CRC CARD

AvailabilityCard

- Responsibilities:
- Render a single availability card
- Render all timeslots
- Trigger per-slot `Activate/Deactivate`
- Trigger card-level `Activate All/Deactivate All`
- Open Edit action

- Collaborators:
- `Availabilities`
- `useAvailability`
- Therapist availability APIs

EditAvailability

- Responsibilities:
- Load selected availability
- Edit date and timeslot data
- Submit PATCH updates

- Collaborators:
- `useAvailability`
- Availability update API

Appointments/AppointmentDetails

- Responsibilities:
- Display therapist appointment list
- Support filtering and search
- Update appointment status without full page reload
- Render patient-linked appointment details safely

- Collaborators:
- `AppointmentTable`
- Appointment APIs

AppointmentService

- Responsibilities:
- Query therapist appointments with populated patient data
- Enforce appointment status transition rules
- Apply slot reservation logic when therapist accepts appointment

- Collaborators:
- Appointment model
- `AvailabilityService`
- Appointment controllers

Availabilities

- Responsibilities:
- Fetch and display all therapist availabilities
- Render availability list
- Handle refresh
- Navigate to Create/Edit pages

- Collaborators:
- `AvailabilityCard`
- `useAvailability`
- Router (`TherapistDashboard` routes)

useAvailability

- Responsibilities:
- Encapsulate availability API calls
- Support full availability activate/deactivate
- Support per-timeslot status toggle
- Provide normalized data/actions to UI

- Collaborators:
- `Availabilities`
- `AvailabilityCard`
- `EditAvailability`
- Axios/API service layer

AvailabilityController

- Responsibilities:
- Expose therapist availability HTTP endpoints
- Validate request payloads
- Handle activate/deactivate (full and slot-level)
- Return standardized API responses

- Collaborators:
- `AvailabilityService`
- therapist.routes.js
- Authentication middleware

TherapistController

- Responsibilities:
- Provide therapist dashboard statistics
- Compute distinct patient count correctly
- Return aggregate dashboard data

- Collaborators:
- Appointment model
- Therapist dashboard frontend

Education

- Responsibilities:
- Manage patient education flow (`topic -> content -> detail`)
- Fetch topic content from backend API
- Filter by `article/video` ; render source links and detail view
- Render source links and detail view

+ Collaborators:
api client
getEducationContentByTopic
EducationContent

getEducationContentByTopic

- Responsibilities:
- Validate `topic` query against supported list
- Query published education content and sort results
- Return standardized JSON response and errors

+ Collaborators:
EducationContent
patient.routes
validateToken

EducationContent

- Responsibilities:
- Persist education content fields (topic, type, title, summary, duration, body, sourceName, sourceUrl, isPublished, order)

+ Collaborators:
getEducationContentByTopic
seed scripts

patient.routes

- Responsibilities:
- Expose patient education endpoint (GET /education/content)
- Route request to controller
- Ensure endpoint is behind authentication middleware

+ Collaborators:
validateToken
getEducationContentByTopic

validateToken

- Responsibilities:
- Authenticate incoming patient requests
- Reject unauthorized requests
- Attach validated user context for downstream handlers

+ Collaborators:
patient.routes
all protected controllers including education

EducationSeedScripts

- Responsibilities:
- Populate and refresh education topic content in MongoDB
- Maintain topic-specific datasets
- Support repeatable setup for Sprint demo/testing

+ Collaborators:
EducationContent
MongoDB connection/config

Patient

- height: Number = null
- weight: Number = null
- bloodType: String = null

+ save(): Promise<void>
controller: mongoose
+ comparePassword(password): Promise
(The rest is provided by Mongoose)

PatientsController (patients.controller.js)

(No instance attributes; exported function)

+ editPatientProfile(req: Request, res: Response):
= Promise<void>
Read height, weight, bloodType from req.body
Validate and write to patient, then patient.save()
controller: Patient

Topic

responsibilities:
- Represent an education topic/category (e.g., NCD Management)
- Provide a list of associated content items

collaborators:
- ContentItem
- EducationService

ContentDetails

responsibilities:
- Store and present full content (article text or video URL
- Store optional source/reference link
- Save a content item for a user
- Remove a saved content item for a user
- Share the content to a therapist

