# Marketplace Builder Hackathon 2025

## Day 2 : Planning The Technical Foundation

This document outlines the technical framework and requirements for the development of the marketplace platform, **MORENT**. It provides a comprehensive overview of the project's core aspects, including:

1. **System Architecture**: A detailed breakdown of the system's design and components.
2. **API Endpoints:** A specification of the API endpoints required for seamless communication between the frontend and the backend (Sanity CMS).
3. **Frontend Requirements**: A list of the tools, frameworks, and design standards to be employed on the client side.
4. **Backend Requirements**: A detailed specification of the server-side technologies, databases, and infrastructure.
5. **Workflow Diagram**: A visual representation of the system's operational flow, highlighting interactions between components.
6. **Data Schema**: A structured representation of the data model and its relationships.
7. **Project Roadmap**: A step-by-step timeline for the successful completion of the marketplace.

This document aims to serve as a clear guide for the systematic execution of the MORENT marketplace project, ensuring clarity and alignment across all technical aspects.

## System Architecture

- **Overview**: The system architecture provides a clear depiction of how various components interact to deliver an efficient and seamless marketplace experience. Key components include:
  - **Frontend (Next.js)**: Responsible for the user interface, including browsing cars data, managing the cart, and processing checkouts. Next.js leverages server-side rendering to enhance performance and SEO.
  - **Sanity CMS**: Acts as the centralized backend system, managing cars data, customer profiles, and order records. It provides a scalable and flexible schema design for business needs.
  - **Third-Party APIs**: Ensures critical functionalities such as real-time inventory updates, and secure payment processing are integrated seamlessly.

## Frontend Requirements

- **User Interface**:
  - Designed a user-friendly, intuitive interface tailored for both mobile and desktop users.

    - The platform is responsive, providing consistent performance across various devices and screen sizes.
- **Essential Pages**:
    - **Home Page**: Serves as the gateway to the marketplace with featured Cars, categories, and promotional banners.
    - **Car Listing Page**: Displays a catalog of cars with filtering and sorting options to enhance the browsing experience.
    - **Car Details Page**: Provides comprehensive information about individual Cars, including images, descriptions, specifications, and reviews.
    - **Checkout Page**: Streamlines the purchasing process with secure payment options and user-friendly forms for billing and shipping details.
    - **Order Confirmation Page**: Confirms successful transactions and provides order details.
- **Framework**:
    - Utilized **Next.js** for its advanced server-side rendering capabilities, which improve performance and search engine optimization (SEO).
    - Leverage reusable components and modular architecture to simplify development and ensure scalability.
    - Implemented dynamic routing to support a seamless navigation experience.

This structured approach ensures the frontend aligns with user expectations while leveraging modern technologies to enhance performance and maintainability.

## ***Backend Requirements***

- **Sanity CMS**:
    - Serves as the primary database for managing essential marketplace data, including cars, customer details, and order records.
    - Provides a scalable and flexible content management solution tailored to the marketplace's unique business needs.
    - Schemas are meticulously designed to align with business objectives and ensure data integrity.
- **Third-Party Integrations**:
    - Leveraged third-party APIs for seamless integration of key functionalities, including:
        - **Delivery Services**: Real-time shipment tracking and delivery status updates.
        - **Payment Gateways**: Secure and efficient processing of online transactions, supporting multiple payment methods.
    - APIs are robust, reliable, and capable of handling high traffic without latency.
- **Performance and Security**:
    - Optimize database queries and API calls to minimize response times and enhance user experience.
    - Implemented advanced security measures.
- **Scalability and Maintainability**:
    - Design backend architecture to accommodate future growth, ensuring it can handle increased traffic and data.

- Used modular and reusable code structures to simplify maintenance and updates.

This approach ensures a robust backend infrastructure capable of supporting a high-performance and secure marketplace platform.

## Data Schema

### Car Schema:

```
export const cars = defineType(
  {
    name:"cars",
    title: "Cars",
    type: "document",
    fields: [
      defineField({
        name:"name",
        title: "Car Name",
        type: "string"
      }),
      defineField({
        name:"rent",
        title: "Rent Price",
        type: "number"
      }),
      defineField({
        name:"description",
        title: "Car Description",
        type: "text"
      }),
      defineField({
        name:"stock",
        title: "Availiable Cars",
        type: "number"
      }),
      defineField({
        name:"image",
        title: "Car Images",
        type: "array",
        of: [{
          type: "image",
          options: {
            hotspot: true
          },
        }]
      }),
      defineField({
        name:"category",
        title: "Category",
        type: "string",
      }),
      defineField({
        name: "reviews",
        title: "Reviews",
          type: "array",
          of: [
            {
              type: "object",
              title: "Review",
              fields: [
                {
                  name: "avatar",
                  title: "Avatar",
                  type: "image",
                  options: {
                    hotspot: true,
                  },
                },
                {
                  name: "name",
                  title: "Name",
                  type: "string",
                },
                {
                  name: "review",
                  title: "Review",
                  type: "text",
                },
              ],
          },
        ],
      })

    ]
  }
)
```

