BALTS FOOD Delivery Project

Tohir Davjanov 221ADB156
Diyorbek Kholmirzaev 221AD193
Ayubkhon Salimov 221ADB220
Myroslav Suprun 221ADB197

Group Members:

Tohir Davjanov 221ADB156 **(Administrator)**

Full-Stack Developer and responsible for code review and design of the project. Works with Admin panel like store the new products, which will be shown in frontend part of the project and transferring the data into database.

Diyorbek Kholmirzaev 221AD193

Front-end Developer, works on frontend part of the project like: design, styling, filtering and logic of the project.

Ayubkhon Salimov 221ADB220

Frontend Developer, works on design of the project, prototype of the project made by this student, also works on fixing the errors during developing the frontend part.

Myroslav Suprun 221ADB197

Backend Developer, works on backend part of the project like: registering the user, transfer the data to database,

Github Link and Project website link

Github: https://github.com/TohirD/DIP392-group_TMDA

Website: https://balts-t.onrender.com/

Admin: https://balts-admin.onrender.com

Video link: https://drive.google.com/file/d/1UE_k4oahuTigqY7nKuQ_6X2zPZrdAZlM/view?usp=sharing

Project Overview and Motivation

An online ordering website enables one to view menu items, pay, and order with ease

Why it’s needed? It simplifies ordering and payment for restaurants and customers.

Main Features:

-Registration of the user

-Simple menu browsing with categories

-Quick ordering and secure payment

-Store the products with from Admin panel and data transferred into database.

Compatible with desktop

The system serves to address slow ordering and food delivery issues in restaurants for a given location. The system enables users, such as customers or restaurateurs, to view menus, order food, pay, and monitor their delivery in an easy manner. This matters because it simplifies operations for customers while enabling restaurants to operate more optimally.

Chosen SDLC Methodology

SDLC Model: Agile

Why: Agile accommodates small increments, can easily accommodate changes, and delivers features incrementally, which conforms to shifting necessities of a food ordering system.

Requirements Gathering & Analysis

We gathered information through conducting surveys, interviewing rest
aurant owners, and analyzing our competitors.

Questions focused on user needs, ordering preferences, payment security, and tracking features.

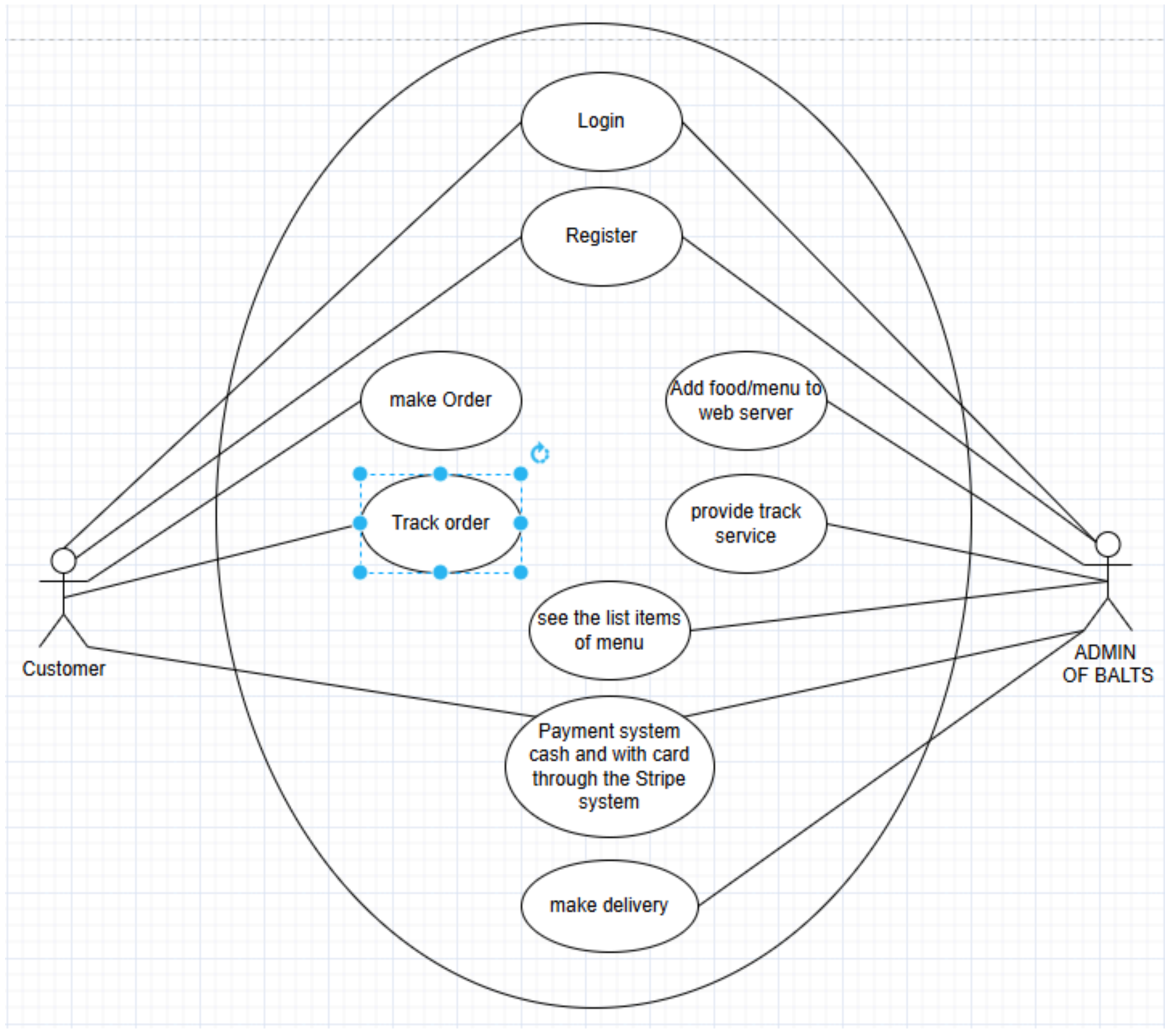
Assumptions: People can access the web; restaurants
can modify their menus online.

