Business Use Case

Use Case

- Develop a solution for a Healthcare Wellness and Preventive Care Portal that seamlessly integrates frontend and backend technologies.
- The portal should be focused on wellness and preventive care to help patients achieve basic health goals and maintain compliance with preventive checkups.
- The portal should prioritize usability and key preventive care features.
- Special attention should be given to responsive design, authentication, profile management, and compliance with healthcare privacy standards.

Scope for 5-Hour MVP

- Given the time constraint, focus on delivering a Minimum Viable Product (MVP) that demonstrates key aspects of the problem statement, with emphasis on security, personalization, and healthcare compliance.

Mock Designs

Login Page

- Email
- Password
- Login Button
- "Forgot Password?"
- "New User? Register here"

Healthcare Portal

Navigation: Home, Health Topics, Services, Contact.

Sections:

- **Latest Health Information**
    - COVID-19 Updates
      Stay informed about the latest COVID-19 guidelines and vaccination information.
      [Read More]

- Seasonal Flu Prevention
  Learn steps to prevent seasonal flu and when to get vaccinated.
  [Read More]

- Mental Health Awareness
  Explore resources and support options for maintaining good mental health.
  [Read More]

Patient Dashboard Mockup

- Welcome, David
- **Wellness Goals**
  - Steps: 3620 / 6000 steps – 60 percent
  - Active Time: 56 minutes – 1712 kcal – 1.23 km
  - Sleep: 6 hr 30 min (graph shown)
- **Preventive Care Reminders**
  - Upcoming Annual blood test on 2nd Jan 2025
- **Health Tip of the Day**
  - Stay hydrated. Aim to drink at least 8 glasses of water per day.

Architecture & DevOps Considerations

MVP Deliverables

- Functional authentication system for patients and providers.
- Patient dashboard for tracking wellness goals and reminders.
- Basic goal tracker.
- Profile management for patients.
- Provider dashboard with patient compliance overview.
- Public health information page.
- Deployed frontend and backend applications.
- Functional CI/CD pipeline with basic automated tests.

Additional Considerations

- Prioritize security and privacy features over complex functionalities.

- Demonstrate understanding of healthcare data protection requirements.
- Be prepared to explain architectural decisions and their impact on security and performance.
- Focus on creating a coherent, if limited, user experience that showcases personalization.

Architecture & DevOps – Technical Details

- Design a scalable architecture separating frontend, backend, and data layers.
- Implement proper error handling and logging.
- Ensure secure data flow between frontend and backend.
- Set up a cloud-based NoSQL database instance.
- Configure environment variables for sensitive information.
- Implement basic HIPAA compliance measures such as data encryption and access controls.

Key Features to Implement

1. Secure Authentication System

- Login and registration for patients and healthcare providers.
- JWT-based session management with expiration.
- Password hashing and security measures.

2. Patient Dashboard

- Display wellness goal progress (steps, active hours, sleep).
- Preventive care reminders
  Example: “Upcoming: Annual blood test on [date]”.
- A simple “health tip of the day”.

3. Profile Management

- Allow patients to view and edit their profile information.
- Include fields for basic health information such as allergies and current medications.

4. Healthcare Provider View

- View assigned patients and their compliance status (e.g., “Goal Met” or “Missed Preventive Checkup”).
- Clickable patient list showing goals and compliance.

5. Public Health Information Page

- Static page with general health information and privacy policy.

6. Goal Tracker for Patients

- Allow patients to log daily goals such as steps or water intake.

7. Privacy & Security Measures

- Implement logging for user actions related to data access.
- Add a consent checkbox for data usage during registration.

Technical Requirements

1. Frontend

- ReactJS or NextJS.
- Basic styling with CSS modules or Sass.

2. Backend

- Node.js with Express or Python Django or a simple JSON server for API.
- NoSQL database of choice: MongoDB, DynamoDB, or Firestore.

3. Authentication

- JWT for secure sessions.
- Role-based access control (patient versus healthcare provider).

4. API

- RESTful API for communication between frontend and backend.

5. Cloud Deployment

- Deploy frontend and backend to cloud platforms of choice.
- Set up environment variables for configuration.

6. CI/CD

- Implement a basic GitHub Actions workflow for automated testing and deployment.

ARCHITECTURE

System overview

A three tiered web application:

- Client layer: Single page application built with React or NextJS that runs in user browsers and mobile web.
- Application layer: REST API server implemented in Nodejs with Express or Python with Django REST that enforces business logic and security.
- Data layer: NoSQL database such as MongoDB or Firestore for user profiles, goals and reminders. Optional object store for static assets.