### Order Schema:

```
export const order = defineType({
  name: "order",
  title: "Orders",
  type: "document",
  fields: [
    defineField({
      name: "customer_info",
      title: "Customer Information",
      type: "reference",
      to: [{ type: "customer" }],
    }),
    defineField({
      name: "car",
      title: "Car Details",
      type: "reference",
      to: [{ type: "cars" }],
    }),
    defineField({
      name: "order_status",
      title: "Order Status",
      type: "string",
      options: {
        list: [
          { title: "Pending", value: "pending" },
          { title: "Completed", value: "completed" },
          { title: "Canceled", value: "canceled" },
        ],
      },
    }),
    defineField({
      name: "order_date",
      title: "Order Date",
      type: "datetime",
    }),
    defineField({
      name: "price",
      title: "Order Price",
      type: "number",
      validation: (Rule) => Rule.min(0).precision(2),
    }),
  ],
});
```

## Customer Schema:

```
export const customer = defineType({
  name: "customer_info",
  title: "Customer Information",
  type: "document",
  fields: [
    defineField({
      name: "name",
      title: "Customer Name",
      type: "string"
    }),
    defineField({
      name: "email",
      title: "Customer Email",
      type: "email"
    }),
    defineField({
      name: "phone",
      title: "Customer Phone',
      type: "number"
    }),
  ]
})
```

## Payment Schema:

```
export const payment = defineType({
 name: "payment",
 title: "Payments",
 type: "document",
 fields: [
   defineField({
     name: "order",
     title: "Order",
     type: "reference",
     to: [{ type: "order" }],
   }),
   defineField({
     name: "paymentMethod",
     title: "Payment Method",
     type: "string",
     options: {
       list: [
         { title: "Credit Card", value: "credit_card" },
         { title: "PayPal", value: "paypal" },
         { title: "Bank Transfer", value: "bank_transfer" },
       ],
       layout: "radio",
     },
     description: "Payment method used by the customer",
   }),
   defineField({
     name: "amount",
     title: "Payment Amount",
     type: "number",
     validation: (Rule) => Rule.min(0).precision(2).required(),
   }),
   defineField({
     name: "currency",
     title: "Currency",
     type: "string",
     options: {
       list: [
         { title: "USD", value: "USD" },
         { title: "EUR", value: "EUR" },
         { title: "GBP", value: "GBP" },
       ],
       layout: "dropdown",
     },
     initialValue: "USD",
   }),
   defineField({
     name: "paymentStatus",
     title: "Payment Status",
     type: "string",
     options: {
        list: [
          { title: "Pending", value: "pending" },
          { title: "Completed", value: "completed" },
          { title: "Failed", value: "failed" },
          { title: "Refunded", value: "refunded" },
        ],
      },
    }),
    defineField({
      name: "transactionId",
      title: "Transaction ID",
      type: "string",
    }),
    defineField({
      name: "paymentDate",
      title: "Payment Date",
      type: "datetime",
    })
  ],
});
```

## *API Endpoints*

API endpoints based on the marketplace workflows. Examples include:

