Hackathon Day 2 Task 2

Technical Planning Documentation

Overview

This Document outline the technical planning for developing an E-Commerce Marketplace to empower small business and individuals by providing a platform to buy the seating products

Components and Role

Front-End (NextJs):

• User Interface
• API Communication to fetch and display data

Sanity CMS:

• Manage content such as products, User details and order History

Third Party APIs:

• Provide additional functionalities such as shipment tracking and payment integration

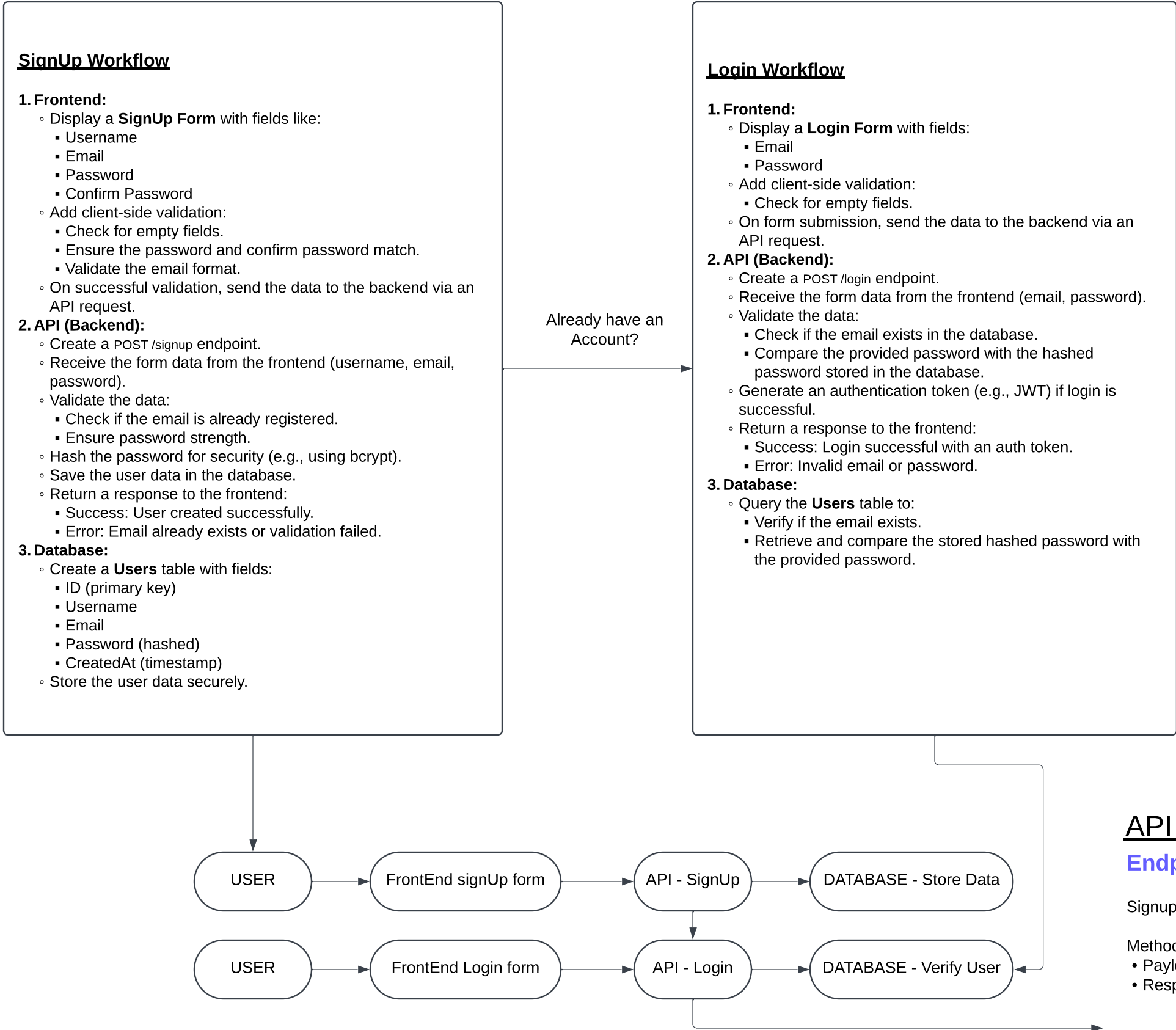### Workflow

1. User Journey

a. Visitor Journey:
• **Landing Page**: User visits the homepage.
    ◦ Highlights: Featured cameras, categories, promotions.
    ◦ Call-to-Action (CTA): "Explore Cameras," "Sign Up."
• **Browse Inventory**:
    ◦ Search functionality (keywords, filters for type, brand, price, location).
    ◦ Product listing with thumbnails and quick details.
b. Customer Journey:
• **Product Selection:**
    ◦ Click on a camera for detailed information.
    ◦ Details: Price per day, specifications, availability calendar, reviews, and ratings.
    ◦ Option to add items to the cart.
• **User Authentication**:
    ◦ **New Users**: Register via email, social accounts, or phone number.
    ◦ **Returning Users**: Login functionality.
