TITLE

Title: Real-Time Courier Tracking and Delivery Estimation Mobile Application (KuryeNet)

Prepared by: Burak Göksu 190315029

Submitted to: Manisa Celal Bayar University

Date: 30/05/2024

ABSTRACT

In this project, we aim to develop a mobile application that allows users to track the real-time location of their courier and provides an estimated delivery time based on a unique order code obtained from third-party applications such as Yemeksepeti or Trendyol. The order code will be simulated within our system, and the application will process it to initiate live tracking. The system will handle order details, recipient information, location coordinates (longitude, latitude), timestamps, and the positions of the shop and courier.

The back-end of the project will be implemented in Java using the Spring Boot framework, incorporating Maven for package management. PostgreSQL will serve as the database to store order details, recipient and courier location coordinates, and other relevant information. JPA Repository will facilitate database operations, decoding the order code to extract courier information, and updating the database when the courier logs in, enabling real-time location updates.

The integration of Google Maps API will be crucial for displaying maps in the mobile application. Users will be able to visualize the courier's live location and track the estimated delivery time. For real-time courier tracking, Spring Boot's WebSocket technology will be employed, ensuring bidirectional communication between the server and the mobile application.

The second part of the project involves the development of a mobile application using React Native. Couriers will have a dedicated login screen where they can view available orders, select and initiate deliveries. Customers, on the other hand, will only need to enter the order code to track the order status (preparing, ready for delivery, delivered) and monitor the live location of the courier if the order is out for delivery.

Additionally, the application will send mobile notifications for courier movements, delivery updates, and approaching deliveries, encouraging active user engagement. Couriers will benefit from features such as real-time traffic information, alternative routes, and fuel-efficient paths, optimizing the delivery process.

In this integration, we introduce a predictive analytics feature using a CatBoost machine learning model into the existing KuryeNet system, which is primarily based on Spring Boot, PostgreSQL, and React Native technologies. The enhancement focuses on predicting the delivery time for courier services through a Flask API that interacts seamlessly with the Spring Boot backend. The CatBoost model utilizes a variety of features including the delivery person's age, weather conditions, type of order, vehicle used, and urban density to accurately estimate the time taken for a delivery to be completed. This prediction is crucial for optimizing logistics and improving user satisfaction by providing more accurate delivery times.

The Flask API serves as a bridge between the machine learning model and the main application, handling requests to calculate delivery times (/predict) and to log delivery data (/write_data) once orders are completed. When an order starts, the prediction API is triggered, fetching the estimated delivery time from the CatBoost model. This estimate helps in setting realistic expectations for the customers and aids couriers in managing their schedules efficiently. Post-delivery, the writeData API captures essential metrics from the completed deliveries which are then utilized to further train and refine the machine learning model, thereby enhancing its accuracy and reliability in real-world scenarios. This continual learning process ensures the model remains effective in the face of changing conditions and data variances.

To enhance the functionality of our real-time courier tracking system, we have integrated a Flask-based API that interacts seamlessly with our CatBoost machine learning model. This Flask API serves as a crucial intermediary, facilitating real-time predictions and data handling between the courier tracking interface and the predictive model backend. The API consists of four main endpoints:

/predict - Receives input variables related to the delivery (such as age of the courier, type of vehicle, weather conditions, etc.) through POST requests, and utilizes the loaded CatBoost model to predict the delivery time based on these variables. The result is then sent back as a JSON response, which includes the predicted time and a rounded-off prediction for simpler interpretation.

/write_data - Post-delivery, this endpoint is called to log the delivery data into our system. It takes detailed delivery information (including the outcomes of the delivery and operational metrics) via POST requests, and stores this data in a structured format, which can later be used for further analysis or model retraining purposes.

/retrain_model - To maintain the accuracy of our predictions, this endpoint allows for dynamic retraining of the CatBoost model. It combines the newly collected data with the existing dataset and re-trains the model to reflect the latest trends and conditions in the delivery environment.

/logs - Provides access to the server logs, enabling easy troubleshooting and monitoring of the API's operations through GET requests. This endpoint ensures transparency and allows for quick diagnostics in case of issues.

This Flask API not only enhances the real-time capabilities of our system by enabling dynamic data processing and immediate feedback but also supports continuous improvement of the delivery predictions through iterative model training. This setup exemplifies a robust use of modern web frameworks and machine learning technology to significantly improve operational efficiencies in real-time tracking systems. The API's modular design and its integration into the broader system architecture demonstrate an innovative approach to solving traditional problems in the courier service industry through technology.

Key Words

Real-time Tracking

Courier Delivery

Mobile Application

Estimated Delivery Time

CatBoost

Machine Learning

Flask API

Introduction

In an age defined by rapid technological advancement, the logistics and delivery sector stands at the forefront of transformative innovations. The surge in e-commerce and on-demand services has underscored the critical importance of efficient, transparent, and user-centric courier tracking systems. Recognizing this paradigm shift, our project, titled "Real-time Courier Tracking System with Estimated Delivery Time," aims to revolutionize the delivery experience by amalgamating cutting-edge technologies to create a dynamic, real-time tracking solution. This multifaceted project encompasses both back-end and front-end development, weaving together Java, Spring Boot, React Native, and Google Maps API, backed by the real-time capabilities of Spring Boot's WebSocket technology.

Context and Rationale:

The contemporary landscape of consumer expectations demands more than just timely deliveries; it necessitates visibility and control throughout the entire delivery process. The integration of real-time tracking mechanisms has become a hallmark of progressive logistics solutions, offering a symbiotic enhancement to customer satisfaction and courier efficiency. By initiating this project, we respond to the evolving needs of the digital age, where immediacy and transparency are paramount.

Project Overview:

The cornerstone of our project lies in the realization of a seamless and intuitive courier tracking system. It comprises two integral components: the back-end, designed with Java and the Spring Boot framework, and the front-end, articulated through React Native. The strategic incorporation of Google Maps API ensures not only accurate geospatial representation but also a visually engaging interface for users. Meanwhile, the adoption of Spring Boot's WebSocket technology facilitates real-time communication, a fundamental necessity for live tracking updates.

