FoodTuck Project: Full Documentation (Days 1–7)

Overview

Food Tuck is a QCommerce platform designed to connect local restaurants and food vendors with customers, offering a seamless and secure online food ordering experience. Over the course of six days, the project progressed from conceptualizing ideas to successfully deploying a staging environment. Each day focused on specific tasks that contributed to the overall development of the platform.

Day 1: Conceptualization and Marketplace Design

Key Achievements:

- Defined the marketplace as a QCommerce platform for food delivery.

Business Goals:

- Empower local restaurants and small food vendors to expand their reach.
- Provide customers with a convenient and efficient way to discover and order food online.

Data Schema Design:

- **Entities:** Food Items, Orders, Users, and Delivery Zones.
- **Relationships:**
  - Users place orders that reference specific food items.
  - Delivery zones are assigned to riders for order fulfillment.

Day 2: Technical Planning

Key Achievements:

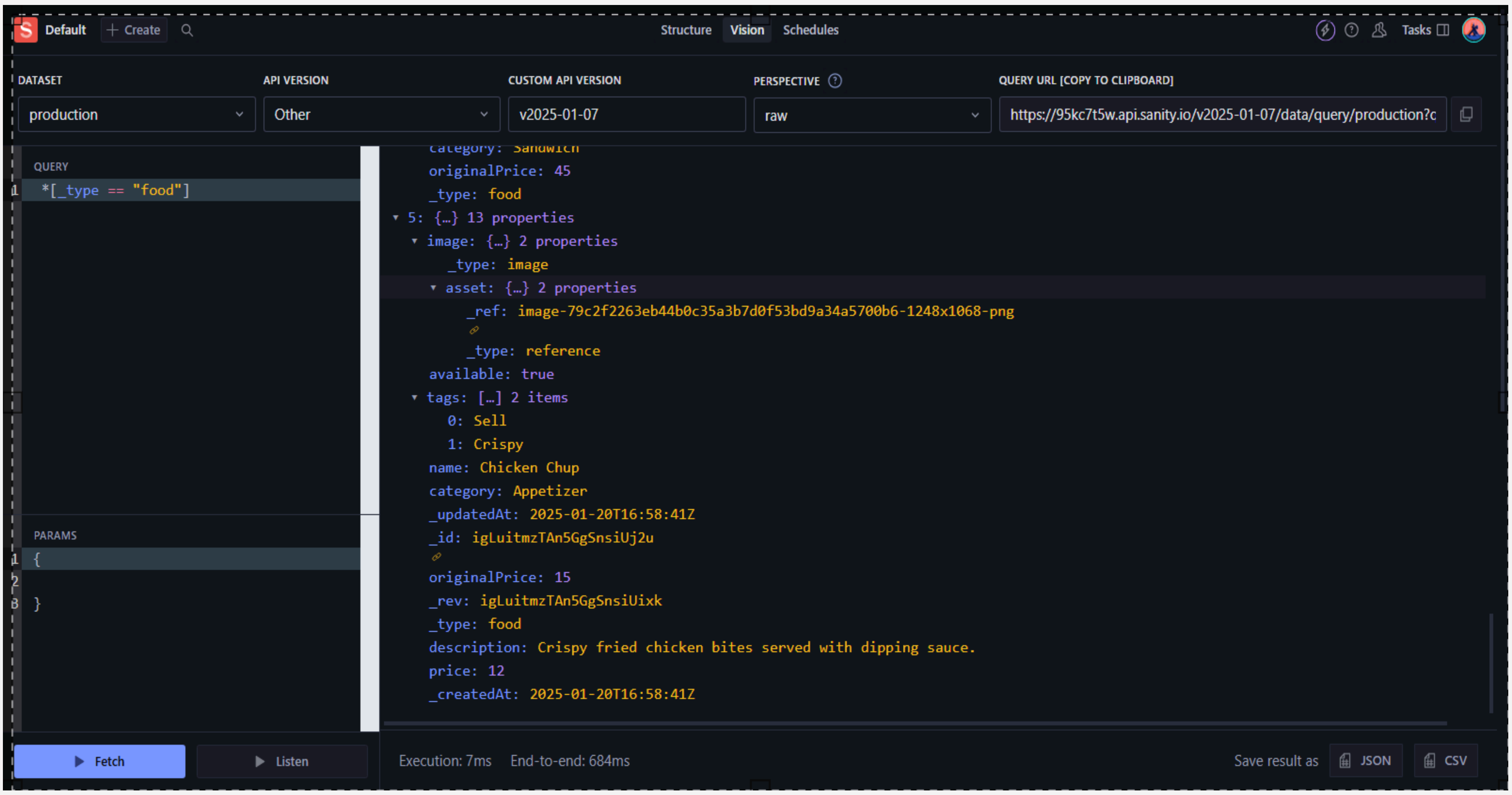
- **Tech Stack:**
  - **Frontend:** Next.js with Tailwind CSS for responsive and modern styling.
  - **Backend:** Sanity CMS for efficient content and product management.
  - **Database:** Firebase for real-time database management and secure user authentication.
  - **APIs:** ShipEngine for order tracking and Stripe for seamless payment integration.
- **API Requirements:**
  - **User Management:** `/register`, `/login`, and `/verify-route` for user authentication and account management.
  - **Product Management:** `/food-items`, `/food-item/:id` for adding, updating, and retrieving menu items.
  - **Orders:** `/orders` (POST) for placing orders and `/delivery/:id` (GET) for tracking order status.
- **Deployment Plan:**
  - **Frontend:** Deployed on Vercel for fast, reliable hosting.
  - **Backend:** Hosted on AWS Lambda using a serverless architecture for scalability and efficiency.

