify the other participant.
/api/auth/me	GET	Bearer	Restore current session (used on socket reconnect).
/api/auth/refresh	GET	Bearer	Silent token refresh for active users.
/api/commute/matches	GET	Bearer	Now accepts faculty, program, and availabilityWindow filters in addition to existing route/time filters.

New/Updated Pages & Components

- pages/Messages.jsx, pages/ChatWindowPage.jsx with chat UI components (ChatList.jsx, ChatItem.jsx, ChatWindow.jsx, MessageBubble.jsx, MessageInput.jsx).
- contexts/NotificationsContext.jsx + components/common/Notifications.jsx for real-time toasts.
- App.js socket listeners, idle logout, and silent refresh hooks.
- ProfilePage.jsx adds faculty/program/availability inputs and interest chip UX; Matches.jsx consumes the new filters.

Updated Flows & Notes

- Accepting a commute request now creates a chat room and routes users into messaging.
- Notifications panel routes users to Requests/Matches/Messages depending on the event type.
- Rate limiting is active on register/login; repeated failed attempts may trigger the limiter response.

Other Changes

- Commute requests auto-expire after 24h via TTL index (cleanup is database-driven).
- Default avatars are applied consistently for notifications, matches, and chats when profile images are missing.
- Admin dashboard keeps only the User table functional; other tabs are scaffolds for Sprint 4.

Keep this document in sync with future changes (new admin tools, group chat, analytics, etc.).

System Design Architecture

```
┌──────────────────────────────────────────────────────────
──────┐
│                    Frontend (React 19)              │
├──────────────────────────────────────────────────────────
──────┤
│ App.js: BrowserRouter + Navbar + Pages via <Routes>      │
│                                               │
│ ├─ Auth pages: Signup, Login                          │
│                                               │
│ ├─ Protected pages: Matches, Profile, CommuteRequests,  │
│ │                 Messages, Content                 │
│                                               │
│ ├─ Chat system: ChatList + ChatWindow + Socket listeners │
│                                               │
│ ├─ Notifications: Socket-driven real-time alerts        │
│ │   -> incoming requests                           │
│ │   -> request responses                           │
│ │   -> new matches                               │
│ │   -> new messages                               │
│                                               │
│ ├─ Admin view: AdminDashboard (role-restricted)          │
│                                               │
│ State & Auth: token + user stored in localStorage, idle  │
│ logout (1 hr), silent refresh (5 min), role checks.      │
└──────────────────────────────────────────────────────────
──────┘

                   ▲
                   │  HTTPS (axios + fetch, JWT in header)
                   ▼

┌──────────────────────────────────────────────────────────
──────┐
│          Backend API (Node.js 20 + Express 5)          │
├──────────────────────────────────────────────────────────
──────┤
│ server.js: express app + CORS + JSON + Mongo connection  │
│                                               │
│ Routes:                                  │
│ ├─ /api/auth      authRoutes.js      → authController  │
│ ├─ /api/users     userRoutes.js      → userController  │
│ ├─ /api/commute   commuteRoutes.js   → commuteCtrl     │
│ ├─ /api/messages  messageRoutes.js   → messageCtrl     │
│ └─ /api/content   contentRoutes.js   → contentCtrl     │
│                                        │
│ Middleware:                               │
│ ├─ protect.js  (JWT verification, user attach)        │
│ ├─ isAdmin.js  (admin role guard)                     │
│ ├─ rateLimiter (register/login 5 req/min)             │
│ └─ validateYorkEmail, validateRoute, validateProfile  │
└──────────────────────────────────────────────────────
──────────┘

              ▲
              │  WebSocket (Socket.io)
              ▼

┌──────────────────────────────────────────────────────
──────────┐
│            Realtime Layer (Socket.io)            │
├──────────────────────────────────────────────────────
──────────┤
│ User connects → token verification → join(userId) room  │
│                                        │
│ Events (server → client):                  │
│ ├─ incoming-request                        │
│ ├─ request-response (accepted/declined)           │
│ ├─ new-match                               │
│ └─ new-message (chat updates)                  │
│                                        │
│ Events (client → server):                  │
│ ├─ send-message                            │
│ ├─ join-chat-room (chatRoomId)                 │
│ └─ typing indicators (future)                │
│                                        │
│ Manages: activeChats, chat rooms, direct user targeting  │
└──────────────────────────────────────────────────────
──────────┘
```

```
                ▲
                │  mongoose ODM
                ▼

┌──────────────────────────────────────────────────────────
──────┐
│                 MongoDB Atlas                           │
├──────────────────────────────────────────────────────────
──────┤
│ Collections / Models                                    │
│                                         │
│ users: models/User.js                                   │
│   → profile info (faculty, program, gender, interests)  │
│   → route data (start, end, time window)                │
│   → auth data (email, passwordHash, role)               │
│   → metadata (seenMatches, availabilityWindow)          │
│                                         │
│ commuteRequests: models/CommuteRequest.js               │
│   → requester, receiver                                 │
│   → status (pending/accepted/declined)                  │
│   → timestamps + TTL expiry                             │
│                                         │
│ messages: models/Message.js                             │
│ chatRooms: models/ChatRoom.js                           │
│   → sorted user pair, message refs, unread counts       │
│                                         │
│ content: models/Content.js                              │
│   → text blocks, announcements for admin pages          │
└──────────────────────────────────────────────────────────
──────┘
```