Real-time Tracking Dynamics:

The crux of our innovation hinges on the ability to provide users, both customers and couriers, with real-time tracking capabilities. This involves decoding third-party order codes, simulating orders within our system, and seamlessly integrating with Google Maps for live location updates. The interplay between the back-end and front-end ensures that users experience a dynamic and responsive tracking interface, where the courier's movements are mirrored on their screens in real-time.

Optimizing the Delivery Experience:

The objective is not merely to track but to enhance the entire delivery experience. For couriers, the system presents a streamlined interface where they can view and select available orders, initiating deliveries with efficiency. Customers, on the other hand, gain access to a simplified interface where a unique order code is the gateway to real-time status updates, estimated delivery times, and live tracking if the delivery is en route.

Innovation and Technological Integration:

The back-end, powered by Java and Spring Boot, capitalizes on the robustness of these technologies for efficient data processing and management. PostgreSQL serves as the repository for order details, recipient information, and location coordinates, managed seamlessly through JPA Repository. The incorporation of Google Maps API elevates the user interface, offering an immersive and visually informative tracking experience. Real-time tracking dynamics are enabled by Spring Boot's WebSocket technology, facilitating bidirectional communication between the server and the mobile application.

Environmental Impact and Health Considerations:

Beyond the technological and logistical dimensions, our project carries ramifications for health, the environment, and safety. Real-time tracking minimizes uncertainty, reducing stress for both customers and couriers. The mental well-being of individuals involved in the delivery process is thus positively influenced. From an environmental standpoint, optimizing courier routes not only enhances efficiency but also reduces fuel consumption and associated carbon emissions. In an era increasingly defined by environmental consciousness, our project aligns with the broader global effort to address climate change through sustainable transportation practices.

Legal Dimensions:

The integration of a real-time courier tracking system introduces legal considerations, paramount among them being data privacy and security. In an era where data breaches and privacy concerns loom large, our commitment is to handle customer information securely and ethically. Compliance with data protection laws, such as GDPR, underscores our dedication to safeguarding user privacy.

In the evolving landscape of urban logistics and on-demand delivery services, predictive modeling has become a pivotal technology, enhancing efficiency and customer satisfaction. KuryeNet, an established courier management platform, harnesses Spring Boot, PostgreSQL, and React Native to provide comprehensive delivery services. The recent integration of a CatBoost machine learning model through a Flask API represents a significant stride towards leveraging artificial intelligence to predict delivery times with greater precision.

CatBoost Integration Overview

CatBoost is an advanced gradient boosting machine learning algorithm that excels in handling categorical variables and complex data correlations without extensive data preprocessing that other machine learning models might require. The integration of the CatBoost model into KuryeNet involves several key parameters and features that influence the prediction accuracy:

Delivery Person Age: Age of the courier which might affect the delivery speed and efficiency.

Weather Conditions: Real-time weather data (e.g., sunny, stormy, cloudy) that can impact delivery times due to traffic slowdowns or faster routes in favorable conditions.

Road Traffic Density: The current traffic conditions on the courier's route, categorized into levels such as low, medium, high, and jam. This is crucial as it directly affects the travel time from the pickup to the delivery point.

Type of Order: Differentiates what kind of order is being delivered (e.g., meals, groceries, electronics), which can vary the preparation and handling times significantly.

Type of Vehicle: The vehicle type (e.g., motorcycle, car, scooter) used for the delivery, influencing navigation speed, route selection, and parking time considerations.

Multiple Deliveries: Indicates whether the courier is making more than one delivery in a single trip, which can extend the time spent per delivery.

City Category: Urban or metropolitan; larger areas might lead to longer delivery times due to the distance and possible complications in navigation.

Distance: The total distance to be covered to complete the delivery.

Day Type: Working day or weekend/holiday, which affects traffic patterns.

Time Category: Time of the day the delivery is made, segmented into morning, noon, evening, and night.

Operational Flow with Flask API

The Flask API acts as a middleware between the KuryeNet system and the CatBoost model, facilitating two main operations: prediction requests and data logging post-delivery. At the commencement of an order, the /predict endpoint is invoked where the Flask API communicates with the CatBoost model to predict the estimated delivery time based on the current conditions and courier details. This prediction helps in optimizing the delivery schedules and managing customer expectations efficiently.

Once the delivery is completed, the /writeData endpoint is triggered, capturing essential delivery metrics such as actual time taken, distance covered, and the conditions under which the delivery was made. These data points are crucial for the continuous training of the CatBoost model, allowing it to adapt to changing patterns and improve its predictive accuracy over time.

Through this sophisticated integration of machine learning with traditional web technologies, KuryeNet aims to not only enhance the operational efficiency of its courier services but also to elevate the overall user experience by providing timely and reliable predictions on delivery times. This proactive approach in managing logistics operations underscores KuryeNet's commitment to innovation and customer-centric solutions in the courier industry.

Realistic Constraints and Conditions:

a. Sustainable Development Goal (SDG):

Our project aligns closely with the Sustainable Development Goal 9: "Industry, Innovation, and Infrastructure." This goal emphasizes the development of resilient infrastructure, the promotion of inclusive and sustainable industrialization, and fostering innovation. By creating a real-time courier tracking system, we contribute to the enhancement of infrastructure efficiency within the delivery sector. The implementation of innovative technologies not only improves the logistics industry's productivity but also aligns with the goal's broader aim of creating sustainable and resilient infrastructure.

Through our integration of advanced machine learning models (CatBoost) and real-time data processing via Flask APIs, we are enhancing the technological landscape of courier services, thereby supporting sustainable industrial growth. Our project leverages real-time data to optimize delivery routes, which minimizes fuel consumption and reduces the carbon footprint associated with courier services. This optimization is a step towards sustainable cities and communities (SDG 11) as it contributes to efficient urban logistics and infrastructure.

b. Effects on Health, Environment, and the Problems of the Age Reflected in the Field of Engineering:

The impact of our project extends beyond the realm of technology, reaching into the domains of health, environment, and safety. Real-time courier tracking reduces the uncertainty associated with deliveries, minimizing stress for customers and couriers. This contributes to the mental well-being of individuals involved in the delivery process.