Day 3: Data Migration

Key Achievements:

- **Custom Migration Code:**
  - Data from Sanity CMS was migrated to Next.js using GROQ queries. .
  - Example GROQ Query: **`*[_type == "food"] {title, description, price, image}.`**
- **Schema Definition:**
  - Defined schema for **Food Items** with fields including:
    - **Title**: Name of the food item.
    - **Slug**: Unique identifier for each item.
    - **Description**: Detailed information about the dish.
    - **Price**: Cost of the food item.
    - **Image**: Visual representation of the dish.
- **Client Integration:**

Dynamically fetched and rendered food item data on the homepage using Next.js components, ensuring real-time updates and a smooth user experience.



DATASET: production
API VERSION: Other
CUSTOM API VERSION: v2025-01-07
PERSPECTIVE: raw
QUERY URL [COPY TO CLIPBOARD]: https://95kc7t5w.api.sanity.io/v2025-01-07/data/query/production?c

QUERY
```
*[_type == "food"]
```

PARAMS
```
{

}
```

```
    category: Sandwich
    originalPrice: 45
    _type: food
▾ 5: {…} 13 properties
  ▾ image: {…} 2 properties
      _type: image
    ▾ asset: {…} 2 properties
        _ref: image-79c2f2263eb44b0c35a3b7d0f53bd9a34a5700b6-1248x1068-png
        _type: reference
    available: true
  ▾ tags: […] 2 items
      0: Sell
      1: Crispy
    name: Chicken Chup
    category: Appetizer
    _updatedAt: 2025-01-20T16:58:41Z
    _id: igLuitmzTAn5GgSnsiUj2u
    originalPrice: 15
    _rev: igLuitmzTAn5GgSnsiUixk
    _type: food
    description: Crispy fried chicken bites served with dipping sauce.
    price: 12
    _createdAt: 2025-01-20T16:58:41Z
```

Execution: 7ms End-to-end: 684ms

Day 4: Building Dynamic Frontend Components

Key Achievements:

- **Dynamic Food Item Listings:**
  - Developed a `FoodList` component to dynamically display food items fetched from Firebase.
- **Filters and Sorting:**
  - Implemented filters for categories (e.g., appetizers, main course, desserts) and price ranges.
  - Added sorting options, including price (low to high, high to low) and popularity.
- **Reusable Components:**
  - **FoodCard:** Displayed food images, names, descriptions, and prices.
  - **FilterSidebar:** A sidebar to enable users to filter and sort food items seamlessly.
  - **PaginationControls:** Allowed navigation through large food item datasets efficiently.

Day 5: Testing and Backend Refinement

Key Achievements:

- **Testing Types:**
  - **Functional Testing:**
    - Ensured smooth workflows for food item listings, cart operations, and API integrations.
  - **Performance Testing:**
    - Analyzed page load speeds and responsiveness using Lighthouse for both desktop and mobile users.
  - **Security Testing:**
    - Validated user input fields, protected API keys, and ensured HTTPS implementation for secure communication.
- **CSV-Based Testing Report:**
  - Documented test cases, including scenarios, expected outcomes, actual results, and resolution statuses.
  - Used a structured table format to track all testing results efficiently.

• CSV-Based Testing Report: Test Case Table

Test Case ID	Test Case Description	Test Steps	ExpectedResult	Actual Result	Status	Severity level	Assigned To	Remarks
TC001	Validate product listing page	Open product page > Verify product	Products displayed correctly	Products displayed correctly	Passed	Low	-	Product listing is accurate and user-friendly.
TC002	Test API error handling	Disconnect API > Refresh page	Show fallback UI with error message	Error message shown	Passed	Medium	-	Error handling works seamlessly during failures.
TC003	Check cart functionality	Add product to cart > Verify cart contents	Cart updates with added product	Cart updates as expected	Passed	High	-	Cart functionality is robust and intuitive.
TC004	Ensure Responsiveness on mobile	Resize browser window > Check layout	Layout adjusts properly to screen size	Responsive layout working as intended	Passed	Medium	-	Design ensures consistency across devices.

Csv Content

Test Case ID,Test Case ID,Test Steps,ExpectedResult,Actual Result,Status,Severity level,Assigned To,Remarks

TC001,Validate product listing page,Open product page > Verify product,Products displayed correctly,Products displayed correctly,Passed,Low,Wajahat Ali,Product listing is accurate and user-friendly.

TC002,Test API error handling,Disconnect API > Refresh page,Show fallback UI with error message,Error message shown,Passed,Medium,Wajahat Ali,Error handling works seamlessly during failures.

