Project details - quickFix

Problem description:-

quickFix -Description:- QuickFix is a mobile and web application designed to connect homeowners with verified local technicians for various home repair services such as plumbing, electrical work, carpentry, and appliance repairs. Users can book services instantly, track the technician's arrival in real-time, and make secure payments through the app. QuickFix aims to simplify the process of finding trustworthy and skilled repair professionals, reducing wait times and ensuring quality service. The platform also includes a rating and review system to maintain high service standards. QuickFix is a scalable solution for busy urban households looking for reliable, quick, and affordable home repair assistanc

---

🔷 Problem Statement (Easy Explanation)

People face a common problem:
👉 When something breaks at home (fan, AC, plumbing, electrical issue), it's hard to find a **trusted and skilled repair person quickly**.
👉 Sometimes they overcharge, don't show up on time, or do poor-quality work.

QuickFix solves this problem by making a **web + mobile app** where:

- Homeowners can book services like plumbing, carpentry, electrical repair, AC fixing, etc.
- Verified technicians are listed.
- User can select service → book instantly → track technician arrival → pay online securely.
- Rating & review keeps the quality high.

Project details - quickFix

Recommended Tech Stack

Frontend:

- **HTML/CSS/JavaScript**: Core web technologies
- **Bootstrap**: For responsive design and UI components
- **jQuery**: For simpler DOM manipulation (optional)
- **Mapbox/Google Maps API**: For real-time tracking
- **React js**. : easy to convert this website to mobile application

Backend:

- **Backend → Node.js + Express.js (API, authentication, booking system).**
- **Database → MongoDB (store users, technicians, services, bookings, payments).**
- **Authentication → JWT (login/signup).**
- **Payments → Razorpay/Stripe integration.**

Additional Services:

- **Twilio**: For SMS notifications
- **Firebase**: For real-time features (optional)

two roles:

- **Admin** → manages platform (users, technicians, bookings, payments).
- **User (Homeowner)** → books services.
- (Optional: **Technician**) → accepts bookings, updates job status.

Project details - quickFix

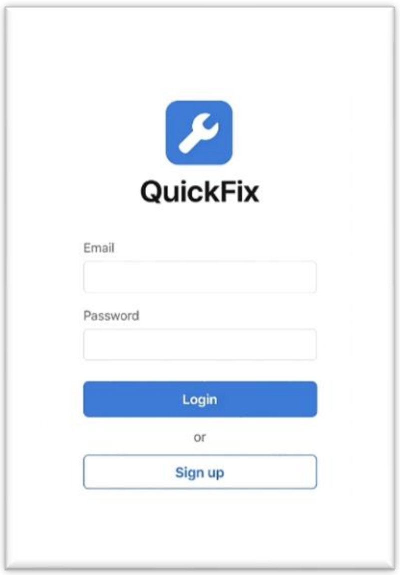
Frontend design pages list and section details :-

1) 1. Navigation Bar (Navbar)

This is the menu at the top that stays fixed as you scroll. It's the primary way users navigate the "different pages" (sections) of the website.

- **Logo:** "QuickFix"
- **Menu Items:** Home, Services, How It Works, Technicians
- **User Actions:** Login and Sign Up buttons (which would typically link to separate pages in a real app).

2) login page :-



data-model for backend database

```
{
  "userId": "U123",
  "name": "Arman",
  "email": "arman@example.com",
  "password": "hashed_pass",
  "role": "user", // can be user, admin, technician
  "address": "Bhavnagar, Gujarat",
  "phone": "*********"
}
```

Project details - quickFix

wirefream

```
--------------------------------
 |       QuickFix Logo          |
 |   [ Email: _____________ ]      |
 |   [ Password: __________ ]      |
 |   [ Login Button ]           |
 |   (or) [ Signup Button ]       |
 --------------------------------
```

? Simple authentication page

? Option to switch between login & signup

Main page all section details :-

2. Main Page Sections (Accessed via the Navbar)

Section 1: Hero Section

- **Purpose:** The first thing a user sees. It's a large, eye-catching banner designed to grab attention and state the core value proposition.
- **Content:** A large background image, the main headline ("Quality Home Repairs, On Demand"), a short description, and a prominent call-to-action button ("Book a Service Now"). (optional)

Section 2: Services Section (`id="services"`)

- **Purpose:** To showcase the different types of repair services QuickFix offers.
- **Content:** A grid of four cards, each for a different service:

1. **Plumbing**

2. **Electrical**
3. **Carpentry**
4. **Appliance Repair…………**

- Each card has an image, a title, a short description, and a "Book Now" button.

