Day 2: Technical Requirements for Morent

Overview

This document outlines the technical requirements and foundational structure for Morent, ensuring the platform is scalable, maintainable, and secure. The chosen stack includes:

- **Frontend**: Next.js (TypeScript)
- **CMS**: Sanity CMS (including assets and database)
- **Deployment**: Vercel
- **Styling**: Tailwind CSS with ShadCN UI components
- **Additional Tools**: Your preferred modern JavaScript libraries and utilities.

System Architecture

High-Level Diagram

The architecture integrates:

1. **Frontend**: Next.js for SSR and CSR.
2. **Backend**: API routes managed in Next.js for lightweight backend processes.
3. **CMS**: Sanity CMS for managing dynamic content like vehicle listings, user reviews, and blogs.
4. **Database**: Sanity's built-in document store for data persistence.
5. **Hosting**: Vercel for optimized and globally distributed deployments.
6. **Styling**: Tailwind CSS for rapid design and ShadCN UI for pre-designed components.

Workflow Details

1. **User Registration & Authentication**:
    - **Role-Based Access**: Users register as renters or vehicle owners.
    - Use Clerk or similar service for authentication.
2. **Vehicle Listings**:
    - Owners can create, edit, and delete listings.
    - Sanity CMS to store vehicle details.
3. **Booking Process**:
    - Users browse and book available vehicles.
    - Real-time availability check using Sanity CMS.
    - Payment processed via Stripe or a local gateway like JazzCash.
4. **Reviews & Ratings**:
    - Users leave reviews linked to specific vehicle IDs.

    - Moderated through the admin panel in Sanity CMS.
5. **Admin Panel**:
    - Manage users, vehicles, transactions, and content.
    - Built with ShadCN UI components.

Technical Breakdown

Frontend

1. **Next.js**:
    - **Routing**: Dynamic routes for vehicles, user profiles, and booking.
    - **SSR/ISR**: Improve SEO and performance.
2. **TypeScript**:
    - Ensure type safety across the application.
3. **Tailwind CSS & ShadCN UI**:
    - Rapidly create responsive and accessible designs.

CMS & Database

1. **Sanity CMS**:
    - Store entities such as users, vehicles, bookings, and reviews.
    - Use Sanity's GROQ for querying data.
2. **Sanity Assets**:
    - Manage images and documents related to vehicle listings.

Deployment

1. **Vercel**:
    - CI/CD pipelines for seamless deployment.
    - Edge caching for low-latency content delivery.

Third-Party Integrations

1. **Payment Gateway**:
    - Integrate Stripe or JazzCash for transactions.
2. **Maps API**:
    - Use Google Maps for location-based search and directions.
3. **Authentication**:
    - Implement role-based authentication using Clerk or similar services.

API Requirements

- **Authentication**:
    - POST `/api/auth/login`
    - POST `/api/auth/register`

- **Vehicle Management**:
  - GET `/api/vehicles`
  - POST `/api/vehicles`
  - PATCH `/api/vehicles/:id`
  - DELETE `/api/vehicles/:id`
- **Bookings**:
  - GET `/api/bookings`
  - POST `/api/bookings`
  - PATCH `/api/bookings/:id`
  - DELETE `/api/bookings/:id`
- **Reviews**:
  - GET `/api/reviews`
  - POST `/api/reviews`
  - PATCH `/api/reviews/:id`
  - DELETE `/api/reviews/:id`

Schema Drafts

Vehicle Schema:

```
export const vehicleSchema = {
  name: 'vehicle',
  type: 'document',
  fields: [
    { name: 'title', type: 'string' },
    { name: 'owner', type: 'reference', to: [{ type: 'user' }] },
    { name: 'type', type: 'string' },
    { name: 'pricePerDay', type: 'number' },
    { name: 'availability', type: 'boolean' },
    { name: 'location', type: 'string' },
    { name: 'images', type: 'array', of: [{ type: 'image' }] },
  ],
```

1. };

Booking Schema:

```
export const bookingSchema = {
  name: 'booking',
  type: 'document',
  fields: [
    { name: 'vehicle', type: 'reference', to: [{ type: 'vehicle' }] },
    { name: 'renter', type: 'reference', to: [{ type: 'user' }] },
    { name: 'startDate', type: 'datetime' },
    { name: 'endDate', type: 'datetime' },
    { name: 'totalPrice', type: 'number' },
  ],
```

2. };

Additional Considerations

- **Caching**: Use Redis for caching high-traffic API endpoints.
- **Testing**: Unit and integration tests using Jest and Cypress.
- **Monitoring**: Set up logging and monitoring with tools like LogRocket and Sentry.

Deliverables

1. Finalized system architecture diagram.
2. Draft Sanity schemas for all key entities.
3. API documentation with sample requests and responses.
4. Deployment pipeline set up on Vercel.

Next Steps

- Build out the base project structure in Next.js.
- Integrate Sanity CMS and configure schemas.
- Implement authentication and role management.