TC003,Check cart functionality,Add product to cart > Verify cart contents,Cart updates with added product,Cart updates as expected,Passed,High,Wajahat Ali,Cart functionality is robust and intuitive.

TC004,Ensure Responsiveness on mobile,Resize browser window > Check layout,Layout adjusts properly to screen size,Responsive layout working as intended,Passed,Medium,Wajahat Ali,Design ensures consistency across devices.

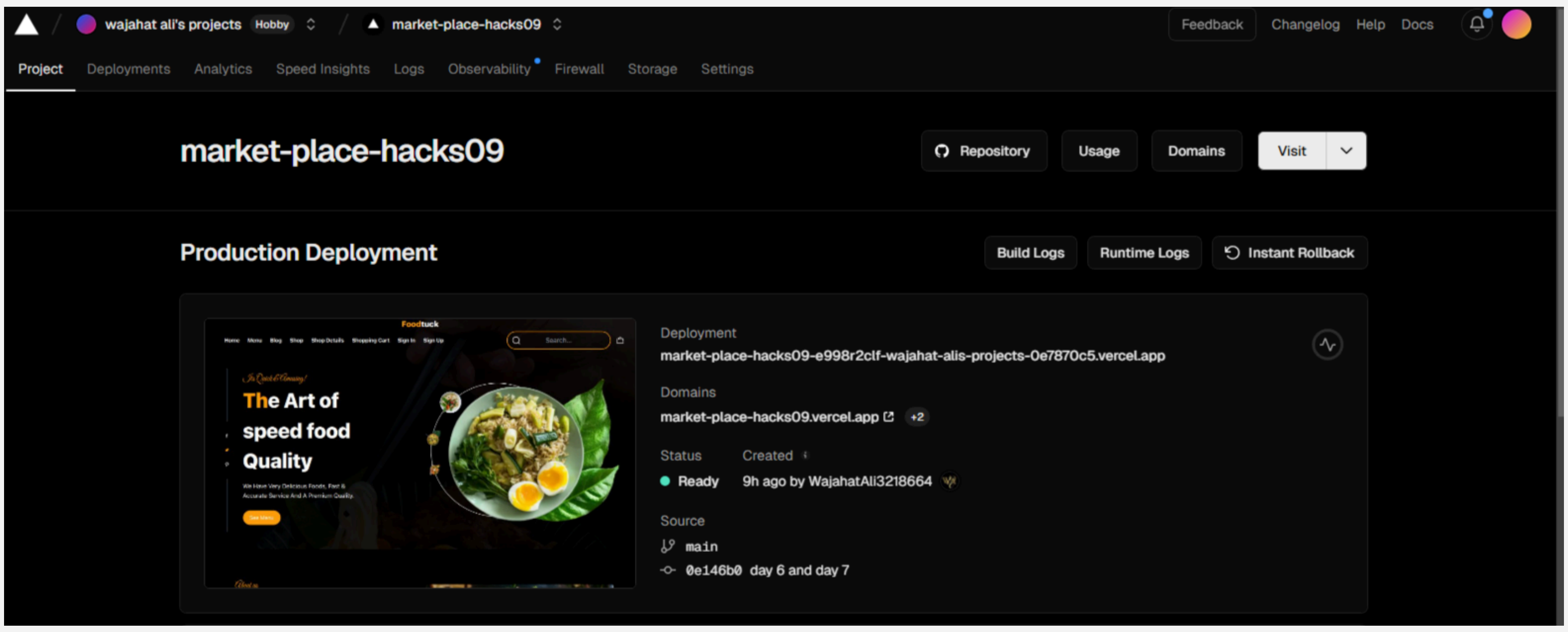
Day 6: Deployment Preparation and Staging Environment Setup

Key Achievements:

- **Deployment Strategy:**
    - Deployed the FoodTuck application on Vercel for fast and reliable hosting.
    - Integrated the GitHub repository to enable Continuous Integration/Continuous Deployment (CI/CD).
- **Environment Variables:**
    - Configured sensitive variables, such as API keys, in a `.env` file for secure handling.
    - Uploaded the `.env` file securely to Vercel for deployment.

Example `.env` File:
`NEXT_PUBLIC_SANITY_PROJECT_ID`=your_project_id
`NEXT_PUBLIC_SANITY_DATASET`=production
`SANITY_API_TOKEN`=your_api_key

-
- **Staging Environment:**
    - Deployed a staging build to simulate a production-like environment and validate app functionality.
- **Staging Testing:**
    - **Functional Testing:** Verified key workflows, including food item listings, cart operations, and checkout.
    - **Performance Testing:** Used GTmetrix to analyze loading speeds, responsiveness, and user experience.
    - **Security Testing:** Ensured HTTPS implementation, input validation, and secure API calls.



market-place-hacks09

Production Deployment

Deployment
market-place-hacks09-e998r2clf-wajahat-alis-projects-0e7870c5.vercel.app