Section 4: Technicians Section (`id="technicians"`)

- **Purpose:** To build trust by putting a "face" to the service. It shows verified, highly-rated technicians.
- **Content:** A grid of three profile cards for sample technicians:

1. **Michael Johnson** (Plumbing & Electrical)
2. **Sarah Williams** (Appliance Repair)
3. **David Chen** (Carpentry)

- Each card includes a photo, name, distance away, star rating, number of reviews, and a short bio. + **btn [see more technicians]**

Services page details :-

Wirefream :-

```
----------------------------------------------
| [🔍 Search Services]    [Filter Icon ⍰]     |
|--------------------------------------------|
| [All]  [Popular]  [Plumbing]  [Electrical]  |  ← New Category Tabs
|--------------------------------------------|
|            --- Popular ---                |
| [🛠 Emergency Plumber] [View All ->]       |
| < Scrollable Card 1 >  < Card 2 >         |  ← New Horizontal Scroll Section
|--------------------------------------------|
|             --- All Services ---         |
```

Project details - quickFix

```
|--------------------------------------------|
|  [SERVICE CARD]                            |
|   Plumbing Repair                          |
|  Fix leaks, clogs, and installations       |  ← Added Description
|  ★★★★☆ (245)    ⏱ 1-2h    Starting at $89  |  ← Added Rating, Time, Price
|  [ Book Now ]                              |  ← More prominent button
|--------------------------------------------|
|  [SERVICE CARD]                            |
|   Electrical Work                          |
|  Lighting, outlets, panel upgrades         |
|  ★★★★½ (189)    ⏱ 2-3h    Starting at $99  |
|  [ Book Now ]                              |
|--------------------------------------------|
|  [ Bottom Menu: (Home) | Bookings | Profile ]|
|  (Services icon is highlighted)            |
```

Service section data model (data backend to frontend)

```
{
  "serviceId": "S789",
  "serviceName": "Plumbing Repair",
  "description": "Fix taps, pipes, water leakage",
  "price": 200
}
```

Project details - quickFix

3. Booking Screen

```
--------------------------------
 |  Service: Plumbing Repair      |
 |-------------------------------|
 |   Date: [Select Date]        |
 |  ⏰ Time: [Select Time]        |
 |   Address: __________________ |
 |   Price: ₹200               |
 |-------------------------------|
 |       [ Confirm Booking ]     |
 --------------------------------
```

Booking datamodel

```
{
  "bookingId": "B001",
  "userId": "U123",
  "technicianId": "T456",
  "serviceId": "S789",
  "date": "2025-08-30",
  "time": "10:30 AM",
  "status": "Confirmed", // Pending, Accepted, Completed, Cancelled
  "paymentStatus": "Paid"
}
```

Technician review datamode (optional

```
{
  "reviewId": "R111",
  "bookingId": "B001",
  "userId": "U123",
  "technicianId": "T456",
  "rating": 5,
  "comment": "Great work! Quick and professional."
}
```

wirefream

```
--------------------------------
|  Job Completed 🎉             |
|------------------------------|
|  ☆ ☆ ☆ ☆ ☆ (Rating)           |
|  [ Write a Review ]          |
|  ___________________________    |
|                      |
|  [ Submit Review ]           |
--------------------------------
```

Project details - quickFix

Admin ,user ,technician role in this website :-

Note(this all role we can also change) this role is only for thinking and analyzing the project)

1. Homeowner (Regular User) Role:

- **Service Booking**: Browse services, select service type, schedule appointments, provide problem details
- **Real-time Tracking**: View technician location and estimated arrival time
- **Payment Processing**: Securely pay for services through integrated payment gateway
- **Review System**: Rate and review completed services
- **History Management**: View booking history and receipts
- **Profile Management**: Update personal information and payment methods

2. Technician Role:

- **Service Management**: View assigned jobs, update job status (en route, arrived, in progress, completed)
- **Schedule Management**: Set availability, manage appointments
- **Earnings Tracking**: View completed jobs and earnings
- **Profile Management**: Showcase skills, certifications, and experience
- **Communication**: Chat with homeowners for clarifications

3. Admin Role:

- **User Management**: Approve/reject technician registrations, manage user accounts
- **Service Management**: Add/update service categories and pricing
- **Analytics Dashboard**: View business metrics, popular services, revenue reports
- **Dispute Resolution**: Handle complaints and service issues
- **Content Management**: Update website content, promotions, and notifications
- **System Monitoring**: Monitor platform performance and security

Project details - quickFix

User dashboard :-

1. Homeowner Dashboard (User Panel)
This is for the homeowners who request services.
Key Features:
- Service Booking: Users can browse services, select a service, choose a time slot, and book.
- Real-time Tracking: Once a technician is assigned, they can track the technician's location on a map.
- Payment History: View past transactions and invoices.
- Review and Rating: After service completion, they can rate and review the technician.
- Profile Management: Update personal information, address, and payment methods.