High level components and responsibilities

- Client
  - Authentication UI and session handling
  - Dashboard components for steps, active time, sleep, reminders and tips
  - Forms for profile, goal, and reminder management
  - Provider view to inspect assigned patients and compliance status
- API server
  - Authentication and authorization
  - User profile management
  - Goal and reminder CRUD and aggregation
  - Provider patient listing and compliance calculations
  - Public content endpoints for health information
  - Audit logging and request logging
- Database

    - Collections for users, goals, reminders, audit logs, public content
    - Indexes on user id and timestamps for performance
- Background worker
    - Optional cron or queue job to send reminder notifications and to aggregate daily metrics
- Identity and secrets
    - JWT signing with a strong secret stored in environment variables
    - Role based access control to separate patient from provider capabilities
- Infrastructure
    - Managed hosting for API such as Render or Railway
    - Static hosting for client such as Vercel or Netlify
    - Managed NoSQL database such as MongoDB Atlas or Firestore
    - Optional object storage for artifacts

Data flow, step by step

1. User registers in client and sends registration request to API.
2. API validates request, creates user record in database with password hash, returns JWT on successful auth.
3. Client stores token securely and calls GET users me to fetch profile and goals.
4. Client displays dashboard cards using API responses.
5. When user updates a goal or logs progress, client sends POST or PUT to API with JWT in Authorization header.
6. API updates database and writes an audit log entry for data access or modification.
7. Background worker runs scheduled job to compute compliance status and to queue any reminder notifications.
8. Provider requests patient list from API. API checks provider role and returns patient summary with compliance flags.

Minimal data model

User

- id

- name
- email
- passwordHash
- role with values patient or provider
- profile fields such as allergies and medications
- createdAt and updatedAt

Goal

- id
- userId
- goalType such as steps or water or sleep
- targetValue
- progressValue
- date
- createdAt and updatedAt

Reminder

- id
- userId
- title
- dueDate
- status such as pending or done
- createdAt and updatedAt

AuditLog

- id
- userId
- action such as viewProfile or updateGoal
- targetResource
- timestamp
- metadata

PublicContent

- id
- title
- body
- tags

API surface map

Authentication

- POST auth register
- POST auth login
- POST auth refresh token optional

User

- GET users me
- PUT users me

Goals

- GET goals
- POST goals
- PUT goals id
- DELETE goals id optional

Reminders

- GET reminders
- POST reminders
- PUT reminders id
- DELETE reminders id optional

Provider

- GET provider patients
- GET provider patients id details

Public

- GET public health info

Security controls and privacy

- Passwords hashed with bcrypt with appropriate cost parameter.
- JWT short lived with refresh token rotation if you implement refresh.
- Transport security with HTTPS everywhere.
- Role based access control middleware to block data access across roles.
- Audit logging for sensitive operations such as reading or modifying patient health data.
- Environment variables and secrets stored in provider secret manager.
- Consent checkbox at registration documenting data usage.
- Limit sensitive fields returned by APIs for patients that are not the owner or an authorized provider.

Compliance basics to demonstrate

- Encrypt data at rest where available from the cloud provider.
- Use TLS for data in transit.
- Maintain audit trail for access to patient data.
- Minimize data collected and retain only what is required for the MVP.
- Show a simple privacy policy page with how data is used and retained.

Deployment and devops plan

- Continuous integration on push to main branch
  - Linting and unit tests
  - Build frontend and backend
- Continuous deployment
  - Deploy backend to a managed host and set environment variables there
  - Deploy frontend to a static host and configure environment variables for API URL
- Database provision
  - Create managed NoSQL instance and whitelist backend IPs or use private networking

- Background jobs
  - Run a scheduled worker using your host cron feature or a managed job runner
- Monitoring
  - Request logging with a log aggregator
  - Error tracking using a service such as Sentry
  - Basic uptime checks

Scaling and availability

- Start single replica for API and scale horizontally when load increases.
- Use connection pooling for the database.
- Use pagination and limit queries to avoid large reads.
- Cache public content or frequently requested dashboards for short time using an in memory cache.
- Use a managed database with automatic scaling.

Observability and operations

- Structured request logs with request id, user id and latency.
- Error tracking and alerting on high error rate or slow responses.
- Health check endpoint for readiness and liveness.
- Backups for database and a tested restore procedure.

Example deployment diagram in text

Client browser or mobile web
-> CDN and static host hosting client
-> API gateway or API host
-> Application server instances
-> Managed NoSQL database
-> Background worker and job queue
-> Optional notification service for reminders
-> Monitoring and logging sinks

Design trade offs and reasoning

- Use NoSQL for flexible schema to store varied health goals and quick iteration.

- Use JWT for stateless scaleable sessions; consider refresh tokens for long lived sessions.
- Prioritize security and auditability over many features for MVP to match healthcare constraints.
- Keep provider features read heavy and compute compliance in background to minimize API latency.

Next steps I can take for you

Pick one and I will produce it:

- Detailed component diagram image you can use in a presentation.
- OpenAPI specification for the APIs above.
- Starter repo scaffolding for both client and server with auth implemented.