SOFTWARE DESIGN DOCUMENT
MADAD-MITRA

AUTHORS:-

Deepanshu

Harshit Kukreja

Table Of Content:-

Project Overview

Vision Statement: "To revolutionize the domestic service industry by creating a seamless, trusted platform that enhances the quality of life for households while providing sustainable employment opportunities for service professionals."

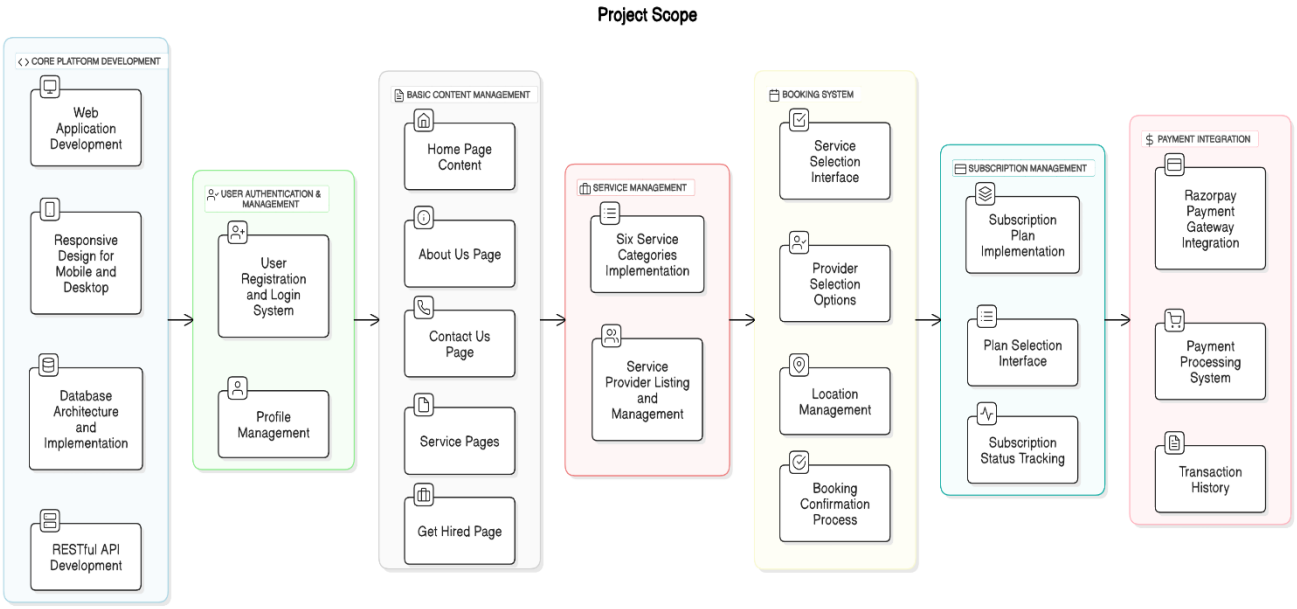
Mission Statement: "To deliver reliable, professional-grade domestic services through a user-friendly platform that connects households with verified service providers, offering customizable subscription plans that meet diverse household needs."

Introduction

Madad-Mitra represents an innovative solution in the domestic service industry, designed to bridge the gap between households seeking reliable domestic help and verified service professionals. The platform addresses the growing demand for organized, trustworthy domestic services in urban areas by offering a comprehensive digital marketplace for various household services including cooking, cleaning, childcare, and full-time domestic assistance. Through its subscription-based model, Madad-Mitra ensures consistent service delivery while providing flexible options that cater to diverse household needs.

Scope:

The technical scope of Madad-Mitra involves developing a web application using the MERN (MongoDB, Express.js, React.js, Node.js) stack. The frontend will be built using React.js with Redux for state management, implementing responsive UI components for user authentication, service booking, profile management, and subscription handling, while integrating Razorpay for payments. The backend will be developed using Node.js and Express.js, creating RESTful APIs to handle business logic, user authentication with JWT, and data management, with MongoDB serving as the database to store user profiles, service listings, bookings, and subscription data using well-structured schemas.