From an environmental perspective, optimizing courier routes and providing alternative paths help in reducing fuel consumption and, consequently, carbon emissions. This aligns with the global effort to address climate change and create sustainable practices within the transportation sector. The efficient use of resources through route optimization reflects a commitment to responsible engineering practices.

Moreover, our project addresses the contemporary challenges faced by the engineering field. The demand for streamlined logistics in the age of e-commerce and on-demand services necessitates innovative solutions. By incorporating real-time tracking, we tackle the challenges posed by the increasing complexity and expectations within the logistics and delivery sector. This reflects the adaptability and problem-solving capability inherent in engineering.

Our project's impact on health, environment, and safety is multi-dimensional. Utilizing CatBoost machine learning models to predict delivery times more accurately helps in reducing the time vehicles spend idling on the roads, which directly reduces air pollution and greenhouse gas emissions. This contributes positively to environmental health and aligns with global efforts to combat air pollution.

From a health perspective, reducing delivery times decreases the stress levels of delivery personnel by providing them with more predictable schedules and reducing pressure during their shifts. This improvement in working conditions can lead to better overall health outcomes for employees involved in the logistics sector.

The real-time tracking system also enhances road safety by optimizing delivery routes, which helps in avoiding congested and potentially dangerous routes. By integrating weather data, our system can warn drivers about unsafe driving conditions, thereby reducing the risk of accidents. These advancements reflect the ongoing problems in the field of engineering such as the need for real-time data integration and the application of IoT in everyday logistics and infrastructure management.

c. Legal Consequences:

The implementation of a real-time courier tracking system introduces legal considerations related to data privacy, security, and adherence to regulations within the transportation industry. Ensuring that customer information is handled securely and ethically is a primary concern. Compliance with data protection laws, such as GDPR, is crucial in safeguarding user privacy.

Additionally, legal consequences may arise concerning the use of location-based services. Clear communication and consent mechanisms must be established to address potential concerns related to tracking individuals' real-time locations. Adhering to industry regulations and standards is essential to mitigate legal risks and build trust among users and stakeholders.

In conclusion, our project not only addresses the immediate need for an advanced courier tracking system but also aligns with broader societal and environmental goals. The sustainable development goal, coupled with the consideration of health, environmental impact, and legal consequences, underscores the comprehensive approach we take in developing a responsible and impactful solution within the field of engineering.

The integration of a real-time courier tracking system introduces several legal considerations, primarily concerning data protection and privacy laws. As our system processes significant amounts of personal and operational data, it must comply with the General Data Protection Regulation (GDPR) in the EU, or similar regulations in other jurisdictions. This compliance involves ensuring that all collected data is used strictly for its intended purpose, stored securely, and not disclosed without consent.

Additionally, the use of machine learning models like CatBoost raises ethical questions about decision-making processes, particularly if these decisions impact human jobs or privacy. It is imperative to maintain transparency about how data is used within these models and to provide assurances that decisions are fair and equitable.

Moreover, our reliance on third-party services like Google Maps and OpenWeatherMap for real-time data integration means we must be diligent in managing data sharing agreements and service level agreements (SLAs) to avoid legal breaches that could arise from data mishandling by these providers.

In summary, our project not only addresses the immediate need for an advanced courier tracking system but also reflects broader societal concerns including sustainable economic development, environmental health, and legal and ethical data use. The systems and practices we have implemented ensure that the project adheres to relevant laws and regulations, promoting trust and reliability among users and stakeholders. These measures mitigate potential legal risks and align with our commitment to ethical engineering practices.

Literature Analysis

The advent of technology in the logistics and delivery sector has revolutionized the way we perceive and manage courier services. Real-time tracking systems have become a hallmark of efficiency, transparency, and customer satisfaction within this domain. This literature analysis aims to provide an in-depth exploration of the state-of-the-art in real-time courier tracking and delivery management, shedding light on key technologies, trends, and similar applications that have shaped the landscape.

1. Real-Time Tracking Technologies:

Real-time tracking technologies have witnessed significant advancements, primarily driven by the ubiquity of GPS technology. Kumar and Reddy (2019) highlight the transformative impact of GPS in enabling precise location tracking for couriers and deliveries. This technology has become a cornerstone for modern courier tracking systems, allowing for accurate geospatial representation and real-time updates on parcel locations.

Complementing GPS, RFID technology has emerged as a viable solution for real-time parcel tracking. Ghadge et al. (2020) discuss the implementation of RFID tags to monitor packages, providing a scalable and efficient means of tracking. Additionally, Internet of Things (IoT) devices integrated into courier vehicles contribute to real-time tracking by offering insights into vehicle conditions and dynamic route optimization (Mishra et al., 2021). These technologies collectively contribute to the comprehensive real-time tracking ecosystem.

2. Similar Applications:

Several applications stand out as exemplars in the realm of real-time courier tracking, setting benchmarks for efficiency and user experience. Notable among these are Yemeksepeti and Trendyol. Yemeksepeti, a popular food delivery application, employs real-time tracking to keep users informed about the status of their food orders and the location of the delivery person. Similarly, Trendyol, a prominent e-commerce platform, integrates real-time tracking to enhance the transparency of the delivery process.

Ride-sharing applications like Uber and Lyft also serve as analogous models. These platforms allow users to track the real-time location of their assigned drivers, estimate arrival times, and receive notifications throughout the journey. Borrowing from these applications, our project aims to adapt real-time tracking principles to optimize the courier and delivery experience.