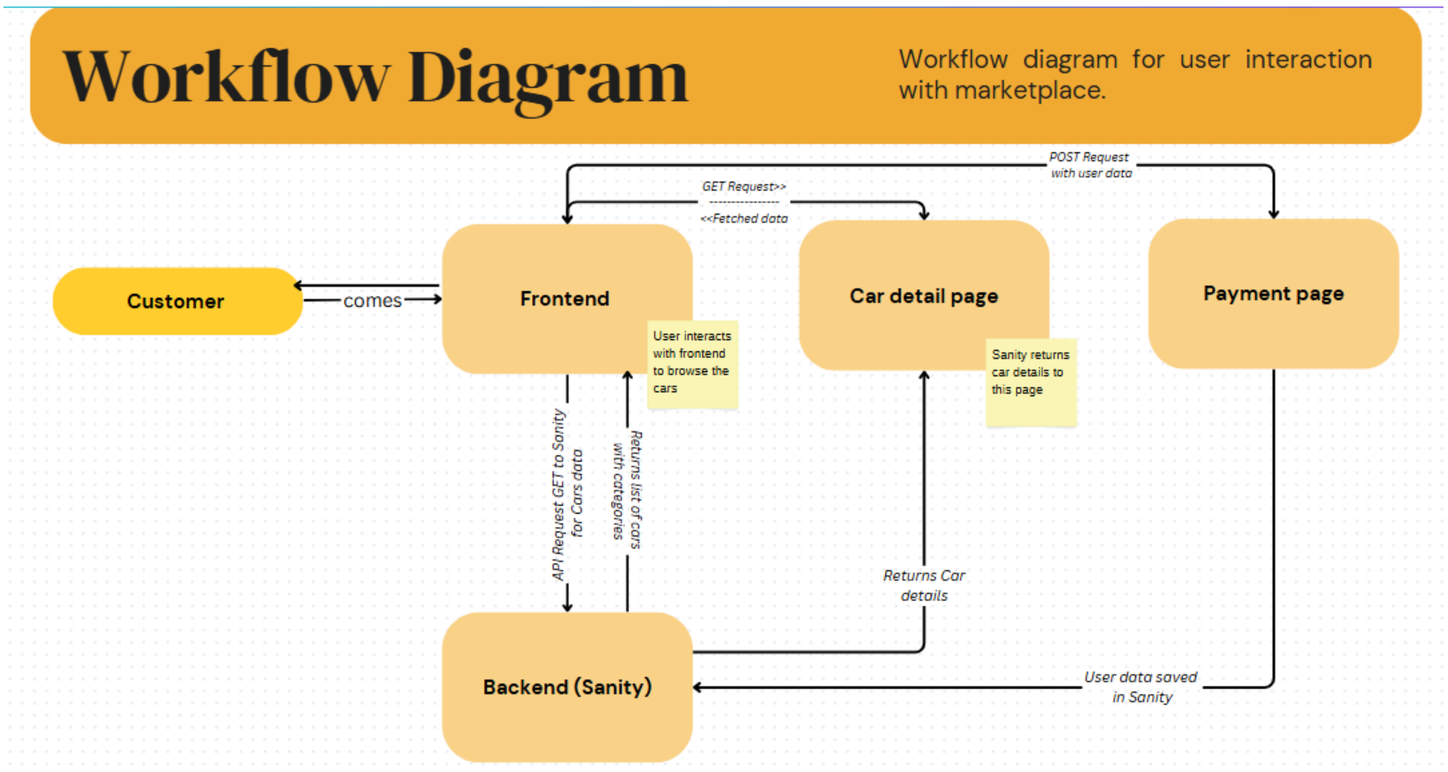
**Rental E-Commerce**:

- **Endpoint**: /car
  - **Method**: GET

    - **Description**: Fetch all available cars from Sanity CMS.
    - **Response**:
    - [
        - {
        -   "id": 1,
        -   "name": "car A",
        -   "rent": 100,
        -   "stock": 50,
        -   "image": "url-to-image"
        - }
        - {
        -   "id": 1,
        -   "name": "car A",
        -   "rent": 100,
        -   "stock": 50,
        -   "image": "url-to-image"
        - }
    - ]
- **Endpoint**: /car/id
    - **Method**: GET
    - **Description**: Fetch specific car data according to id from Sanity CMS.
    - **Response**:
    - {
    -   "id": 1,
    -   "name": "car A",
    -   "rent": 100,
    -   "stock": 50,
    -   "image": "url-to-image"
    - }
- **Endpoint**: /orders
    - **Method**: POST
    - **Description**: Create a new order in Sanity CMS.
    - **Payload**: Customer Info, car details, payment status.
- **Endpoint**: /orders/id
    - **Method**: GET
    - **Description**: Fetch a specific order from Sanity CMS.

## ***Workflow Diagram***

Visualize user interactions and system workflows. For example:


Workflow Diagram
Workflow diagram for user interaction with marketplace.
POST Request with user data
GET Request>>
<<Fetched data
Customer
comes
Frontend
Car detail page
Payment page
User interacts with frontend to browse the cars
Sanity returns car details to this page
API Request GET to Sanity for Cars data
Returns list of cars with categories
Returns Car details
User data saved in Sanity
Backend (Sanity)

- **Customer Interaction**:
  - The customer comes to the application and interacts with the frontend to browse cars.
- **Frontend Requests**:
  - The frontend makes an **API GET request** to the backend (Sanity) to fetch car data.
  - Backend responds with the list of cars, which is displayed on the frontend.
- **Car Detail Page**:
  - The customer selects a car, and the frontend fetches specific car details by sending another **GET request** to the backend.
  - Backend returns the requested car details, which are displayed on the **Car Detail Page**.
- **Payment Page**:
  - The customer proceeds to the **Payment Page**. A **POST request** is sent from the frontend to save user data in the backend.
  - The backend processes and stores the data in Sanity CMS.
- **Additional Notes**:
  - Real-time updates are ensured through API calls.
  - User data, including car details and transactions, is stored securely in the backend.

## Project Roadmap

A step-by-step plan to ensure timely project delivery:

1. **Day 1**: Finalize system architecture and data schema.
2. **Day 2**: Develop essential frontend components and pages.
3. **Day 3**: Set up Sanity CMS with defined schemas.
4. **Day 4**: Integrate third-party APIs for shipment tracking and payments.

5. **Day 5**: Conduct end-to-end testing and resolve bugs.
6. **Day 6**: Deploy the marketplace and monitor performance.

This document is a detailed blueprint for building the MORENT marketplace, ensuring a clear, structured approach to achieving the project's objectives.