Domains
market-place-hacks09.vercel.app +2

Status: Ready
Created: 9h ago by WajahatAli3218664

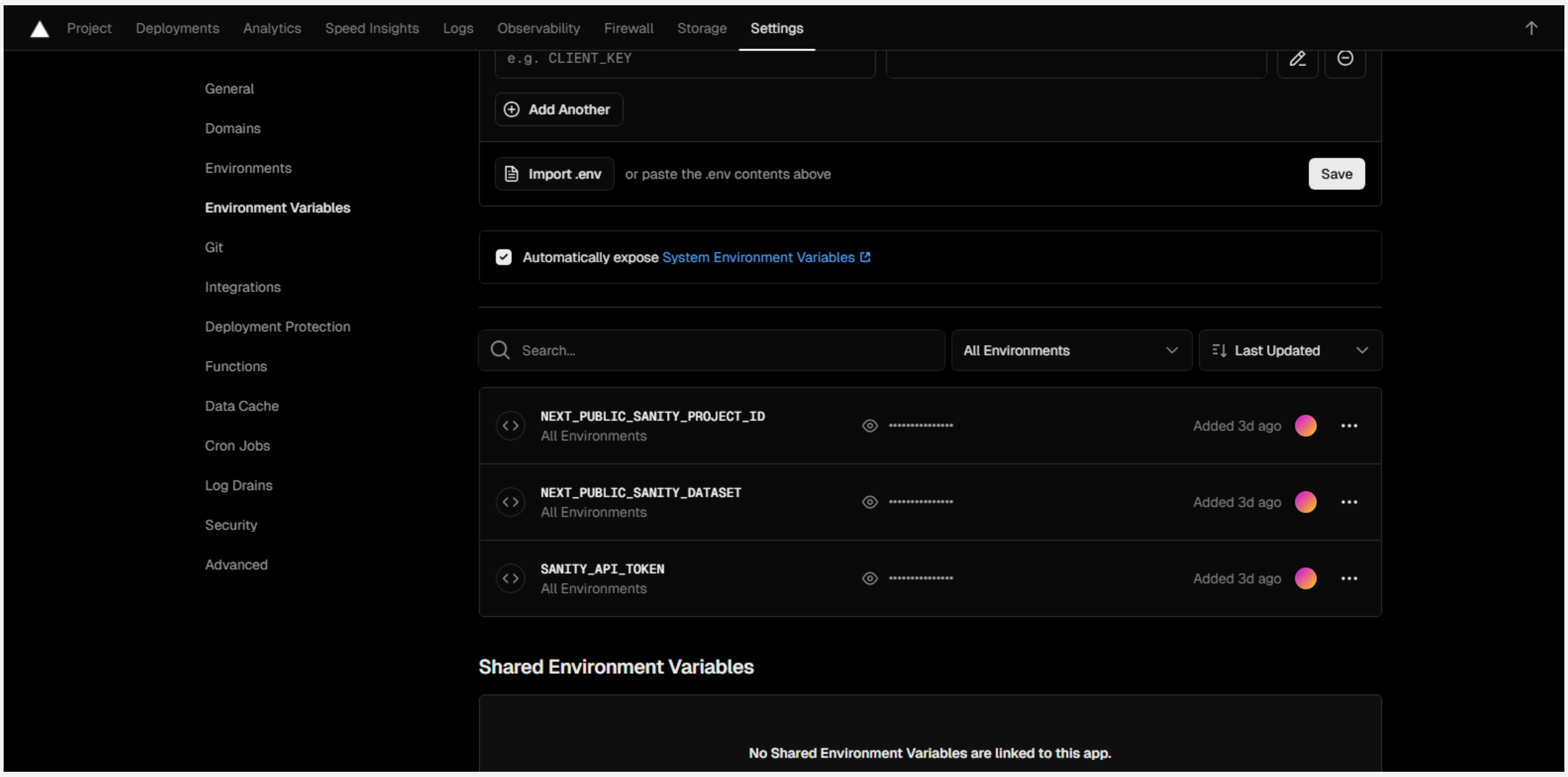
Source
main
0e146b0 day 6 and day 7

- **Documentation:**
    - Prepared a professional README.md file summarizing the project structure, setup instructions, and deployment steps.
    - Organized the GitHub repository into structured folders:
        - `src/`: Source code for the application.
        - `public/`: Static assets and images.
        - `documents/`: Deployment instructions, test reports, and project-related documents.

DAY 7 - LIVE DEPLOYMENT AND POST-LAUNCH

1: Environment Variables Securely Configure

- **Setup:** Store all sensitive information, such as API keys and database credentials, in a `.env` file.
- **Upload:** Securely configure these variables on platforms like **Vercel**, avoiding any hardcoding in the codebase.
- **Best Practice:** Ensure `.env` files are excluded from version control using `.gitignore`. This protects sensitive information during collaborative development.
- **Dynamic Retrieval:** Use environment variables dynamically in your application code to enhance flexibility and security.
- **Regular Updates:** Regularly review and rotate sensitive keys to prevent unauthorized access and mitigate potential risks.