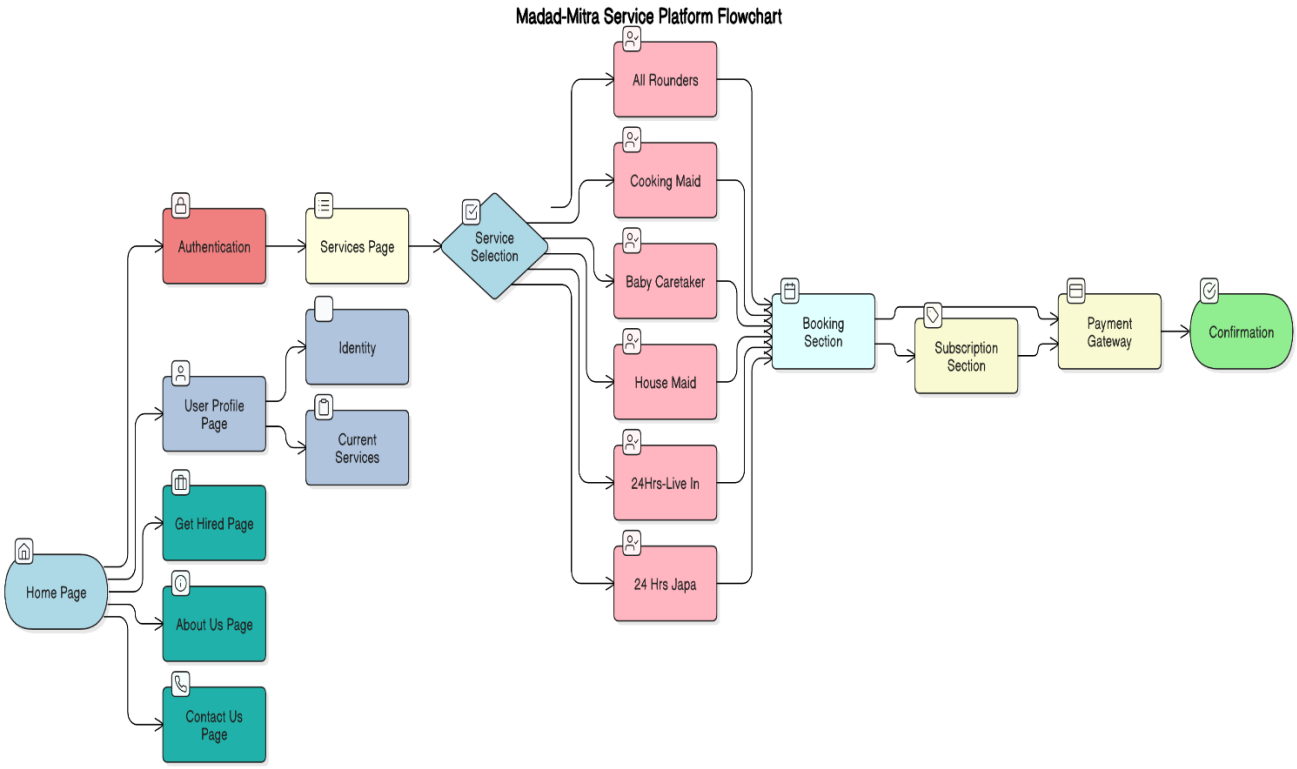


Madad Mitra Process Flow

1. Access the platform through Home Page and verify your identity via Authentication
2. Navigate to Services Page to view available domestic help options
3. Use Service Selection to choose your required service category
4. Select specific service type (All Rounders/Cooking Maid/Baby Caretaker/House Maid/24Hrs-Live In/24 Hrs Japa)
5. Proceed to Booking Section to schedule your service
6. Choose your service duration in Subscription Section
7. Complete payment through Payment Gateway
8. Receive Confirmation of your booking

Optional flows:

- Check your profile and current services through User Profile Page
- Register as a service provider via Get Hired Page
- Learn about the platform through About Us Page
- Get in touch through Contact Us Page



Architecture

High-Level Arhitecture:

Architecture Overview: Three-Tier System Design

1. Frontend (Presentation Layer)

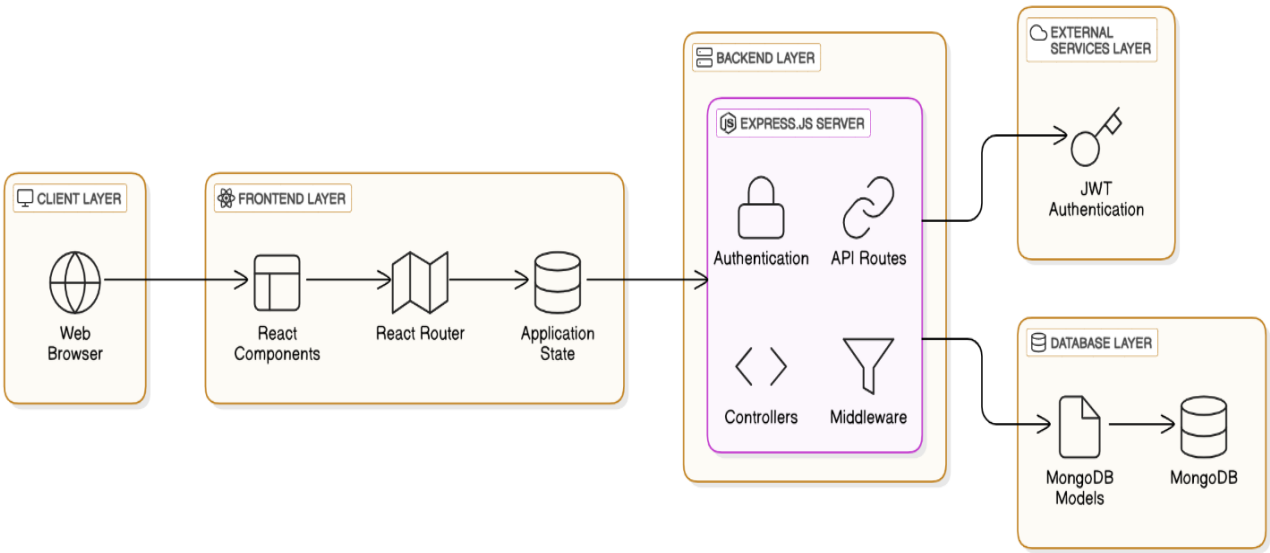
- Implements React.js framework for rendering dynamic user interfaces and managing client-side state management and routing through React-Router-DOM.
- Handles user interactions, form validations, and responsive design while maintaining component-based architecture for scalability

2. Backend API (Application Layer)

- Leverages Express.js on Node.js runtime for RESTful API endpoints, with JWT authentication and middleware processing
- Orchestrates business logic, data validation, and secure API routing while integrating external services.

3. Database (Data Layer)

- Utilizes MongoDB as the NoSQL database solution for flexible schema design and document-oriented data storage
- Implements MongoDB Models for data structure definition and maintains data persistence with optimized query performance



Micro-Architecture:

Madad-Mitra implements a modern, scalable microservices architecture leveraging the MERN (MongoDB, Express.js, React.js, Node.js) technology stack. The architecture follows a client-server model with clear separation of concerns, where the frontend client communicates with the backend server through RESTful API endpoints. The system employs a modular approach with containerized services, ensuring high maintainability and scalability.