3. Integration of Maps and Geospatial Data:

The integration of mapping technologies and geospatial data has become integral to the success of real-time courier tracking systems. Google Maps API, as discussed by Tanwar et al. (2020), stands out as a dominant player in providing accurate and reliable mapping services. The API enables the visualization of courier routes, estimation of delivery times, and dynamic updates based on real-time traffic conditions. Furthermore, Geographic Information System (GIS) technologies have been explored for route optimization and spatial analysis, contributing to informed decision-making in route planning, particularly in urban environments with complex traffic patterns (Deb et al., 2018). The amalgamation of these technologies ensures a robust and efficient tracking infrastructure.

4. Mobile Application Development with React Native:

The choice of React Native for mobile application development represents a strategic decision, aligning with the industry's shift towards cross-platform frameworks. Lerner (2017) emphasizes React Native's ability to strike a balance between performance and development speed, allowing for the creation of a single codebase deployable on both iOS and Android platforms. This approach significantly reduces development time and ensures a consistent user experience across different devices.

The success stories of applications like Facebook, Instagram, and Airbnb, all developed using React Native, underscore its capabilities in building robust and scalable mobile applications. In the context of courier tracking, the utilization of React Native ensures a responsive and user-friendly interface for both customers and couriers.

5. Comparative Analysis of Machine Learning Algorithms in Logistics:

Comparative studies, such as the one conducted by Chen et al. (2019), have evaluated various machine learning algorithms for their effectiveness in logistics and delivery systems. The CatBoost algorithm, with its handling of categorical variables and robustness to overfitting, has been identified as particularly suited for applications in this domain. Our project's choice of CatBoost is thus validated by literature as a sound methodological decision.

Conclusion:

In conclusion, the literature analysis underscores the dynamic landscape of real-time courier tracking and delivery management. The integration of GPS, RFID, IoT devices, mapping technologies, React Native, and WebSocket technology collectively forms a sophisticated framework for modern courier tracking systems. Similar applications such as Yemeksepeti, Trendyol, Uber, and Lyft have set high standards in terms of user experience and efficiency, providing valuable insights for our project.

The synthesis of these technologies and approaches positions our project at the intersection of innovation and practicality. Beyond addressing immediate industry needs, the project aligns with broader societal objectives of transparency, efficiency, and sustainability in logistics and delivery. As we embark on the development journey, the literature analysis serves as a robust foundation, guiding the integration of state-of-the-art technologies and methodologies to redefine the standards for real-time courier tracking systems.

The literature on the application of machine learning in courier tracking and delivery systems provides a solid foundation for the integration of the CatBoost model in KuryeNet’s operations. This analysis not only highlights the potential benefits of such an integration in terms of improved accuracy and efficiency but also aligns with broader trends in the logistics sector towards more data-driven and predictive operational models. As we move forward, it will be crucial to continuously monitor and adapt to new research findings and technological advancements to maintain and enhance the effectiveness of the system.

Standards to be Used

In the development of our real-time courier tracking system, adherence to engineering standards is paramount to ensure robustness, interoperability, and user safety. The project will align with established standards, both industry-specific and general engineering guidelines, to foster a reliable and ethically sound solution.

1. Geospatial Data Standards:

The project will adhere to geospatial data standards such as GeoJSON for the representation of geographical features. This ensures compatibility and interoperability with mapping technologies and Geographic Information System (GIS) platforms.

2. Data Security Standards:

To safeguard user information and uphold data privacy, the project will comply with industry-standard data security measures, including encryption protocols (SSL/TLS) for data transmission and secure storage practices.

3. Mobile Application Development Standards:

Following best practices in mobile application development, the project will adhere to guidelines provided by iOS Human Interface Guidelines and Android App Quality guidelines for an optimal user experience on both platforms.

4. Database Standards:

The project will leverage PostgreSQL as the database management system, following SQL standards for data modeling and storage. The use of JPA (Java Persistence API) ensures adherence to Java EE standards for data access.

5. WebSocket Standards:

In implementing real-time communication, the project will adhere to WebSocket standards (RFC 6455) to enable bidirectional communication between clients and the server. This ensures a reliable and standardized approach to real-time updates.

6. Mapping Standards:

Google Maps API will be integrated, aligning with the standards and guidelines provided by Google for the Maps JavaScript API. This ensures compliance with mapping industry practices and a seamless user experience.

7. Code Quality Standards:

The project will adopt coding standards and best practices, following guidelines such as the Java Code Conventions and Airbnb's JavaScript style guide for server and client-side development, respectively.

8. Accessibility Standards:

The project will adhere to accessibility standards, such as the Web Content Accessibility Guidelines (WCAG), to ensure that the mobile application is inclusive and accessible to users with diverse needs.

9. Machine Learning Model Development Standards:

CRISP-DM (Cross-Industry Standard Process for Data Mining): This standard provides a non-rigid approach to planning, implementing, and evaluating machine learning models, which is crucial for developing predictive systems like ours.

PMML (Predictive Model Markup Language): Ensuring that the CatBoost model can be shared and utilized across different systems, PMML provides a standard way to represent predictive models.

Conclusion:

Adhering to these standards is not only a commitment to compliance but also a reflection of our dedication to providing a high-quality, secure, and reliable courier tracking service. By following these guidelines, KuryeNet aims to enhance the operational efficiency of its services, safeguard user data, and deliver an inclusive user experience across all platforms. These standards will guide the development, implementation, and continuous improvement of the integrated CatBoost model in our courier tracking system.

Approaches, Techniques, and Technologies to be Used

1. Java and Spring Boot Framework:

The back-end development will be implemented using Java and the Spring Boot framework. This allows for the creation of robust and scalable RESTful APIs, leveraging the extensive features and modules provided by Spring Boot for efficient development.

2. React Native:

For the mobile application development, React Native will be employed. This cross-platform framework enables the creation of a single codebase deployable on both iOS and Android platforms, optimizing development time and resources.

3. PostgreSQL and JPA Repository:

PostgreSQL will serve as the relational database management system, offering a robust and scalable solution for storing order details, user information, and geographical coordinates. JPA Repository will be utilized to streamline database interactions.