Automatically expose System Environment Variables

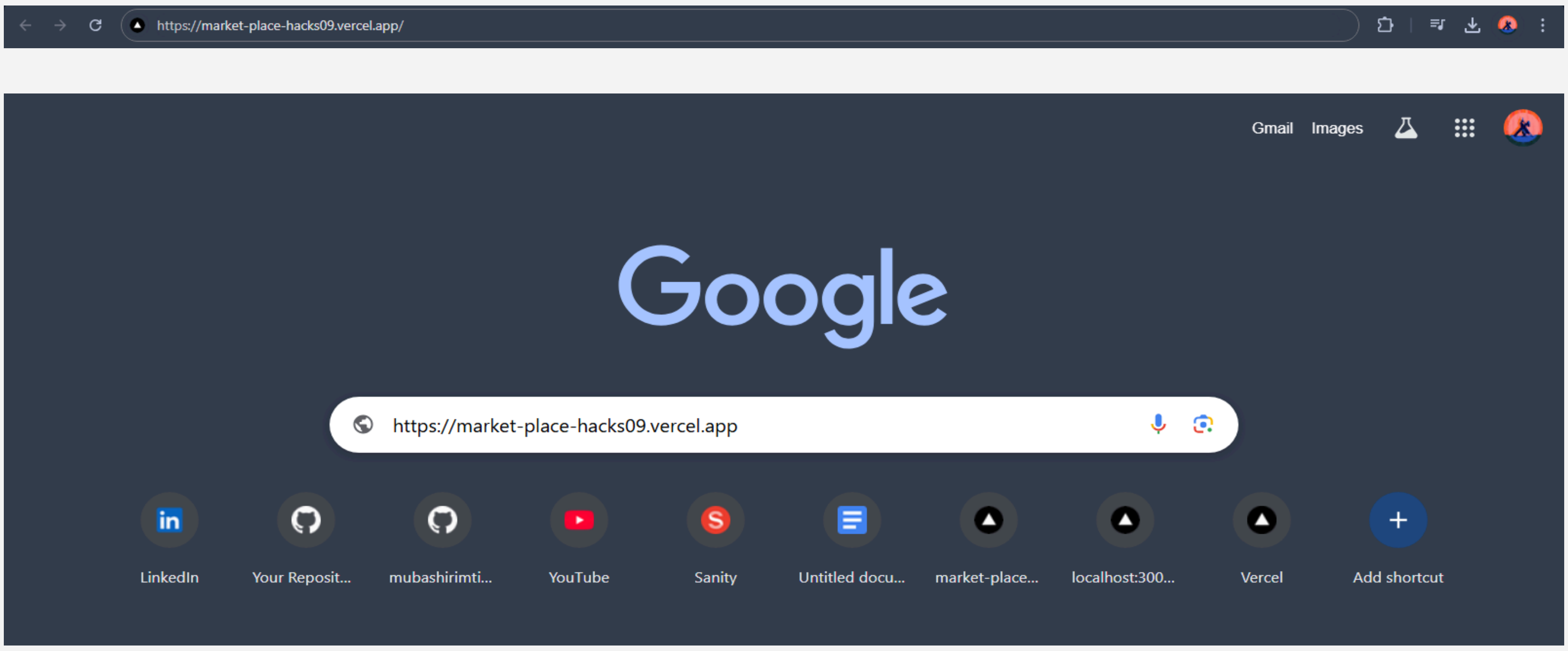
NEXT_PUBLIC_SANITY_PROJECT_ID All Environments	•••••••••••••••	Added 3d ago
NEXT_PUBLIC_SANITY_DATASET All Environments	•••••••••••••••	Added 3d ago
SANITY_API_TOKEN All Environments	•••••••••••••••	Added 3d ago

Shared Environment Variables

No Shared Environment Variables are linked to this app.

