**Sequence Diagrams**

# 目录

Sequence Diagrams ........................ 2

1. Customer Booking a Service ........................ 3

2. Provider Submitting a New Business ........................ 4

3. Admin Approving a Business ........................ 5

4. User Authentication Flow ........................ 6

5. Role-Based Access Control Sequence ........................ 7

*[在Microsoft Word中，您可以右键点击上方目录区域并选择'更新域'来自动生成准确的页码]*

# **Sequence Diagrams**

This document provides Sequence Diagrams that illustrate the step-by-step interactions between users and various system components for key processes. These diagrams are based on an analysis of the codebase and use the correct, validated Mermaid syntax.

## **1. Customer Booking a Service**

This diagram shows the sequence of events when a customer books a service. The interaction flows from the customer’s browser to the Next.js server, which then interacts with the MongoDB database.

**mermaid**

sequenceDiagram

    actor Customer

    participant Browser

    participant Next.js Server

    participant MongoDB

    participant Email Service

    Customer->>Browser: Selects date and time, clicks 'Book'

    activate Browser

    Browser->>Next.js Server: POST /api/bookings

    activate Next.js Server

    Next.js Server->>Next.js Server: Validate booking data

    alt Validation successful

        Next.js Server->>MongoDB: `Booking.create(bookingData)`

        activate MongoDB

        MongoDB-->>Next.js Server: Returns created booking document

        deactivate MongoDB

        Next.js Server->>Email Service: Send booking confirmation

        activate Email Service

        Email Service-->>Next.js Server: Confirmation sent

        deactivate Email Service

        Next.js Server-->>Browser: API Response (200 OK)

        deactivate Next.js Server

        Browser->>Customer: Display 'Success' toast notification

    else Validation failed

        Next.js Server-->>Browser: API Response (400 Bad Request)

        deactivate Next.js Server

        Browser->>Customer: Display validation error message

    end

    deactivate Browser

**Description:**

1. The **Customer** initiates the booking from the BookingSection.jsx component in their **Browser**.

2. The browser sends a POST request to the **Next.js Server’s**/api/bookings route.

3. The server validates the booking data to ensure all required fields are present and valid.

4. If validation is successful, the server-side logic uses the Booking Mongoose model to create a new document in the **MongoDB** database.

5. MongoDB confirms the write operation and returns the new document.

6. The server sends booking confirmation emails to both the customer and service provider.

7. The success response is propagated back to the browser, which then shows a confirmation toast to the customer.

8. If validation fails, an error response is returned and displayed to the user.

## **2. Provider Submitting a New Business**

This diagram illustrates the workflow for a service provider submitting their new business registration, including document uploads to Cloudinary.

**mermaid**

sequenceDiagram

    actor Provider

    participant Browser

    participant Cloudinary

    participant Next.js Server

    participant MongoDB

    Provider->>Browser: Fills out ProviderRegistrationForm.jsx

    activate Browser

    loop For each required document

        Provider->>Browser: Uploads a file (e.g., ghana\_card.jpg)

        Browser->>Next.js Server: POST /api/cloudinary/upload

        activate Next.js Server

        Next.js Server->>Cloudinary: Upload API call with file

        activate Cloudinary

        Cloudinary-->>Next.js Server: Returns secure image URL

        deactivate Cloudinary

        Next.js Server-->>Browser: API Response (200 OK) with URL

        deactivate Next.js Server

        Browser->>Provider: Shows uploaded document in UI

    end

    Provider->>Browser: Clicks 'Submit Complete Registration'

    Browser->>Next.js Server: POST /api/businesses

    activate Next.js Server

    Next.js Server->>Next.js Server: Validate business data

    alt Validation successful

        Next.js Server->>MongoDB: `Business.create(businessData)`

        activate MongoDB

        note right of Next.js Server: Business is created with 'PENDING' approvalStatus

        MongoDB-->>Next.js Server: Returns created business document

        deactivate MongoDB

        Next.js Server-->>Browser: API Response (200 OK)

        deactivate Next.js Server

        Browser->>Provider: Display 'Registration Submitted' toast

    else Validation failed

        Next.js Server-->>Browser: API Response (400 Bad Request)

        deactivate Next.js Server

        Browser->>Provider: Display validation errors

    end

    deactivate Browser

**Description:**

1. The **Provider** fills out the multi-tab registration form in their **Browser**.

2. For each document, the browser makes a separate API call to the /api/cloudinary/upload endpoint on the **Next.js Server**.

3. The server securely uploads the file to **Cloudinary** and receives a URL in return.

4. Once all data is entered and documents are uploaded, the final submission triggers a POST request to the /api/businesses route.

5. The server validates the business data to ensure all required fields and documents are present.

6. The **Next.js Server** uses a Mongoose model to create a new business document in **MongoDB**, ensuring the new business is saved with a PENDING approval status, queuing it for admin review.

7. If validation fails, error messages are returned and displayed to the provider.

## **3. Admin Approving a Business**

This diagram shows the sequence for an administrator reviewing and approving a pending business application.

![](data:None;base64,)

*mermaid chart*

**Description:**

1. The **Admin** loads the approval dashboard in their **Browser**, which triggers a request to the **Next.js Server**.

2. The server queries the **MongoDB** database for all businesses where the approvalStatus is PENDING.

3. The list is displayed to the Admin. When the Admin clicks ‘Approve’, a POST request is sent to the /api/admin/approve-business endpoint.