4. Google Maps API:

Google Maps API will be integrated for geospatial visualization and mapping services. This API provides comprehensive mapping functionalities, including real-time traffic updates, route planning, and location-based services.

5. React Native Libraries:

The project will leverage React Native libraries, such as "react-native-geolocation-service" and "react-native-maps," to access and process user and courier location data. These libraries provide essential functionalities for location-based services.

6. Maven for Package Management:

Maven will be used for package management, streamlining the build and dependency management processes. This ensures a systematic and efficient approach to handling project dependencies.

7. WebSocket:

WebSocket technology will be integrated into the project for real-time communication between the server and the mobile application. This technology facilitates bidirectional communication and ensures instant updates on courier locations and order statuses.

8. Git for Version Control:

Git will be employed for version control, enabling collaborative development and ensuring a systematic approach to code management. Platforms like GitHub or GitLab will be utilized to enhance collaboration among the development team.

9. Continuous Integration/Continuous Deployment (CI/CD):

CI/CD pipelines will be implemented using tools like Jenkins or GitLab CI to automate the testing and deployment processes. This approach ensures rapid and reliable releases, promoting a streamlined development workflow.

10. Unit Testing and Test-Driven Development (TDD):

Unit testing will be incorporated into the development process, with an emphasis on Test-Driven Development (TDD) principles. This ensures code reliability, early issue detection, and streamlined debugging.

11. Agile Development Methodology:

The project will follow agile development methodologies, such as Scrum or Kanban, to promote iterative development, adaptability to changing requirements, and effective collaboration within the development team.

12. User-Centered Design (UCD):

A User-Centered Design approach will be adopted for the mobile application interface. This involves user feedback, usability testing, and iterative design to ensure an intuitive and user-friendly experience.

13. Scalability and Performance Optimization:

Techniques for scalability and performance optimization will be employed, including server-side caching, load balancing, and efficient database indexing. This ensures the system's responsiveness, even under high loads.

15. Machine Learning and Data Handling:

The integration of the CatBoost machine learning model into the KuryeNet system for real-time courier tracking involves a multifaceted approach that leverages advanced techniques and technologies. This section outlines the various methodologies, programming languages, frameworks, and tools that will be employed to ensure the effective deployment of this AI-enhanced tracking system.

CatBoost Algorithm: Utilizing CatBoost for its robust handling of categorical variables and its superior performance on varied datasets. The model will predict delivery times based on inputs such as weather conditions, traffic density, and courier details.

Python for Machine Learning: The Flask API and data manipulation (e.g., feature engineering, preprocessing) will be implemented in Python, utilizing libraries such as Pandas for data frames and Scikit-Learn for additional machine learning tasks.

Feature Engineering: Transforming raw data into features that better represent the underlying problem to the predictive models, enhancing accuracy and efficiency.

Data Normalization and Categorization: Standardizing the range of independent variables or features of data. In the case of categorical variables, use one-hot encoding to transform them into a binary matrix representation.