2: SSL and HTTPS Enablement

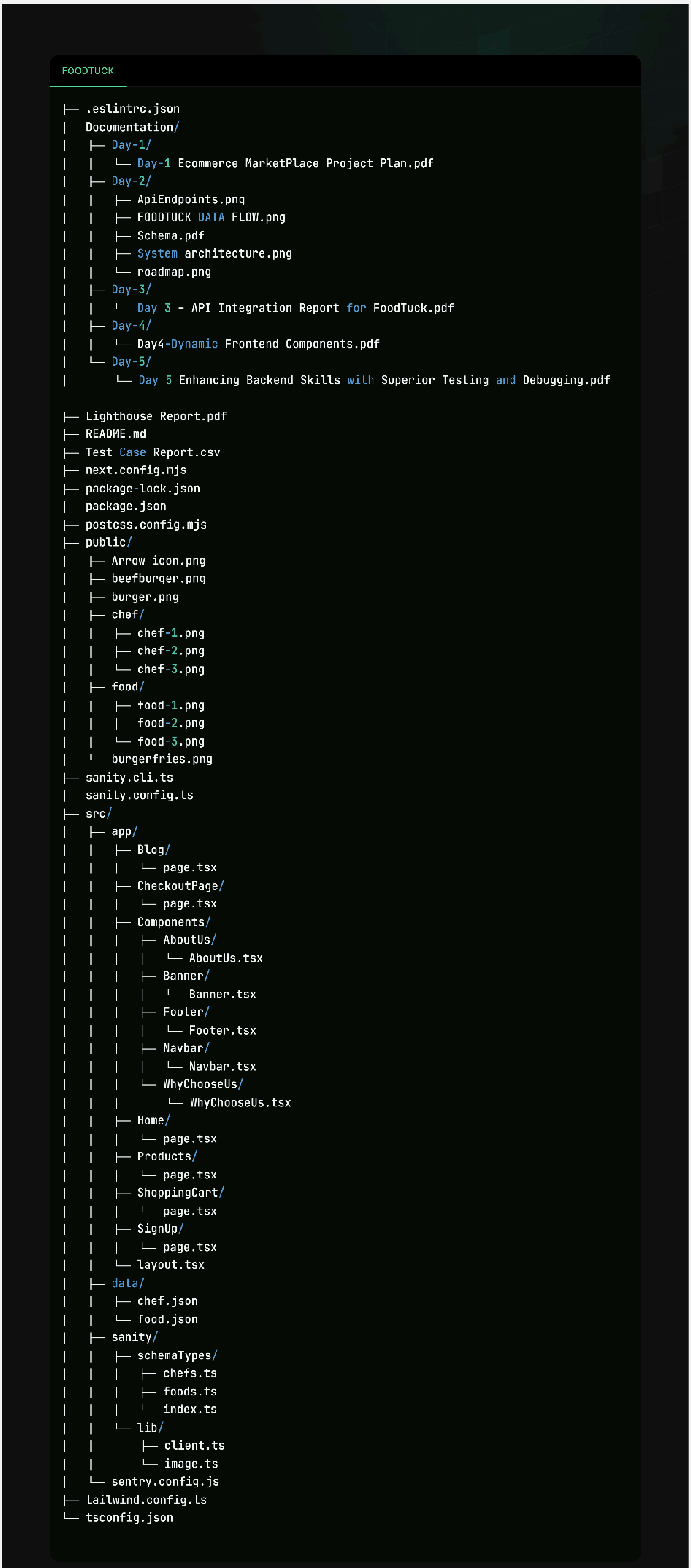
- **Purpose:** Use **SSL/TLS** to encrypt traffic and ensure secure communication between the server and client.
- **Automatic Configuration:** Hosting platforms like **Vercel** automatically enable HTTPS for custom domains.
- **Verification:** Ensure all website URLs are `https://` enabled, providing user trust and preventing data interception.
- **Benefits:** Strengthens data privacy, enhances user confidence, and complies with security standards.



3: Codebase Security and Repository Management

- **Private Repositories:** Keep the production repository private to safeguard sensitive code and proprietary information.
- **Branch Organization:** Separate staging and production branches for clear workflow management:
  - Example: Use `main` for production and `staging` for testing environments.
- **Access Management:** Restrict access to the repository to authorized personnel only, with enforced **2FA** for added security.
- **Documentation:** Include deployment guides, rollback procedures, and repository usage instructions for future projects

GITHUB FOLDER STRUCTURE:

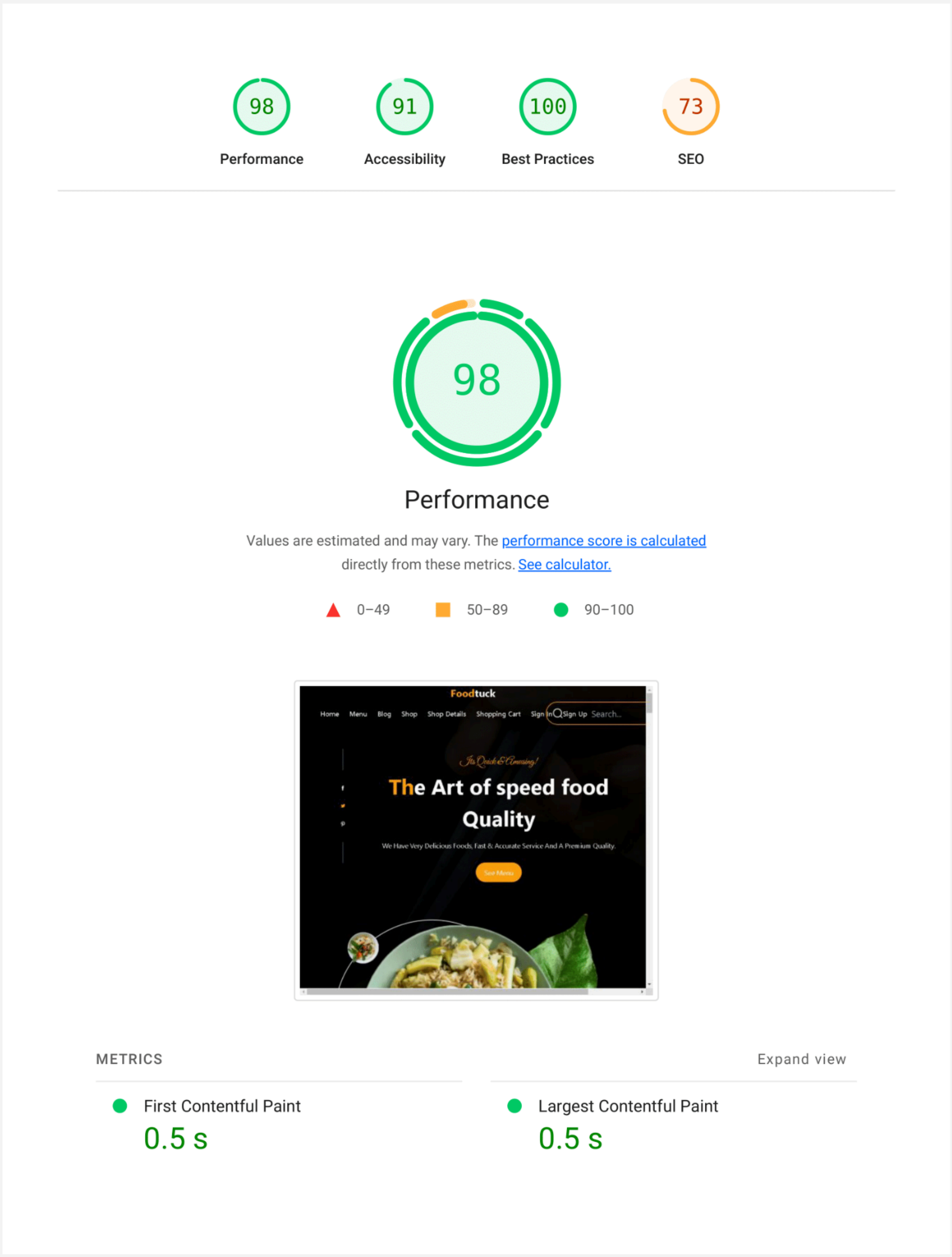

FOODTUCK

~~~
├─ .eslintrc.json
├─ Documentation/
|   ├─ Day-1/
|   |   └─ Day-1 Ecommerce MarketPlace Project Plan.pdf
|   ├─ Day-2/
|   |   ├─ ApiEndpoints.png
|   |   ├─ FOODTUCK DATA FLOW.png
|   |   ├─ Schema.pdf
|   |   ├─ System architecture.png
|   |   └─ roadmap.png
|   ├─ Day-3/
|   |   └─ Day 3 - API Integration Report for FoodTuck.pdf
|   ├─ Day-4/
|   |   └─ Day4-Dynamic Frontend Components.pdf
|   └─ Day-5/
|       └─ Day 5 Enhancing Backend Skills with Superior Testing and Debugging.pdf

├─ Lighthouse Report.pdf
├─ README.md
├─ Test Case Report.csv
├─ next.config.mjs
├─ package-lock.json
├─ package.json
├─ postcss.config.mjs
├─ public/
|   ├─ Arrow icon.png
|   ├─ beefburger.png
|   ├─ burger.png
|   ├─ chef/
|   |   ├─ chef-1.png
|   |   ├─ chef-2.png
|   |   └─ chef-3.png
|   ├─ food/
|   |   ├─ food-1.png
|   |   ├─ food-2.png
|   |   └─ food-3.png
|   └─ burgerfries.png
├─ sanity.cli.ts
├─ sanity.config.ts
├─ src/
|   ├─ app/
|   |   ├─ Blog/
|   |   |   └─ page.tsx
|   |   ├─ CheckoutPage/
|   |   |   └─ page.tsx
|   |   ├─ Components/
|   |   |   ├─ AboutUs/
|   |   |   |   └─ AboutUs.tsx
|   |   |   ├─ Banner/
|   |   |   |   └─ Banner.tsx
|   |   |   ├─ Footer/
|   |   |   |   └─ Footer.tsx
|   |   |   ├─ Navbar/
|   |   |   |   └─ Navbar.tsx
|   |   |   └─ WhyChooseUs/
|   |   |       └─ WhyChooseUs.tsx
|   |   ├─ Home/
|   |   |   └─ page.tsx
|   |   ├─ Products/
|   |   |   └─ page.tsx
|   |   ├─ ShoppingCart/
|   |   |   └─ page.tsx
|   |   ├─ SignUp/
|   |   |   └─ page.tsx
|   |   └─ layout.tsx
|   ├─ data/
|   |   ├─ chef.json
|   |   └─ food.json
|   ├─ sanity/
|   |   ├─ schemaTypes/
|   |   |   ├─ chefs.ts
|   |   |   ├─ foods.ts
|   |   |   └─ index.ts
|   |   └─ lib/
|   |       ├─ client.ts
|   |       └─ image.ts
|   └─ sentry.config.js
├─ tailwind.config.ts
└─ tsconfig.json
```]
~~~

4: Performance Optimization

- **Image Optimization:** Compress and resize images using tools like TinyPNG or ImageMagick to reduce load times.
- **API Optimization:** Minimize API response times by caching responses and reducing unnecessary requests.
- **Performance Testing:** Use Lighthouse or GTmetrix to identify bottlenecks and measure key performance metrics like First Contentful Paint (FCP) and Time to Interactive (TTI).
- **Lazy Loading:** Implement lazy loading for assets and images to improve page speed.



98 Performance

91 Accessibility

100 Best Practices

73 SEO

98

Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 ■ 50–89 ● 90–100

Foodtuck
Home Menu Blog Shop Shop Details Shopping Cart Sign In Sign Up Search...
Its Quick & Amusing!
The Art of speed food Quality
We Have Very Delicious Foods, Fast & Accurate Service And A Premium Quality.
See Menu

METRICS Expand view

● First Contentful Paint 0.5 s

● Largest Contentful Paint 0.5 s

5: Monitoring and Analytics Tools