Workflow:
- User logs in and sees a dashboard with upcoming appointments, past services, and option to book new service.
- When booking, they select service type, describe the problem, choose time, and make payment.
- They can chat with the assigned technician and track their arrival.
- After service, they receive a receipt and can rate the service.

## 2. Technician Dashboard
This is for the service technicians.
Key Features:
- Job Management: View assigned jobs, update job status (e.g., accepted, en route, started, completed).
- Schedule Management: Set availability and manage appointments.
- Earnings Tracking: View completed jobs, earnings, and payment history.
- Profile Management: Update skills, certifications, and personal information.
- Navigation: Get directions to the customer's location.

Workflow:
- Technician logs in and sees a list of assigned jobs for the day.
- They can accept or reject new job requests (if the platform allows).

- They update their status (leaving, arrived, etc.) which triggers notifications to the homeowner.
- After completing the job, they mark it as done and may collect a signature or confirmation from the homeowner.
- They can view their earnings and upcoming schedule.

## 3. Platform Admin Dashboard

This is for the platform administrators who manage the entire system.

Key Features:

- User Management: Manage homeowners and technicians (activate, deactivate, delete accounts).
- Service Management: Manage service categories, pricing, and details.
- Job Monitoring: Monitor ongoing and completed jobs, intervene in case of disputes.
- Financial Management: View overall revenue, technician payouts, and platform earnings.
- Analytics: View reports on popular services, user growth, technician performance, etc.
- Content Management: Manage FAQs, terms of service, privacy policy, and other content.

Workflow:

- Admin logs in to see an overview of the platform's performance.
- They can approve new technician registrations (if verification is required).
- They handle customer support escalations and disputes.
- They generate reports for business analysis.

Project details - quickFix

How These Dashboards Work Together Logically

User Journey Flow:

1. **Homeowner requests service** through their dashboard
2. **System matches** the request with available technicians based on:
   - Service type required
   - Technician skills and certifications
   - Geographic proximity
   - Technician ratings and availability
3. **Technician receives notification** of new job request in their dashboard
4. **Technician accepts** the job through their interface
5. **Homeowner receives confirmation** and can track technician's arrival
6. **Service is performed** and technician marks it as completed
7. **Payment is processed** automatically through the platform
8. **Homeowner rates** the service quality
9. **Admin monitors** the entire process and can intervene if issues arise

Backend Logic:

Database management for users, services, bookings, and transactions

Real-time tracking system using mapping APIs

Payment processing integration with secure gateways

Notification system for all status updates

Matching algorithm that connects homeowners with suitable technicians

Review and rating system that maintains service quality

Project details - quickFix

Technician Onboarding and Notification System for QuickFix

1. Manual Technician Onboarding Process:

- Your team collects technician details through offline methods (in-person meetings, paper forms, etc.)
- Information collected includes: Name, Phone Number, Service Type, Service Areas, Experience, Certification Details, and ID Proof
- Your admin team enters this information into the QuickFix admin dashboard
- Each technician is assigned a unique ID in the system

2. SMS/WhatsApp Notification System:

- When a customer books a service, your system automatically matches them with suitable technicians based on:
  - Service type needed
  - Technician's service area
  - Technician's current availability
  - Technician's rating (if available)
- The system sends a notification via SMS/WhatsApp to the selected technician:

Frontend design

The main routes (pages) we need for QuickFix are:

1. Homepage
2. Services Page (listing all services)
3. Service Detail Page (specific service)
4. Booking Page (form to book a service)
5. Technician Profile Page
6. User Dashboard (for homeowners to see their bookings)

7. Technician Dashboard (for technicians to manage their jobs)
8. Admin Dashboard (for admin to manage the platform)
9. About Us Page
10. Contact Page
11. Login/Register Pages

User Authentication Routes

9. **Login** (`/login.html`)
   - Email/phone and password login
   - Social login options (if implemented)
   - "Forgot password" link
10. **Registration** (`/register.html`)
   - User registration form
   - Technician registration form (different path or toggle)
   - Terms and conditions
11. **Forgot Password** (`/forgot-password.html`)
   - Password reset request form
12. **Password Reset** (`/reset-password.html?token=:token`)
   - New password setup form

Project details - quickFix

Backend planning

For Homeowners:

- Service browsing and search
- Booking creation and management
- Payment processing
- Real-time tracking of technicians
- Review system

For Technicians:

- Profile management
- Job acceptance/decline system
- Schedule management
- Earnings tracking
- Location sharing

For Admins:

- User management
- Service management
- Booking oversight
- Financial reporting
- Content management

---

1. **API specification (Express REST endpoints). Implement these routes (with validation and error handling):**
    - `POST /api/auth/signup` → register (role: user|technician). Validate required fields, hash password, send verification/confirmation SMS/email.
    - `POST /api/auth/login` → login, return JWT access token + user role.