Frontend Architecture:

- React Components: Custom UI components for services, booking, profiles, and payments
- Routing: React Router handling protected and public routes with middleware checks
- API Integration: Fetch for REST API calls with request/response interceptors

Backend Architecture:

- Controllers: Handle business logic for users, services, bookings, and payments
- Middleware: Authentication, request validation, error handling, and file upload processing
- Models: MongoDB schemas for Users, Services, Bookings, Payments, and Reviews
- Routes: API endpoint definitions with proper HTTP method handlers

Database Design:

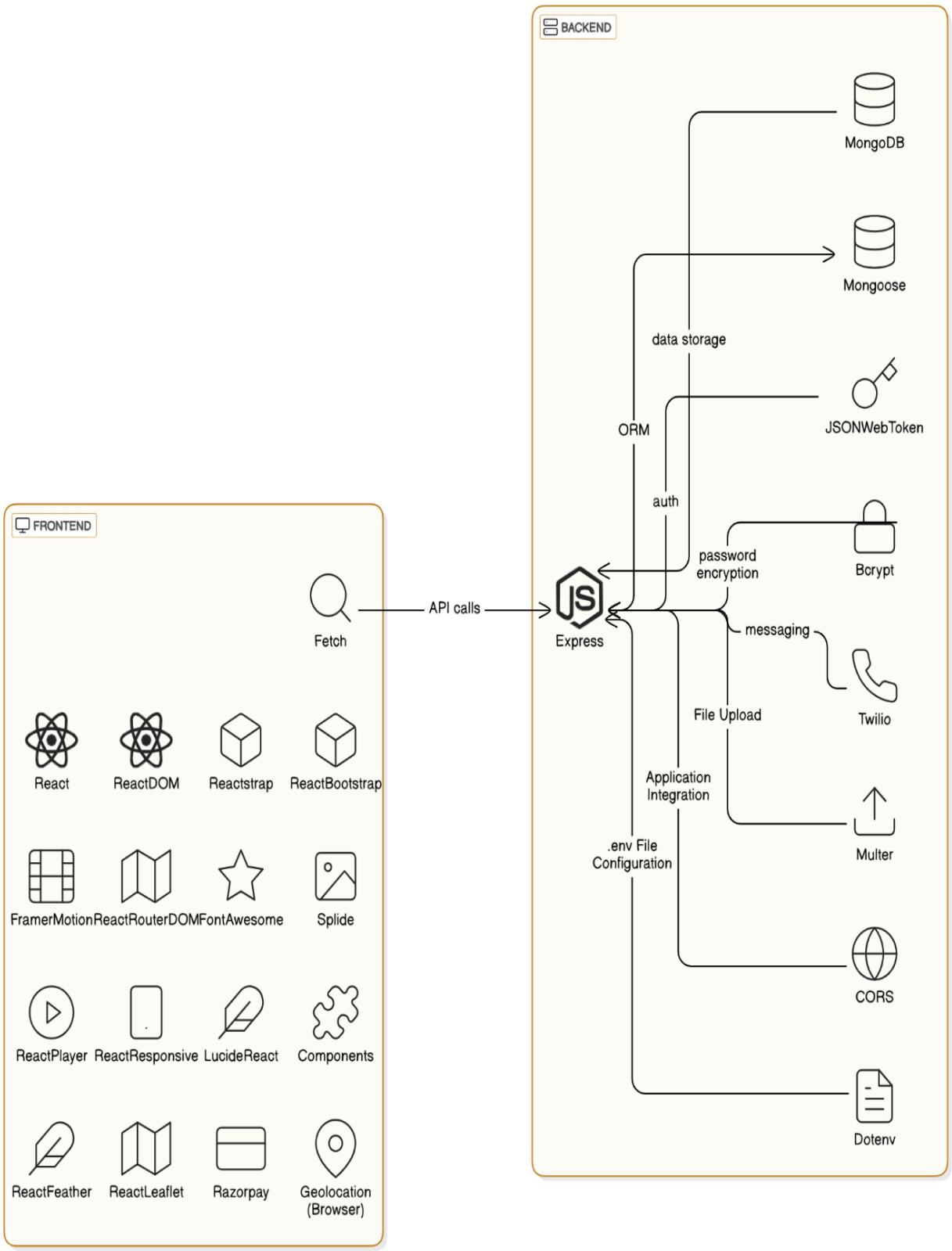
- Users Collection: Customer and service provider profiles with role-based schemas
- Services Collection: Service categories, pricing, and availability information
- Bookings Collection: Booking records with status tracking and service details
- Subscriptions Collection: Plan details, duration, and features mapping
- Transactions Collection: Payment records linked to bookings and subscriptions

Security Layer:

- JWT Authentication: Token-based user session management
- Password Encryption: Bcrypt hashing for secure password storage
- File Security: Secure file upload handling with type checking and size limits

Integration Layer:

- Payment Gateway: Razorpay API integration for payment processing
- Notification Service: Email and SMS notification handling
- File Storage: Multer configuration for document storage
- External APIs: Integration points for maps and verification services


BACKEND
MongoDB
Mongoose
data storage
JSONWebToken
ORM
auth
password
encryption
Bcrypt
FRONTEND
API calls
Fetch
Express
messaging
Twilio
File Upload
React
ReactDOM
Reactstrap
ReactBootstrap
Application
Integration
Multer
.env File
Configuration
FramerMotion
ReactRouterDOM
FontAwesome
Splide
CORS
ReactPlayer
ReactResponsive
LucideReact
Components
Dotenv
ReactFeather
ReactLeaflet
Razorpay
Geolocation
(Browser)