- **Analytics**: Set up Google Analytics to track user interactions, session durations, and traffic sources.
- **Error Tracking**: Integrate Sentry for real-time error reporting and debugging insights.

  Installation Command:
  ```
  npm install @sentry/nextjs
  ```

  ○
- **Performance Monitoring**: Use tools like Pingdom or UptimeRobot to monitor site uptime and detect performance issues proactively.
- **Continuous Updates**: Regularly analyze data from these tools to implement improvements and maintain system stability.

Conclusion

Over the six days, the FoodTuck Marketplace project evolved from initial conceptualization to a fully functional staging deployment. With a meticulously organized GitHub repository, dynamic frontend components, and thorough testing protocols, the platform is now primed for live deployment in a production environment.

Next Steps:

1. Resolve any outstanding issues identified during staging tests to ensure a flawless user experience.
2. Monitor the live environment to gather user feedback and evaluate performance metrics.
3. Strategically scale the platform by incorporating advanced features such as multi-language support and predictive analytics for personalized recommendations.

This milestone marks the successful culmination of the FoodTuck Marketplace Hackathon project, setting the foundation for future growth and innovation!🚀🎉

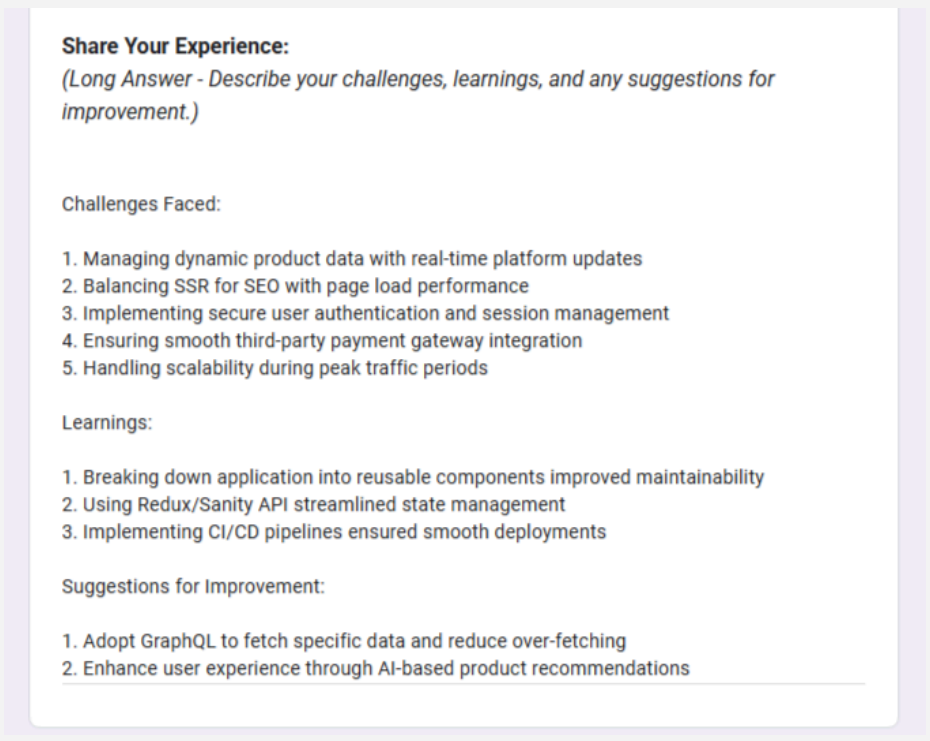
Acknowledgment and Appreciation

We extend our heartfelt gratitude to all the faculty members for their constant support, guidance, and encouragement throughout this incredible journey. Your dedication and mentorship have been the backbone of our success in this hackathon.

A special thanks to our esteemed Dean, Sir Ameen Alam, for his visionary leadership, motivating words, and valuable insights that have inspired us to push our boundaries and achieve this milestone.

This hackathon has been a transformative learning experience, and we are deeply appreciative of the time and effort invested by our mentors and organizers. Thank you for believing in us and paving the way for a bright and successful future! 🌟🎉

Journey:



Share Your Experience:
(Long Answer - Describe your challenges, learnings, and any suggestions for improvement.)

Challenges Faced:

1. Managing dynamic product data with real-time platform updates
2. Balancing SSR for SEO with page load performance
3. Implementing secure user authentication and session management
4. Ensuring smooth third-party payment gateway integration
5. Handling scalability during peak traffic periods

Learnings:

1. Breaking down application into reusable components improved maintainability
2. Using Redux/Sanity API streamlined state management
3. Implementing CI/CD pipelines ensured smooth deployments

Suggestions for Improvement:

1. Adopt GraphQL to fetch specific data and reduce over-fetching
2. Enhance user experience through AI-based product recommendations