4. The **Next.js Server** finds the relevant document in the **MongoDB** and updates its approvalStatus to APPROVED.

5. The server sends an email notification to the provider informing them of the approval.

6. Upon success, the browser notifies the Admin and refreshes the list, removing the approved business from the pending queue.

## **4. User Authentication Flow**

This diagram illustrates the authentication process when a user logs in to the system.

**mermaid**

sequenceDiagram

    actor User

    participant Browser

    participant Next.js Server

    participant NextAuth

    participant MongoDB

    participant JWT Service

    User->>Browser: Enters email/password and clicks 'Login'

    activate Browser

    Browser->>Next.js Server: POST /api/auth/signin

    activate Next.js Server

    Next.js Server->>NextAuth: Call authorize() function

    activate NextAuth

    NextAuth->>MongoDB: Database.healthCheck()

    activate MongoDB

    MongoDB-->>NextAuth: Connection status

    deactivate MongoDB

    alt Database connection healthy

        NextAuth->>MongoDB: User.validatePassword(email, password)

        activate MongoDB

        MongoDB-->>NextAuth: User document or null

        deactivate MongoDB

        alt Valid credentials

            NextAuth->>NextAuth: Check emailVerified flag

            alt Email verified

                NextAuth->>JWT Service: Create JWT with user role

                activate JWT Service

                JWT Service-->>NextAuth: JWT token

                deactivate JWT Service

                NextAuth-->>Next.js Server: Authentication success

                deactivate NextAuth

                Next.js Server-->>Browser: Session with user data and role

                deactivate Next.js Server

                Browser->>Browser: Redirect based on user role

                Browser->>User: Display authenticated UI

            else Email not verified

                NextAuth-->>Next.js Server: Email not verified error

                deactivate NextAuth

                Next.js Server-->>Browser: Authentication error

                deactivate Next.js Server

                Browser->>User: Display "Email not verified" message

            end

        else Invalid credentials

            NextAuth-->>Next.js Server: Invalid credentials error

            deactivate NextAuth

            Next.js Server-->>Browser: Authentication error

            deactivate Next.js Server

            Browser->>User: Display "Invalid email or password" message

        end

    else Database connection failed

        NextAuth-->>Next.js Server: Database connection error

        deactivate NextAuth

        Next.js Server-->>Browser: Authentication error

        deactivate Next.js Server

        Browser->>User: Display "Service unavailable" message

    end

    deactivate Browser

**Description:**

1. The **User** enters their email and password in the login form and submits it.

2. The **Browser** sends a POST request to the NextAuth signin endpoint.

3. The **Next.js Server** passes the request to **NextAuth** which handles the authentication process.

4. NextAuth first checks if the database connection is healthy using the Database.healthCheck() method.

5. If the connection is healthy, NextAuth validates the user credentials against the database using the User.validatePassword() method.

6. If the credentials are valid, NextAuth checks if the user’s email is verified.

7. If the email is verified, NextAuth creates a JWT token with the user’s role and returns a successful authentication response.

8. The browser receives the session data, redirects the user based on their role, and displays the authenticated UI.

9. If any step fails (database connection, invalid credentials, or unverified email), an appropriate error message is displayed to the user.

## **5. Role-Based Access Control Sequence**

This diagram shows how the system enforces role-based access restrictions when a user attempts to access a protected route.

**mermaid**

sequenceDiagram

    actor User

    participant Browser

    participant Next.js Middleware

    participant Protected Route

    participant UI Components

    User->>Browser: Attempts to access protected route

    activate Browser

    Browser->>Next.js Middleware: Request with JWT token

    activate Next.js Middleware

    Next.js Middleware->>Next.js Middleware: Extract token from request

    Next.js Middleware->>Next.js Middleware: Validate token

    alt Token valid

        Next.js Middleware->>Next.js Middleware: Extract user role

        alt Route requires specific role

            Next.js Middleware->>Next.js Middleware: Check if user has required role

            alt User has required role

                Next.js Middleware->>Protected Route: Allow access

                activate Protected Route

                Protected Route->>UI Components: Render with role-specific elements

                activate UI Components

                UI Components-->>Browser: Return rendered page

                deactivate UI Components

                deactivate Protected Route

                Browser->>User: Display protected content

            else User lacks required role

                Next.js Middleware->>Browser: Redirect to error page

                deactivate Next.js Middleware

                Browser->>User: Display access denied message

            end

        else Route doesn't require specific role

            Next.js Middleware->>Protected Route: Allow access

            activate Protected Route

            Protected Route->>UI Components: Render with role-specific elements

            activate UI Components

            UI Components-->>Browser: Return rendered page

            deactivate UI Components

            deactivate Protected Route

            deactivate Next.js Middleware

            Browser->>User: Display protected content

        end

    else Token invalid or missing

        Next.js Middleware->>Browser: Redirect to login page

        deactivate Next.js Middleware

        Browser->>User: Display login page

    end

    deactivate Browser

**Description:**

1. The **User** attempts to access a protected route in the application.

2. The **Browser** sends the request with the JWT token to the **Next.js Middleware**.

3. The middleware extracts and validates the token from the request.

4. If the token is valid, the middleware extracts the user’s role and checks if the route requires a specific role.

5. If the route requires a specific role, the middleware checks if the user has that role.

6. If the user has the required role, the middleware allows access to the protected route.

7. The protected route renders with role-specific UI elements based on the user’s role.

8. If the user lacks the required role, they are redirected to an error page with an access denied message.

9. If the token is invalid or missing, the user is redirected to the login page.

10. This implementation is handled primarily in the middleware.js file and the various layout components that check for role-based access.