Frontend Design

Navigation Layer

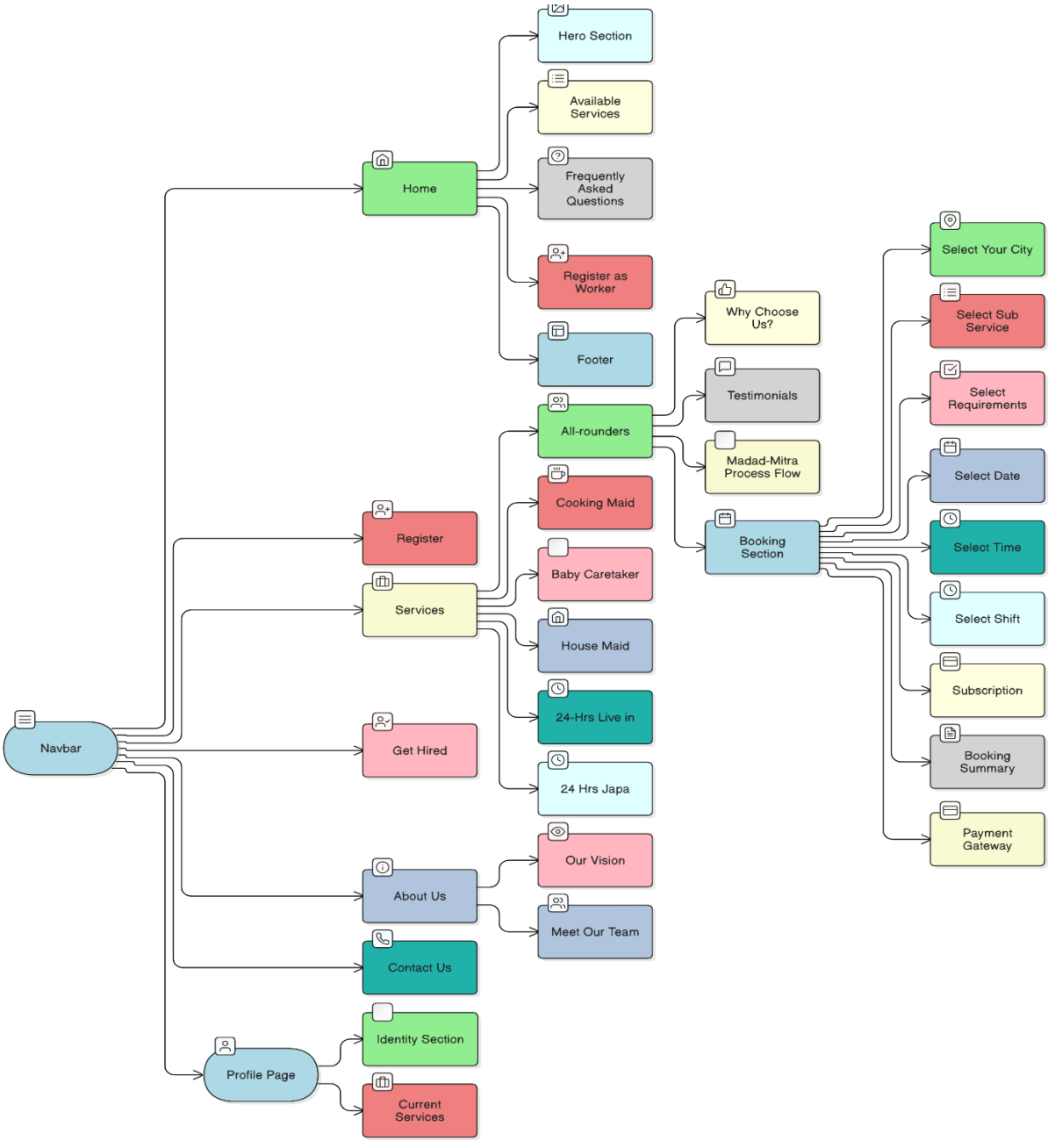
Navbar: Root component with core navigation links for website accessibility.

Primary Routes

1. Home Route:
    - Hero Section > Showcases main services and value proposition
    - Available Services > Service catalog display
    - FAQ Section > Common queries and responses
    - Register as Worker > Get Hired > Worker onboarding portal
    - Footer > Site-wide navigation and information
2. Register Route: Sign Up > Login
3. Services Route:
    - Service Categories:
        - All-rounders > Multi-skilled service providers
        - Cooking Maid > Kitchen service specialists
        - Baby Caretaker > Childcare professionals
        - House Maid > Housekeeping staff
        - 24-Hrs Live-in > Full-time residential care
        - 24 Hrs Japa > Specialized service category
    - Booking Flow:
        - City Selection > Location-based filtering
        - Sub-Service Selection > Specific service type
        - Requirements Input > Service customization
        - Date Selection > Scheduling
        - Time Selection > Time slot booking
        - Shift Selection > Duration preference
        - Subscription > Service plan options
        - Booking Summary > Order review
        - Payment Gateway > Transaction processing
    - Testimonials: Client feedback integration

- Madad-Mitra Process Flow: Service delivery workflow

4. About Us Route:
    - Our Vision > Company mission statement
    - Meet Our Team > Team member profiles
5. Contact Us Route:
    - Contact information and support
6. Profile Page Route:
    - Identity Section > User verification
    - Current Services > Active service management



Backend Design - API, Database Schema and Middleware

Database Design Models

The database design can be divided into three main functional areas:

1. Service Management
2. User Management and Bookings
3. Worker and Hiring System

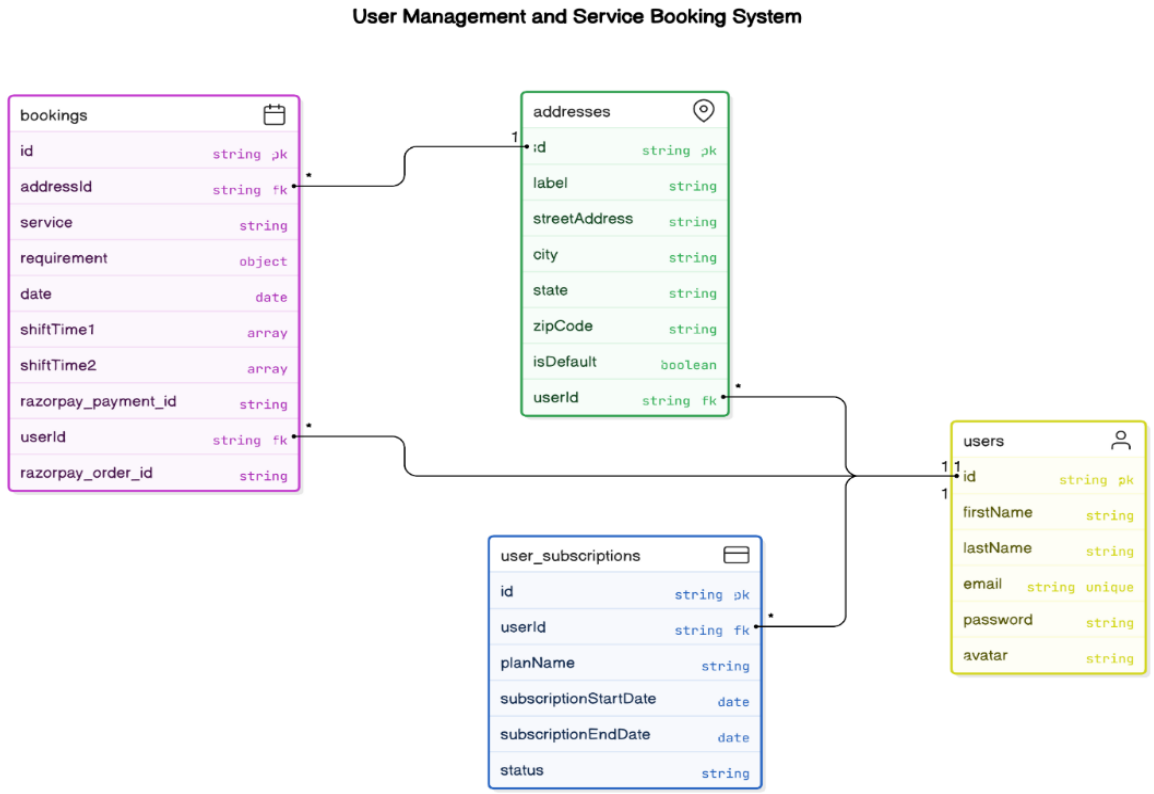
1. User Management and Booking System

Core User Management users: Stores essential user information.

Booking System bookings: Handles service reservations.

Address Management addresses: Maintains user location information.

Subscription Tracking user_subscriptions: Monitors user subscription status.



1. Service Management Components

Core Service Tables all_service: Stores the main service catalog with fields.

Service Customization service_types: Contains detailed service configurations.

Subscription Management subscription_plan: Manages service subscriptions.

Testimonial System testimonial: Captures service feedback.

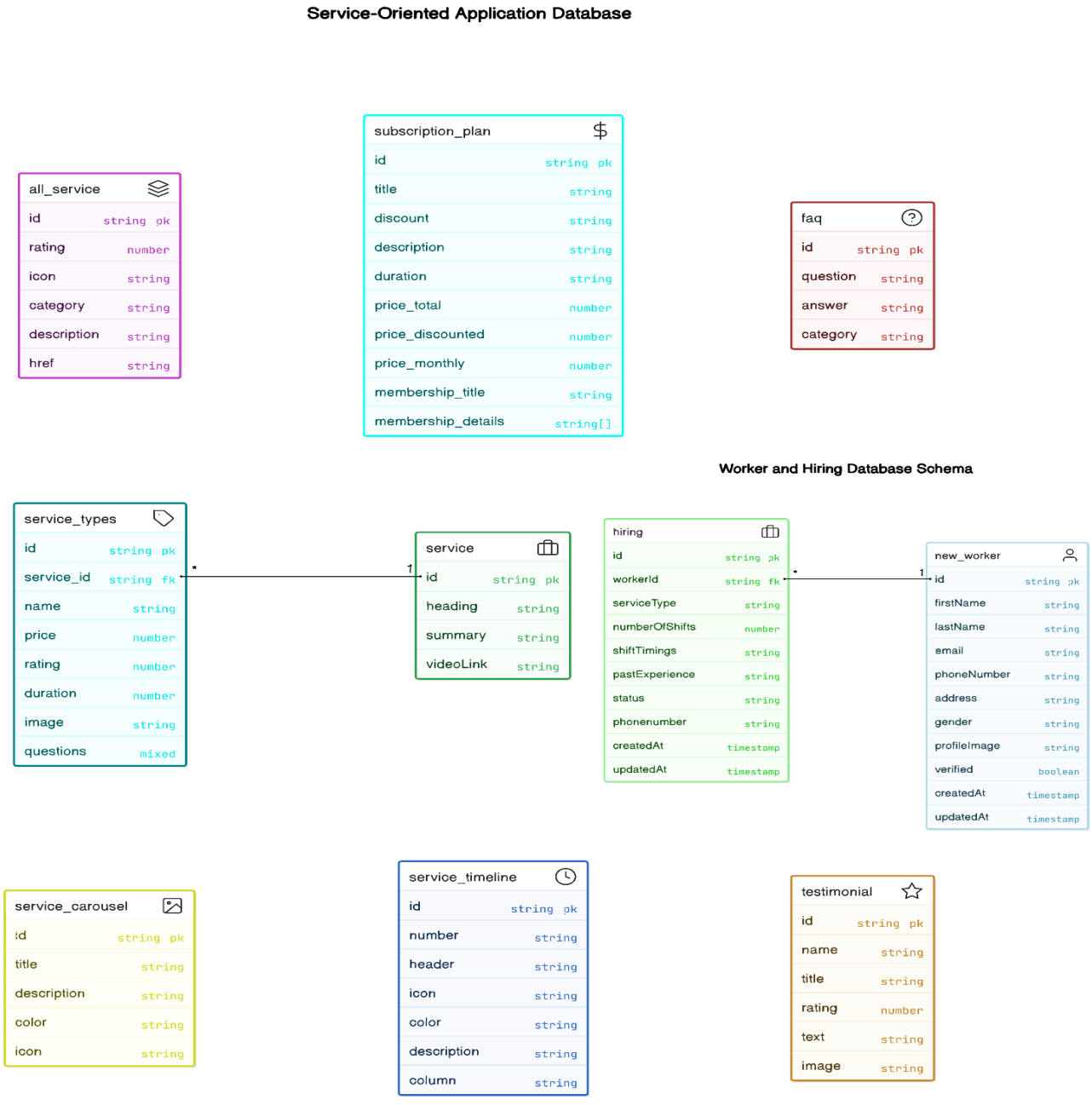
Supporting Content Tables

- faq: Stores frequently asked questions organized by category
- service_carousel: Manages promotional content with styling options.
- service_timeline: Tracks service-related temporal information.

3. Worker and Hiring System

Worker Management worker: Stores service provider information.

Service Assignment hiring: Manages worker assignments.