• **Rental Request**:
    ◦ Select rental dates and confirm availability.
    ◦ Choose delivery/pickup options.
c. Payment & Confirmation:
• **Cart Overview**:
    ◦ List of items, rental period, price breakdown, and delivery charges.
• **Payment Gateway**:
    ◦ Integration with credit/debit cards, and local payment options.
• **Order Confirmation**:
    ◦ Unique rental order ID.
    ◦ Receipt and rental details emailed to the user.
d. Post-Purchase:
• **Order Tracking**:
    ◦ Real-time tracking for delivery.
• **Rental Period Management**:
    ◦ Notifications: Rental start, reminders for return date.
• **Returns**:
    ◦ Scheduled pickup or user drop-off.
• **Feedback and Reviews**:
    ◦ Option to rate the service and the product.
2. Admin Workflow

a. Product Management:
• Add/Edit/Delete camera listings.
• Manage categories, brands, and stock.
• Set promotional offers and discounts.
b. Order Management:
• View and update order status.
• Resolve conflicts (e.g., late returns, damages).
c. Customer Management:
• Access user profiles and history.
• Respond to inquiries and complaints.
d. Reporting and Analytics:
• Monitor revenue, popular items, and user engagement.
• Generate daily, weekly, and monthly reports.
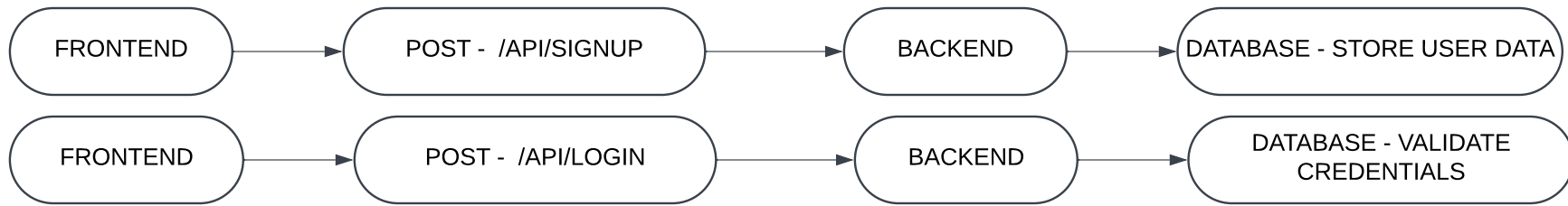
Technical Schema