collaborators:
- ContentItem
- SavedMaterialsService
- User

ContentItem

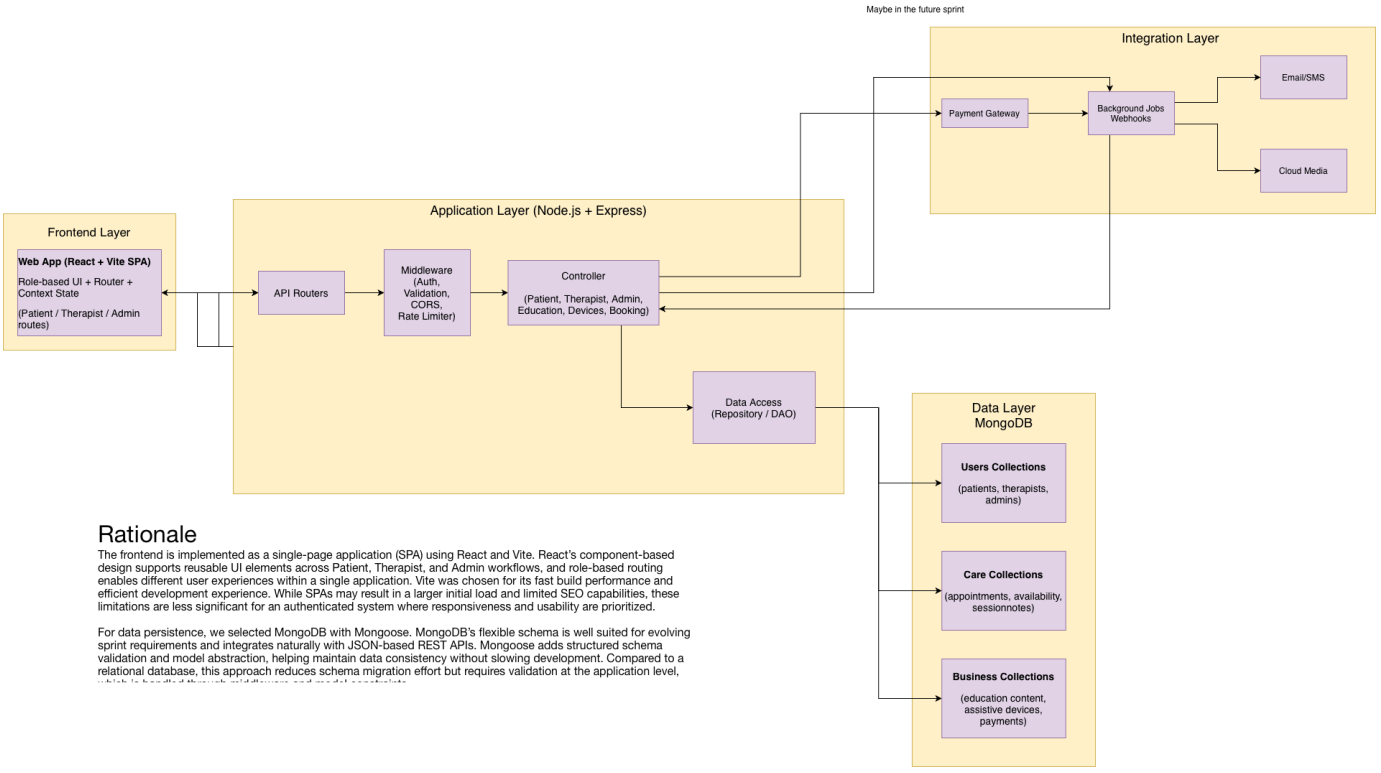
responsibilities:
- Store content metadata for listing/preview (title, summary, type, read time)
- Provide identifiers/links to retrieve full content details

collaborators:
- Topic
- EducationService
- SavedMaterialsService

SavedMaterialsService

responsibilities:
- Show the saved content item for a user
- Retrieve a user's saved materials list
- Provide saved materials count for a user

collaborators:
- User
- ContentItem
- SavedItem



Rationale

The frontend is implemented as a single-page application (SPA) using React and Vite. React's component-based design supports reusable UI elements across Patient, Therapist, and Admin workflows, and role-based routing enables different user experiences within a single application. Vite was chosen for its fast build performance and efficient development experience. While SPAs may result in a larger initial load and limited SEO capabilities, these limitations are less significant for an authenticated system where responsiveness and usability are prioritized.

For data persistence, we selected MongoDB with Mongoose. MongoDB's flexible schema is well suited for evolving sprint requirements and integrates naturally with JSON-based REST APIs. Mongoose adds structured schema validation and model abstraction, helping maintain data consistency without slowing development. Compared to a relational database, this approach reduces schema migration effort but requires validation at the application level,

System Interactions and Assumptions

System Architecture

The system follows a standard three-tier architecture:

- **Frontend (React + Vite)**
  Provides user interface for education content browsing and saved materials.

- **Backend (Node.js + Express)**
  Handles REST API requests, authentication, and business logic.

- **Database (MongoDB)**
  Stores users, education content, and saved materials.

External Services

- **JWT** for authentication

- **SendGrid** for email notifications

System Interactions

- The frontend communicates with the backend via REST APIs.

- The backend processes requests and interacts with MongoDB.

- The backend uses JWT for authentication.

System Assumptions

- MongoDB service is running before application start.

- Backend and frontend communicate via configured CORS.

- Users access the system through modern web browsers.

- HTTPS will be required for production deployment.