Software Requirements Specification

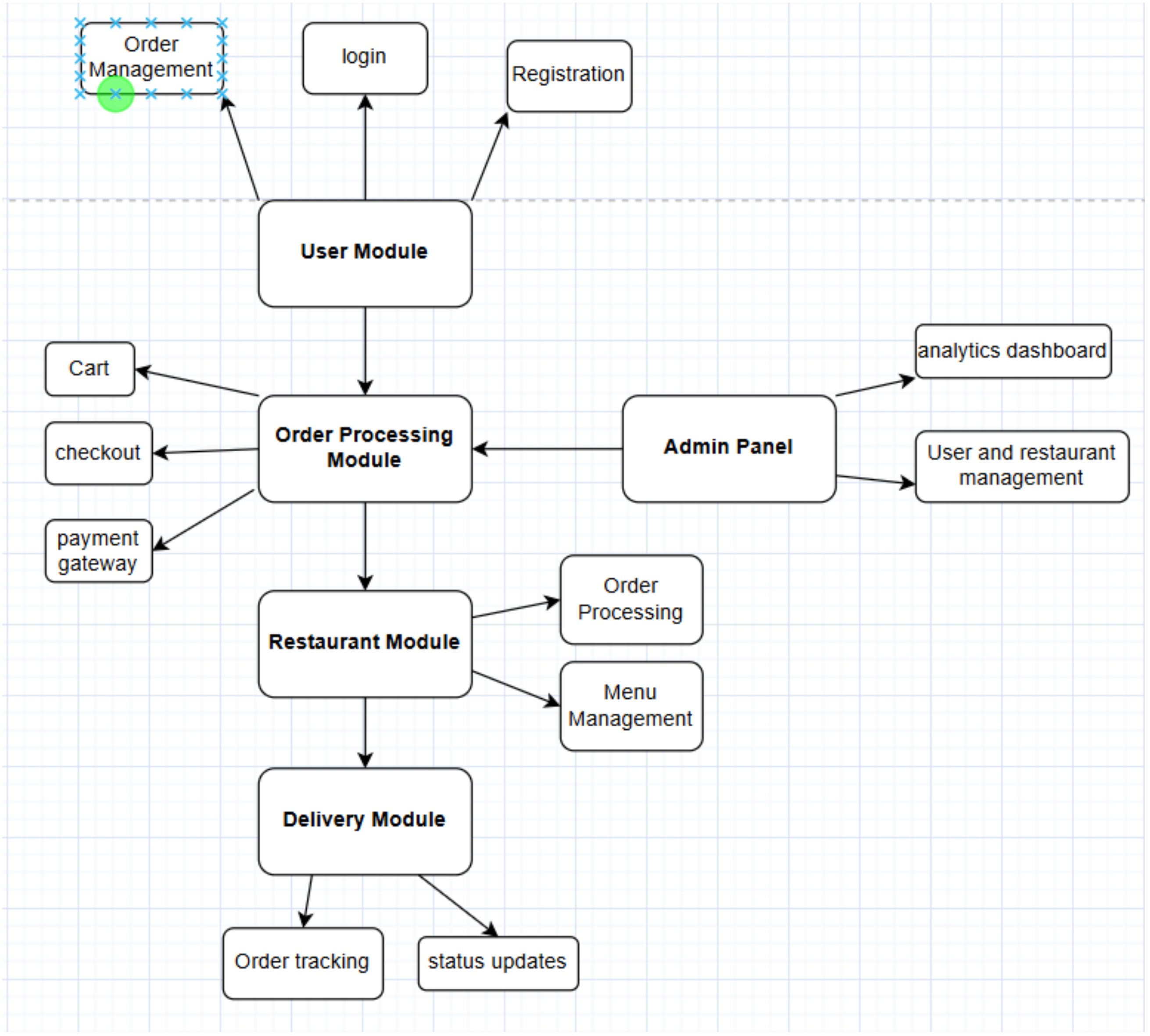
- Functional:
  - User registration/login
  - Menu browsing and order placement
  - Online payment via Stripe
  - Real-time order status updates
- Non-functional:
  - Mobile responsiveness
  - Secure data handling
  - Fast load time

System and Software Design

The system architecture employs React.js on the frontend to enable a dynamic and responsive user interface. The backend is built using Node.js and Express.js, which handle API requests and database interactions. MongoDB is the storage system for customer profiles, order details, and menu items, using a scalable and flexible schema.