1. Frontend
• **Technologies**: TypeScript, Next.js, Tailwind CSS for styling.
• **Features**:
    ◦ Responsive design for mobile and desktop.
    ◦ Dynamic forms for user registration, login, and checkout.
    ◦ Calendar component for date selection.
2. Backend
• **Technologies**: Next.js API routes for server-side logic, integrated with Sanity CMS.
• **Sanity CMS**:
    ◦ Store and manage camera listings, categories, and promotional content.
    ◦ Schema definitions for products, orders, and users.
• **Database**: Sanity's GROQ for querying content.
3. APIs
• RESTful APIs built using Next.js API routes:
    ◦ **Auth APIs**: Login, Register, Password Recovery.
    ◦ **Product APIs**: Fetch list, filter, product details.
    ◦ **Order APIs**: Create order, update status, fetch history.
    ◦ **Payment APIs**: Integration with Stripe or PayPal.
4. Third-Party Integrations
• **Payment Gateways**: Stripe
• **Notification Services**: Twilio/SendGrid for SMS and email notifications.
• **Maps API**: Google Maps for delivery tracking.
5. Cloud Infrastructure
• **Hosting**: Vercel for Next.js application and API routes.
• **CDN**: Vercel Edge for caching and speed optimization.
• **Storage**: Sanity's asset management for product images and media.
6. Security Features
• **Authentication**:
    ◦ JWT for token-based authentication.
    ◦ OAuth2 for social logins.
• **Encryption**:
    ◦ SSL for secure communication.
    ◦ Passwords stored as hashes (bcrypt).
• **Fraud Prevention**:
    ◦ Limit rental requests from flagged users.
    ◦ Monitor for suspicious activity.
7. Monitoring and Maintenance
• **Monitoring Tools**: Vercel Analytics for performance and usage tracking.
• **Error Tracking**: Sentry for logging and managing errors.
• **Regular Updates**:
    ◦ Schema updates in Sanity CMS for new fields or features.
    ◦ Incremental static regeneration (ISR) in Next.js for content updates.
    ◦ Backup Sanity CMS content regularly.

SignUp Workflow

1. Frontend:
    ◦ Display a **SignUp Form** with fields like:
        ▪ Username
        ▪ Email
        ▪ Password
        ▪ Confirm Password
    ◦ Add client-side validation:
        ▪ Check for empty fields.
        ▪ Ensure the password and confirm password match.
        ▪ Validate the email format.
    ◦ On successful validation, send the data to the backend via an API request.
2. API (Backend):
    ◦ Create a POST /signup endpoint.
    ◦ Receive the form data from the frontend (username, email, password).
    ◦ Validate the data:
        ▪ Check if the email is already registered.
        ▪ Ensure password strength.
    ◦ Hash the password for security (e.g., using bcrypt).
    ◦ Save the user data in the database.
    ◦ Return a response to the frontend:
        ▪ Success: User created successfully.
        ▪ Error: Email already exists or validation failed.
3. Database:
    ◦ Create a **Users** table with fields:
        ▪ ID (primary key)
        ▪ Username
        ▪ Email
        ▪ Password (hashed)
        ▪ CreatedAt (timestamp)
    ◦ Store the user data securely.

Already have an Account?

Login Workflow

1. Frontend:
    ◦ Display a **Login Form** with fields:
        ▪ Email
        ▪ Password
    ◦ Add client-side validation:
        ▪ Check for empty fields.
    ◦ On form submission, send the data to the backend via an API request.
2. API (Backend):
    ◦ Create a POST /login endpoint.
    ◦ Receive the form data from the frontend (email, password).
    ◦ Validate the data:
        ▪ Check if the email exists in the database.
        ▪ Compare the provided password with the hashed password stored in the database.
    ◦ Generate an authentication token (e.g., JWT) if login is successful.
    ◦ Return a response to the frontend:
        ▪ Success: Login successful with an auth token.
        ▪ Error: Invalid email or password.
3. Database:
    ◦ Query the **Users** table to:
        ▪ Verify if the email exists.
        ▪ Retrieve and compare the stored hashed password with the provided password.

USER → FrontEnd signUp form → API - SignUp → DATABASE - Store Data

USER → FrontEnd Login form → API - Login → DATABASE - Verify User

API Workflow

Endpoints

Signup API : /api/signup

Method: POST
• Payload: {name, email, password}
• Response: { "sucess": true,
                 "UserId": "12345"}

Login API: /api/login

Method: POST
• Payload: {email, Password}
• Response. {"Sucess": true,
                 "token". "abcd 1234"}

FRONTEND → POST - /API/SIGNUP → BACKEND → DATABASE - STORE USER DATA

FRONTEND → POST - /API/LOGIN → BACKEND → DATABASE - VALIDATE CREDENTIALS