Backend API Routes Structure with HTTP Methods

Root Level Routes

- **Address Routes**
  - GET /address - Fetch addresses POST /address - Create new address
- **City Routes**
  - GET /city - Get city list GET /city/:id - Get specific city details
- **Booking Routes**
  - POST /bookingdata - Create booking GET /bookingdata - Get booking details

Service & Subscription Routes

- **Service Data**
  - GET /servicedataroute - Fetch services
  - POST /servicedataroute - Add new service
- **Subscription Plans**
  - GET /subscriptionplandata - Get plans
  - POST /usersubscription - Create subscription

Static Content Routes

- GET /services - Get service list GET /service-carousel - Get carousel data
- GET /service-timeline - Get timeline data GET /testimonials - Get testimonials
- GET /all-services - Get complete service catalog GET /faq - Get FAQ list

Worker API Routes (/newworkers)

- POST /newworkers/register - Register new worker
- POST /newworkers/send-otp - Send OTP
- POST /newworkers/verify-otp - Verify OTP
- GET /newworkers - Get workers list
- POST /hiring – new worker enrollment

User Management Routes (/user)

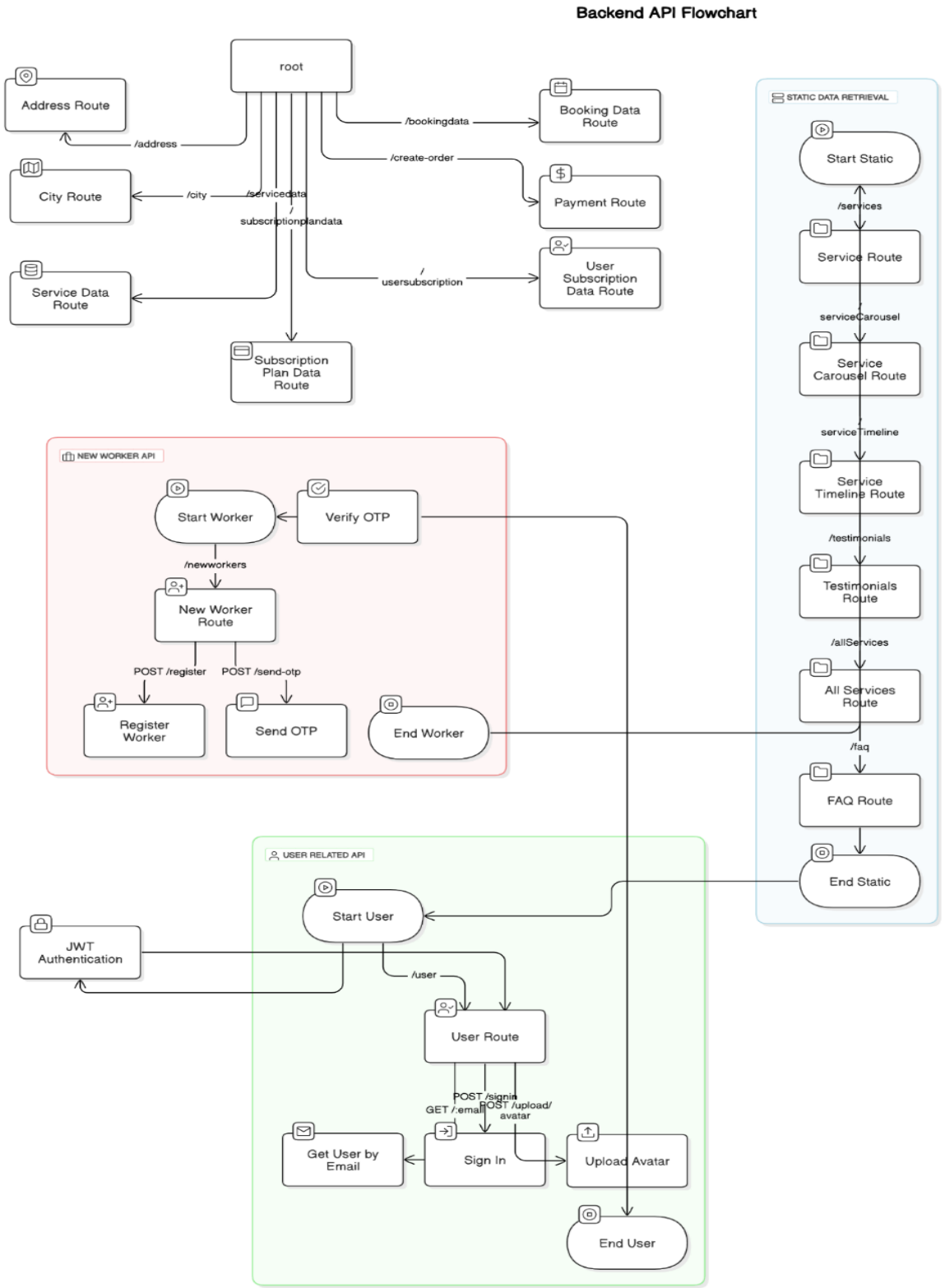
- POST /user/signin - User login POST /user/ - New User SignUp
- GET /user/email/:email - Verify email GET /user/:id - Get user details
- POST /user/upload-avatar - Upload profile picture
- Get /auth/jwt – Get details using JWT token

Payment Routes

- POST /payment/create - Create payment order

Subscription Management

- POST /subscription/ - Create new subscription GET /subscription/ - Check status