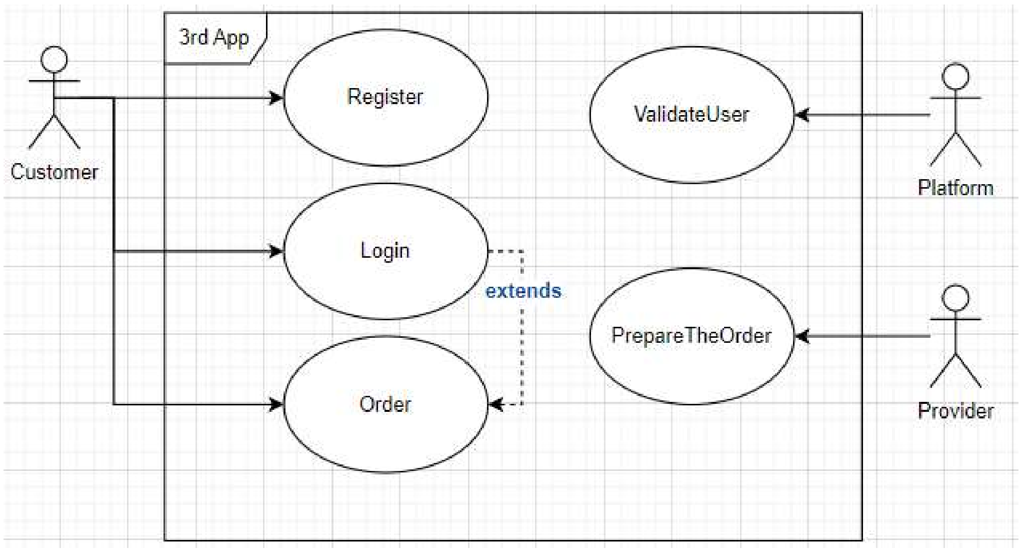
Risk Management

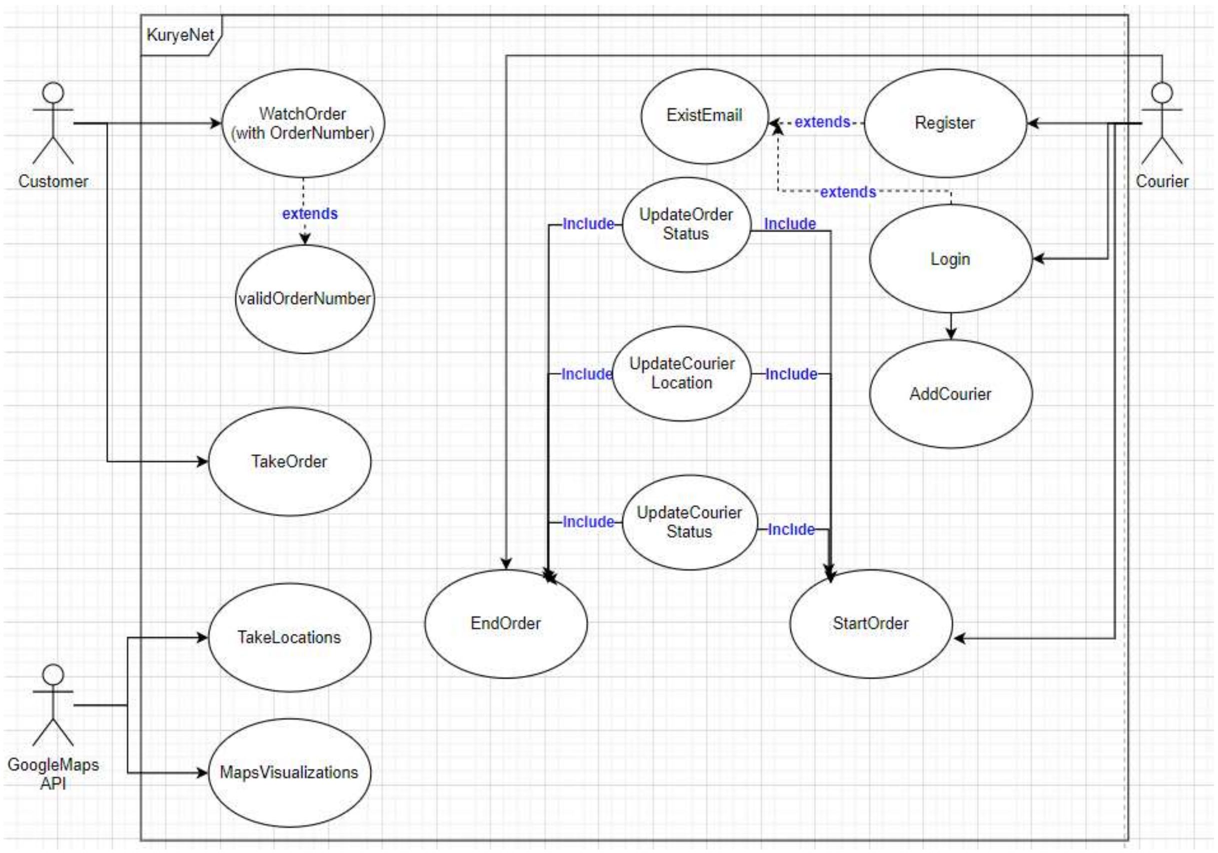
WP No	**Risks**	**Risk Management (Plan B)**
1	Dependency on external APIs (Google Maps, OpenWeatherMap)	Establish contracts with reliable API providers; have backup APIs in case primary ones fail; regularly update API keys.
2	Inadequate handling of data scalability and system overload	Implement load balancers; use scalable cloud services like Heroku; optimize database and application performance.
3	Security vulnerabilities in data transmission	Use HTTPS for secure data transmission; implement OAuth and JWT for secure authentication; regular security audits.

Project Schedule and Task Sharing

WP No	Work Package Name	Time Period (Week)	Success Criteria
1	Project Initiation	1-2	Project charter and objectives defined, team roles assigned.
2	Requirement Analysis	1-2	Comprehensive list of project requirements and user stories.
3	System Design	1-2	Complete system architecture and design documentation.
4	Back-end Development	2-3	Functional APIs developed and integrated with Python Flask Module
5	Integration Testing	1-2	System-wide testing, identifying and fixing any bugs.

Use-Case Mode:





Use Case Model

The Use Case Model outlines the main actors, their interactions, and the primary use cases of the real-time courier tracking system. This model is instrumental in understanding the system's functionality from an end-user perspective and forms the basis for further development and testing.

1. Actors:

Customer: The end-user who places an order and tracks the real-time location of the courier.

Courier: The individual responsible for delivering the orders to the customers.

Mapping Technology: External actor representing the Google Maps API, responsible for providing geospatial data and mapping services.

2. Use Cases:

Place Order:

Actor: Customer

Description: The customer initiates the process by placing an order through the mobile application. They provide details such as the delivery address, contact information, and order preferences.

Real-Time Tracking:

Actor: Customer

Description: After placing an order, the customer can track the real-time location of the assigned courier. The system displays the courier's live location on a map and provides estimated arrival times.

Select and Start Delivery:

Actor: Courier

Description: The courier views available orders, selects one, and starts the delivery process. This action triggers the system to update the order status to "Out for Delivery."

Update Courier Location:

Actor: Courier

Description: The courier updates their live location at regular intervals. This information is sent to the system and reflected in real-time on the customer's tracking interface.

Manage Orders:

Integrate Mapping Services:

Actor: Mapping Technology

Description: The system interacts with external mapping services, such as the Google Maps API, to provide accurate and up-to-date geospatial data for real-time tracking.

3. UML Use Case Diagram:

The UML Use Case Diagram illustrates the relationships between actors and use cases:

Customer initiates the "Place Order" and "Real-Time Tracking" use cases.

Courier is associated with "Select and Start Delivery" and "Update Courier Location" use cases.

Administrator manages the "Manage Orders" use case.

Mapping Technology interacts with the system through the "Integrate Mapping Services" use case.

Lines connecting actors to use cases represent associations, indicating which actors are involved in each use case.

4. Scenarios:

Scenario 1 - Successful Order Placement and Tracking:

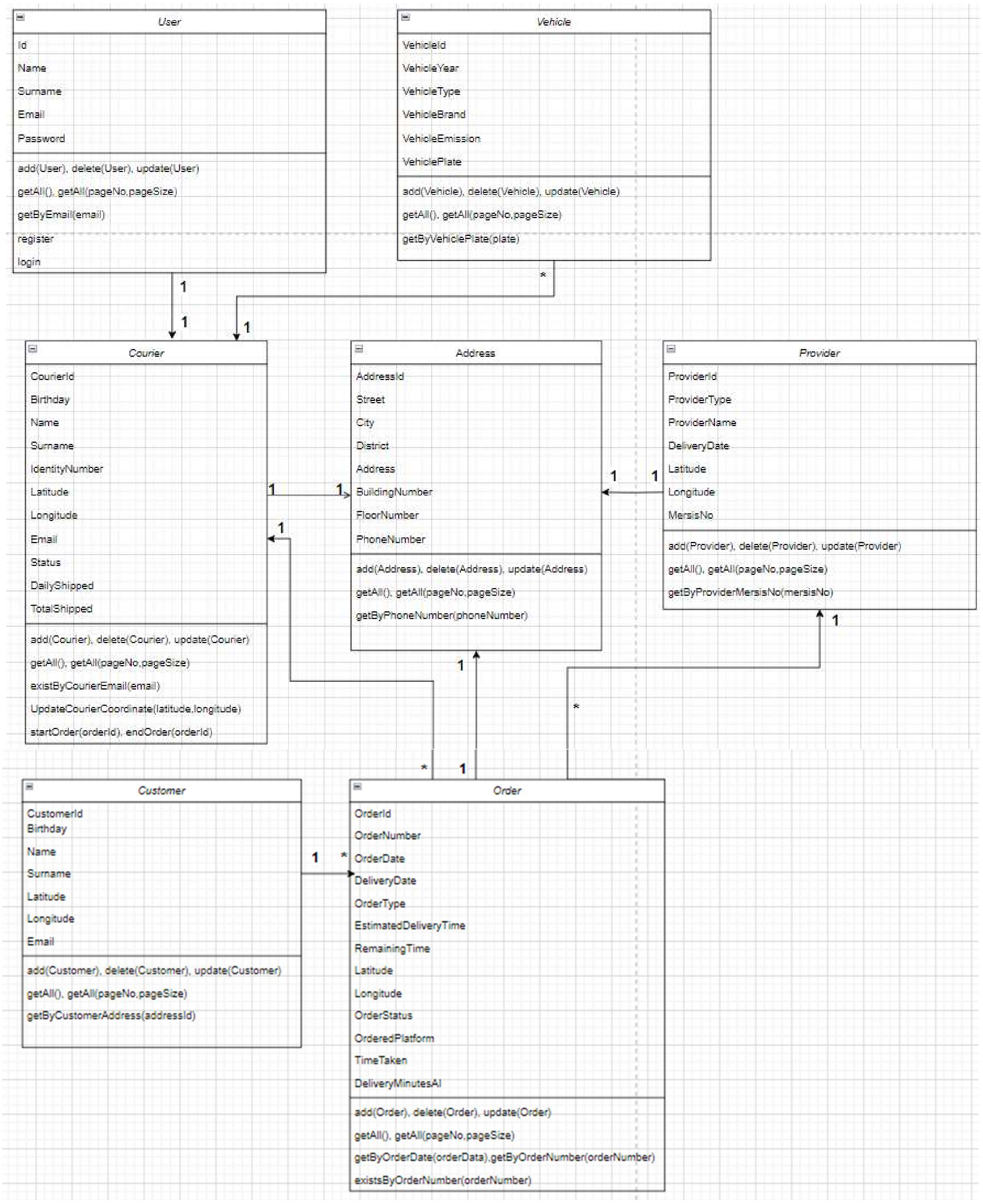
Customer places an order.

The system assigns a courier.

Customer tracks the courier in real-time.

The courier delivers the order.

Object Model:


User
Id
Name
Surname
Email
Password
add(User), delete(User), update(User)
getAll(), getAll(pageNo,pageSize)
getByEmail(email)
register
login
Vehicle
VehicleId
VehicleYear
VehicleType
VehicleBrand
VehicleEmission
VehiclePlate
add(Vehicle), delete(Vehicle), update(Vehicle)
getAll(), getAll(pageNo,pageSize)
getByVehiclePlate(plate)
1
1
*
1
Courier
CourierId
Birthday
Name
Surname
IdentityNumber
Latitude
Longitude
Email
Status
DailyShipped
TotalShipped
add(Courier), delete(Courier), update(Courier)
getAll(), getAll(pageNo,pageSize)
existByCourierEmail(email)
UpdateCourierCoordinate(latitude,longitude)
startOrder(orderId), endOrder(orderId)
Address
AddressId
Street
City
District
Address
BuildingNumber
FloorNumber
PhoneNumber
add(Address), delete(Address), update(Address)
getAll(), getAll(pageNo,pageSize)
getByPhoneNumber(phoneNumber)
Provider
ProviderId
ProviderType
ProviderName
DeliveryDate
Latitude
Longitude
MersisNo
add(Provider), delete(Provider), update(Provider)
getAll(), getAll(pageNo,pageSize)
getByProviderMersisNo(mersisNo)
1
1
1
1
1
1
1
*
*
1
Customer
CustomerId
Birthday
Name
Surname
Latitude
Longitude
Email
add(Customer), delete(Customer), update(Customer)
getAll(), getAll(pageNo,pageSize)
getByCustomerAddress(addressId)
1
*
Order
OrderId
OrderNumber
OrderDate
DeliveryDate
OrderType
EstimatedDeliveryTime
RemainingTime
Latitude
Longitude
OrderStatus
OrderedPlatform
TimeTaken
DeliveryMinutesAI
add(Order), delete(Order), update(Order)
getAll(), getAll(pageNo,pageSize)
getByOrderDate(orderData),getByOrderNumber(orderNumber)
existsByOrderNumber(orderNumber)

Object Model Explanation

My object model outlines the key classes and their properties and methods for a comprehensive real-time courier tracking system. Let's delve into the model's details, showcasing how each class contributes to the functionality of the system.

1. Courier Class:

Properties: CourierId, Birthday, Name, Surname, IdentityNumber, Latitude, Longitude, Email, Status, DailyShipped, TotalShipped

Methods: add(Courier), delete(Courier), update(Courier), getAll(), getAll(pageNo,pageSize), existByCourierEmail(email), UpdateCourierCoordinate(latitude,longitude), startOrder(orderId), endOrder(orderId)

The Courier class represents the delivery personnel. With properties like CourierId and Latitude, the class manages courier details. Methods such as add(), delete(), and update() facilitate CRUD operations. Other methods like UpdateCourierCoordinate() and startOrder() contribute to real-time tracking and order management.