Login
Register
make Order
Add food/menu to web server
Track order
provide track service
see the list items of menu
Payment system cash and with card through the Stripe system
make delivery
Customer
ADMIN OF BALTS

System and Software Design



Implementation Phase

Frontend: React.js, CSS

Backend: Node.js, Express.js

Database: MongoDB (NoSQL)

Payment Integration: Stripe, PayPal

Code Organizations:

Structure of codes of the project divided into /frontend, /backend and /admin folders and inside of each there are components, routes, models and etc.

Implementation Phase

Teamwork:

As being 4 people in our group, we decided to divide the project into 2 equal work between us, both of the students Diyorbek and Ayubkhon worked in frontend part, Me (Tohir Davjanov) and Myroslav worked in backend. Design of the project was done by Diyorbek.

Testing Strategy

1)Used black-box to track and make validation of the user input/ output

2) Testing system that it works properly: login, order placement, payment processing and tracking.

Testing Strategy

Testing encompassed two dimensions:

Functional Testing: A test was performed to ensure that critical functionalities such as user registration, login processes, shopping cart handling, order processing workflows, and payment options like Stripe and cash on delivery (COD) functioned as expected. Access controls have been established, review tokens were validated upon checkout, and unauthorized activity did not play out in the system

Security Testing: As part of the evaluation of security functionalities, we performed supplementary assessments centred on specific vulnerabilities related to the payment or checkout systems. Our efforts involved attempts at bypassing the login interface directly to access the checkout; however, we failed to achieve this, thus proving the strength of the protective elements integrated into the login process. Then, we tested the processing of transactions with Stripe using a web hook, and when payments succeeded, they remained; however, payments that did not succeed were dropped according to the specified business rules.

Test Results and Validation

User authentication and authorization.

Test 1: When the user tries to go to the shopping cart or proceed with an order without previous authentication, they should be redirected to the login page with an error message saying, "sign in first in order to make an order."

Testing 2: The authentication of the registered password and email should ensure that the token for a user is saved into localStorage and will allow access to the private routes (e.g. cart, order submission).

Cart:

Testing 3: Adding an item to the cart should change the cart state and save the creates the item and the quantity in the logged in user's MongoDB cartData field.

Testing 4: The application should automatically redirect to the cart if the user attempts to navigate to checkout and their cart is empty.

Order Acquisition:

Testing 5: If all fields are filled out on the delivery form and the user clicks Place Order with "Cash on Delivery" as the payment method, a new order should be created in the MongoDB database, and the payment status should remain false.

- Test Results
  User authentication and access control mechanisms:
- The system successfully restricted access to secured paths, such as the checkout page. Users who tried to proceed without valid authentication were asked to log in and then redirected accordingly.
- Upon successful authentication, the system stored the token in local storage, thus allowing the continuous access of the user to authenticated routes.
- **Shopping Cart Functionality and Data Synchronization:**
- The products added into the shopping cart were accurately presented in both user interface and in the MongoDB database under the field cartData relevant to the user.
- When they tried to checkout with an empty shopping cart, the users were taken to the cart page, thus fulfilling the necessary logical flow.
- **Delivery Information Document:**
- The delivery details form allowed a structured input of the user details, including name, address, and contact details.
- Before submission, all fields required validation, thus ensuring complete and consistent data collection for each order.

Test Results and Validation

- Stripe:
- **Testing 6:** If Stripe is the selected payment method and payment is complete, the order in MongoDB should be updated to payment: true.
- **Testing 7:** Selecting Stripe and clicking Place Order should create a secure payment session URL and redirect the user to the Stripe Checkout website.
- **Test 8:** If payment is successful, the system should update the corresponding order's payment status to true. If failed, the order should be deleted automatically via verifyOrder() function.

- **Payment Modalities (Cash on Delivery and Stripe Integration):**
- Under the Cash on Delivery (COD) arrangement, the order was successfully placed and recorded within the database, with the related payment status being unpaid.
- The Stripe integration was tested under evaluation mode and performed as expected. After a successful payment, the payment status for the corresponding order was indicated as true on the database.
- Canceled and failed transactions caused the automatic deletion of the related order from the database, thus preventing the storage of incorrect information.

Process Reflection and Lessons Learned

- Challenges:

  Stripe payment system,
  Make application responsive,

  Admin Panel

- Lessons:
  - Importance of early API testing
  - Commit comments and work in team that really helps for students

Conclusion & Next Steps

Summary: The system successfully enables end-to-end food ordering and delivery management.

Gather user feedback for improvements

Expand restaurant partnerships and optimize UI/UX