High Level Design Document

Introduction

This High Level Design (HLD) document outlines the architecture and core components for **MediTrack - Patient Record API**. The project delivers a secure backend API for managing patient records, supporting CRUD operations, file uploads for medical reports, and user authentication. The solution is built using Node.js, Express, and MongoDB.

---

1. System Architecture Overview

Architecture Description:
MediTrack is structured as a RESTful API backend. Clients interact via HTTP(S) requests. The API server handles authentication, business logic, and file uploads, and persists data in MongoDB. Medical report files are stored in a dedicated storage location.

Component	Role/Responsibility
API Server	Handles HTTP requests, authentication, business logic
Database (MongoDB)	Stores patient records, user data, metadata
File Storage	Stores uploaded medical report files
Authentication	Secures endpoints, manages user sessions/tokens

---

2. Component Interactions

Sequence Step	Interaction Description
1	Client sends HTTP request (e.g., login, CRUD, file upload) to API Server
2	API Server authenticates user (via token/session)
3	API Server processes request, interacts with MongoDB for data operations
4	For file uploads, API Server stores files in File Storage and saves metadata in MongoDB
5	API Server returns response to client

---

3. Data Flow Overview

Data Flow	Source	Destination	Description
User Authentication	Client	API Server	Login/signup requests, token issuance
Patient Record CRUD	Client	API Server	Create, read, update, delete patient records
Data Persistence	API Server	MongoDB	Store/retrieve patient/user data, file refs

File Upload	Client	API Server	Upload medical reports (files)
File Storage	API Server	File Storage	Save uploaded files, link to patient records

---

4. Technology Stack

Layer/Function	Technology/Framework
API Server	Node.js, Express
Database	MongoDB
Authentication	JWT (JSON Web Tokens)
File Upload	Multer (Express middleware)
File Storage	Local disk or cloud storage
API Documentation	Swagger/OpenAPI (optional)

---

5. Scalability & Reliability

- **Security:** All endpoints require authentication; sensitive data is encrypted in transit (HTTPS).
- **Scalability:** Stateless API design enables horizontal scaling; MongoDB supports sharding/replication.
- **Reliability:** Input validation, error handling, and logging are implemented; file uploads are validated for type/size.
- **Data Integrity:** Patient records and file references are consistently managed in MongoDB.

End of Document