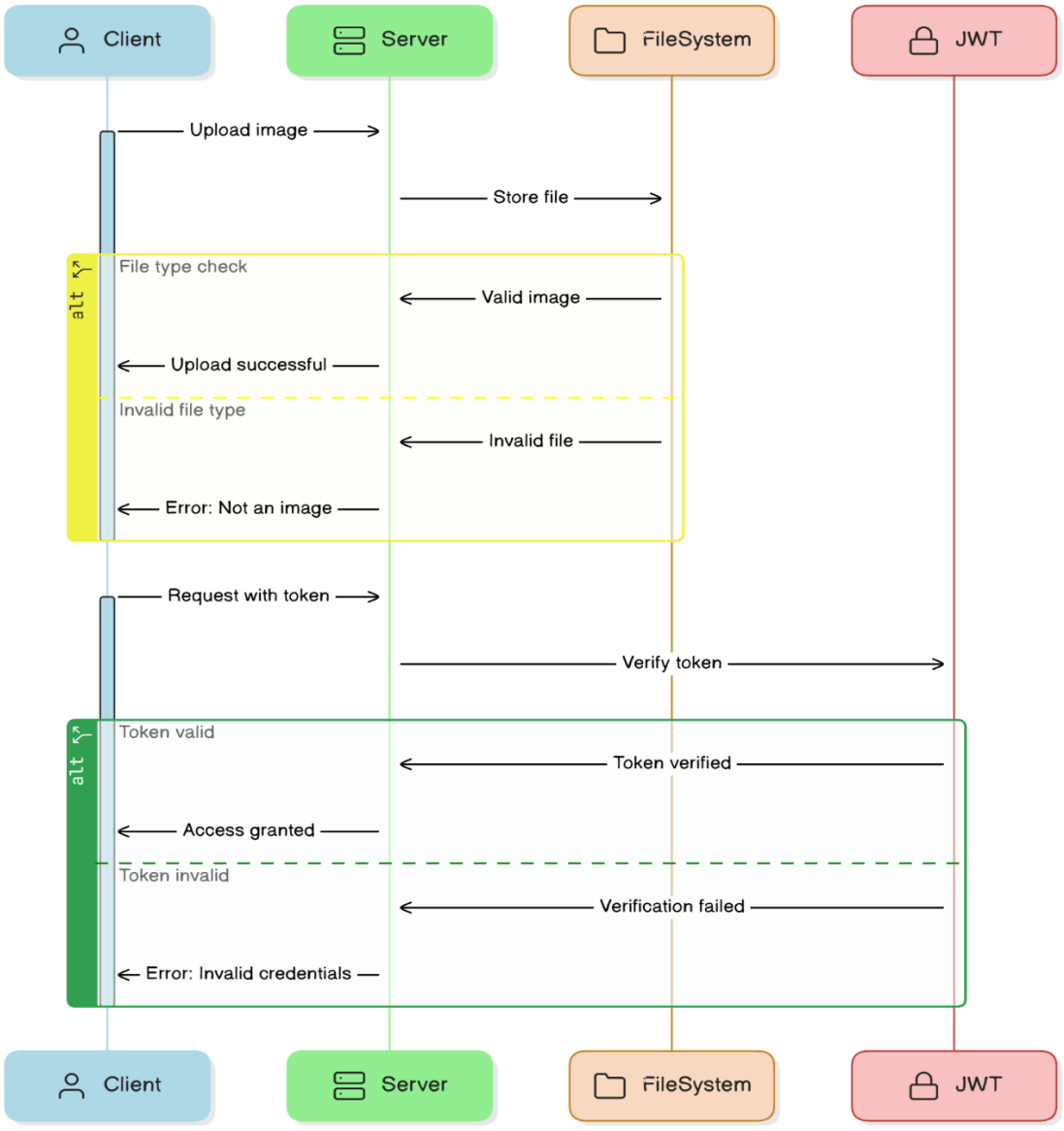


Middleware

Authentication Middleware (verifyToken) A JWT-based authentication middleware that validates user credentials through token verification. It extracts the authorization token from request headers, decodes it using a secret key, and injects the user's email into the request object for downstream use. Failed authentication attempts return a 401 status code.

File Upload Middleware (upload) A configurable file handling middleware using Multer for managing image uploads. Key features:

- Implements disk-based storage with unique filename generation using timestamp and random suffix
- Restricts uploads to image files through MIME type validation
- Enforces a 2MB file size limit
- Stores uploads in a dedicated avatar directory



Security Considerations

Security Layer is divided into three main components: Client Security, Communication Security, and Server Security.

1. Client Security:
    - JWT Token handling: Stored in localStorage for persistent authentication
    - User Input processing: Goes through input validation
    - Protected Routes: Require authentication checks
    - These components work together to ensure validated data and proper token management
2. Communication Security:
    - API Requests: Central hub for client-server communication
    - CORS Policy: Controls which domains can access the API
    - Handles both validated data from user input and authentication tokens
3. Server Security:

A. Password Security Flow:

- Salt Generation: Creates unique salt for each password
- Bcrypt Hashing: Uses the salt to hash passwords securely
- Secure Storage: Safely stores hashed passwords

B. JWT Process:

- Secret Key: Used for signing tokens
- Token Signing: Creates secure JWTs
- Token Verification: Validates incoming tokens