- `POST /api/auth/forgot-password` → generate reset token + email link.
- `POST /api/auth/reset-password` → accept token, update password.
- `GET /api/services` → list services (filter by category, search q, pagination).
- `GET /api/services/:id` → service details + reviews.
- `POST /api/services` → admin create service.
- `PATCH /api/services/:id` → admin update service.
- `GET /api/technicians` → list technicians (filter by skill/availability/rating).
- `GET /api/technicians/:id` → technician profile + reviews.
- `POST /api/bookings` → create booking (requires auth). Body: { userId, serviceId, preferredDate, preferredTime, address, phone, additionalNotes } → returns booking object & payment session link.
- `GET /api/bookings/:id` → booking details (auth).
- `PATCH /api/bookings/:id/status` → technician/admin update status (Pending → Accepted → En route → Arrived → Completed → Cancelled).
- `POST /api/bookings/:id/payment-intent` → create Stripe payment intent or session; confirm payment server-side.
- `POST /webhooks/stripe` → verify webhook events (payment succeeded).
- `POST /api/reviews` → post a completed-job review.
- `POST /api/contact` → contact/inquiry saving + admin email.

2. Data models (Mongoose schemas). Provide clear schemas:

- **User**: `{ _id, name, email, passwordHash, phone, role: 'user'|'technician'|'admin', address, createdAt, updatedAt, isVerified, avatarUrl }`
- **Technician**: `{ _id, userId (ref), skills: [String], bio, hourlyRate, ratingAvg, reviewsCount, availability: [{date, from, to}], location: {lat, lng}, isActive }`
- **Service**: `{ _id, name, category, description, images:[], basePrice, estimatedTime, tags, ratingAvg, isActive }`
- **Booking**: `{ _id, userId, serviceId, technicianId (nullable), date, time, address, price, status, paymentStatus, createdAt, updatedAt }`

- **Review**: `{ _id, bookingId, userId, technicianId, rating, comment, createdAt }`
- **Payment**: `{ _id, bookingId, stripePaymentId, amount, currency, status, createdAt }`
- **Inquiry**: `{ name, email, phone, message, createdAt }`

3. Security best practices

- Use bcrypt with a secure salt rounds (e.g., 12).
- Use JWT with short expiry (e.g., 1h) and refresh token strategy (optional).
- Validate inputs server-side with a validation library or custom checks; sanitize inputs.
- Use Helmet, CORS configured, rate limiting for auth endpoints, express-validator or Joi.
- Never expose secrets; use `.env` (provide `.env.example` with placeholders).

4. Real-time behavior

- Technician client (mobile/desktop) can emit `technician:location` events with `{technicianId, lat, lng}`.
- Server maps technician location to booking rooms (e.g., `booking_<bookingId>`) and emits `booking:location-update` to the booking's user(s).
- Booking status updates emit `booking:status-update` with new status and a short message; user dashboard listens and updates UI.
- Implement reconnect logic and basic presence detection for technicians.

5. Maps & tracking (frontend)

- Integrate **Mapbox** (or Google Maps if preferred) on user dashboard booking tracking; show tech location marker and ETA calculation (simple haversine/time estimate).
- Map component should gracefully degrade if API key missing (show text fallback & last-known location).

6. Payments

- Use Stripe Checkout or Payment Intents. Backend must create the session/intent and return client secret or sessionId.
- On successful payment, Stripe webhook marks `paymentStatus: 'paid'` and triggers booking confirmation (emit socket event + send SMS/email).

- Provide test-mode only keys in `.env.example`.

7. Notifications

- Use **Twilio** to send SMS at key events: booking created, technician accepted, technician arriving, booking completed.
- Also send email receipts via nodemailer (SMTP) for completed jobs.

11. Admin features

- Basic admin panel with protected routes (JWT + admin role) to manage users/services/bookings and view revenue summary.
- Simple charts on admin page (client-side chart library like Chart.js) showing monthly revenue and most requested services.

13.Code quality & comments

- Clean, modular server code (routes/controllers/services, bookings, auth).
- Easy to understand and simple easy for biggner level code
- Clear comments documenting important functions and flows.

Provide a concise README with: tech stack, env variables, setup steps, run commands, how to run seed, and how to test Stripe & Twilio integration (test mode).

```
• ├─ backend/
•    │     ├─ package.json
•    │     ├─ server.js (entry)
•    │     ├─ config/db_connection.js
•    │     ├─ controllers/
•    │     ├─ models/schemas/
•    │     ├─ routes/
•    │     ├─ sockets/
•    │     ├─ utils/
•    │     ├─ seed/
•    │     └─ .env.example
```

Project details - quickFix

Step for backend emplementatin:-