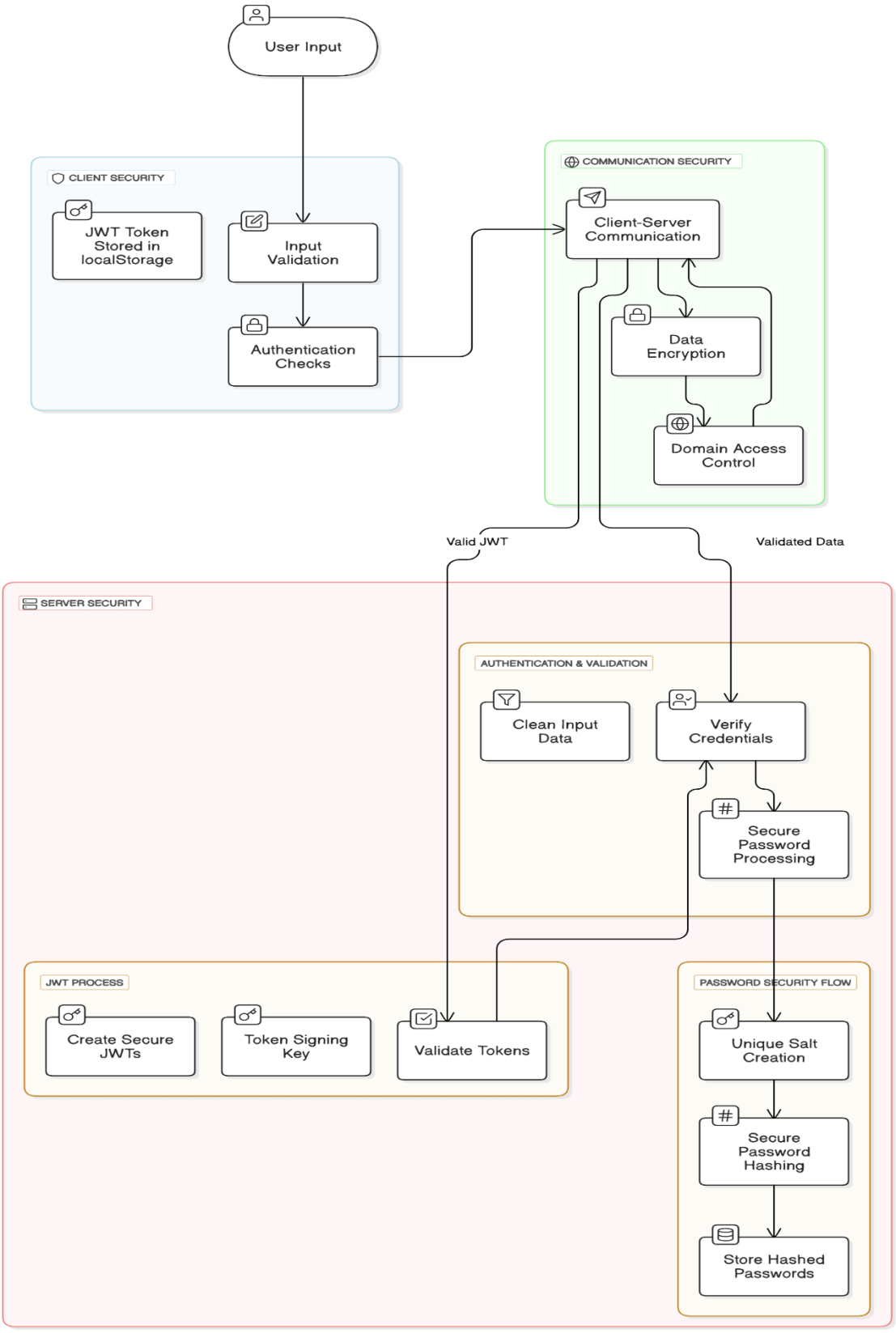
C. Authentication & Validation:

- Authentication: Verifies user credentials
- Password Hashing: Processes passwords securely
- Sanitization: Cleans and validates input data

The overall flow:

1. User provides input, which is validated on the client side
2. Protected routes check for valid JWT tokens
3. API requests are made with validated data and tokens

4. Server authenticates requests using JWT process
5. Passwords are securely handled using salt and hashing
6. CORS policies restrict unauthorized domain access



Testing Strategy :-

Frontend Testing Tools:-

1) Browser Developer Tool:-

- Network tab monitoring for API requests and responses
- Elements tab for DOM inspection and manipulation
- Console for JavaScript debugging and error monitoring
- Application tab for localStorage and session management

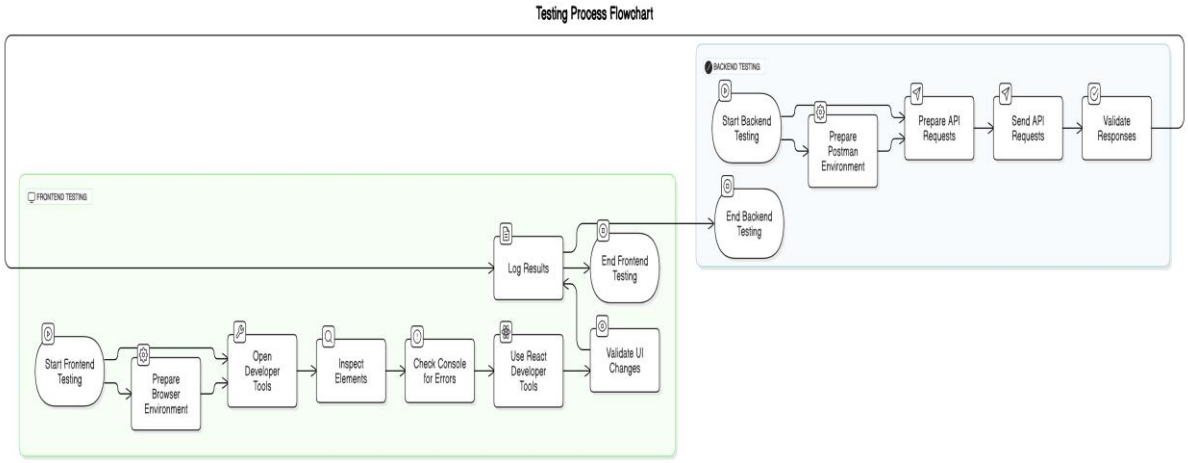
2**) React Developer Tools :-**

- Components tab for component hierarchy inspection
- Props and state monitoring
- Hook dependencies tracking

Backend Testing Tools:-

1**) Postman :-**

- API endpoint testing
- Request method validation (GET, POST, PUT, DELETE)
- Response status code verification
- Authentication testing



Deployment Plans

Environment Configuration

Server-Side Variables

- PORT: Application server port configuration
- DB_CONNECTION_STRING: MongoDB connection credentials and path
- JWT_SECRET: Authentication token encryption key

Client-Side Variables

- VITE_API_URL: Server endpoint reference URL

Deployment Steps:

1. **Server Deployment Strategy**
    - Select cloud platform (AWS EC2/Heroku/DigitalOcean)
    - Implement secure environment variable configuration
    - Configure PM2 for application lifecycle management
2. **Client Deployment Strategy**
    - Execute production build command: npm run build
    - Deploy to static hosting service (Netlify/Vercel) or serve through NGINX.
3. **Database Implementation**
    - Deploy on MongoDB Atlas platform
    - Configure network security rules and access control measures
4. **Continuous Development**
    - Implement GitHub Actions workflow
    - Configure automated deployment