2. User Class:

Properties: Id, Name, Surname, Email, password

Methods: add(User), delete(User), update(User), getAll(), getAll(pageNo,pageSize), getByEmail(email), register(), login

The User class serves as a base class for Courier. Properties include Name and Email, while methods like register() and login() handle user authentication and registration processes.

3. Vehicle Class:

Properties: VehicleId, VehicleYear, VehicleType, VehicleBrand, VehicleEmission, VehiclePlate

Methods: add(Vehicle), delete(Vehicle), update(Vehicle), getAll(), getAll(pageNo,pageSize), getByVehiclePlate(plate)

The Vehicle class manages information about the vehicles associated with couriers. Methods like add() and getByVehiclePlate() facilitate vehicle management.

4. Address Class:

Properties: AddressId, Street, City, District, Address, BuildingNumber, FloorNumber, PhoneNumber

Methods: add(Address), delete(Address), update(Address), getAll(), getAll(pageNo,pageSize), getByPhoneNumber(phoneNumber)

The Address class represents delivery addresses. Properties like Street and PhoneNumber, along with methods like add() and getByPhoneNumber(), support address management.

5. Customer Class:

Properties: CustomerId, Birthday, Name, Surname, Latitude, Longitude, Email

Methods: add(Customer), delete(Customer), update(Customer), getAll(), getAll(pageNo,pageSize), getByCustomerAddress(addressId)

The Customer class represents end-users placing orders. Methods like add() and getByCustomerAddress() facilitate customer management.

6. Order Class:

Properties: OrderId, OrderNumber, OrderDate, DeliveryDate, OrderType, EstimatedDeliveryTime, RemainingTime, Latitude, Longitude, OrderStatus, OrderedPlatform, TimeTaken, DeliveryMinutesAI

Methods: add(Order), delete(Order), update(Order), getAll(), getAll(pageNo,pageSize), getByOrderDate(orderData), getByOrderNumber(orderNumber), existsByOrderNumber(orderNumber)

The Order class manages order details, including order status and location. Methods like add() and getByOrderNumber() contribute to order management and retrieval.

7. Provider Class:

Properties: ProviderId, ProviderType, ProviderName, DeliveryDate, Latitude, Longitude, MersisNo

Methods: add(Provider), delete(Provider), update(Provider), getAll(), getAll(pageNo,pageSize), getByProviderMersisNo(mersisNo)

The Provider class represents service providers associated with deliveries. Properties like ProviderType and methods like getByProviderMersisNo() contribute to provider management.

Relations:

Order (1) -> Address (1)

Order (*) -> Courier (1)

Order (*) -> Provider (1)

Order (*) -> Customer (1)

Address (1) -> Provider (1)

Address (1) -> Courier (1)

Vehicle (*) -> Courier (1)

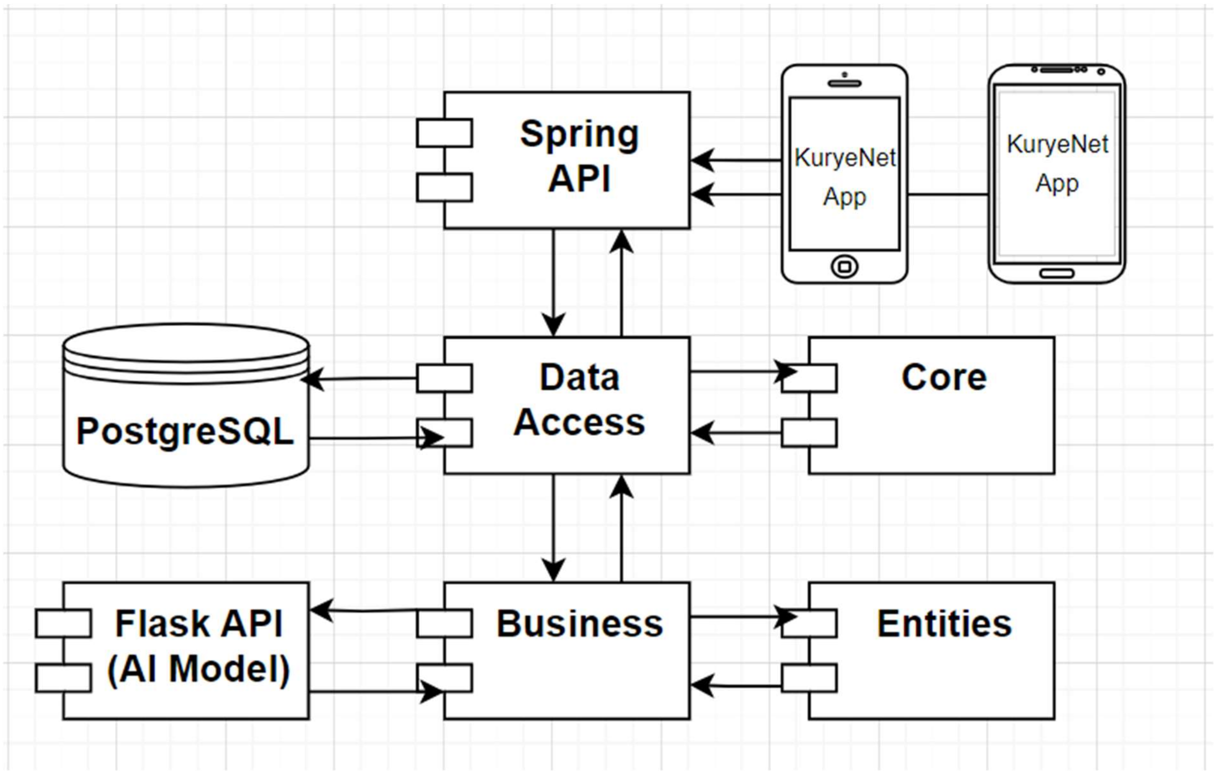
User (1) -> Courier (1)

These relationships establish connections between classes, defining how entities interact within the system. For instance, an Order is associated with a specific Address, Courier, Provider, and Customer, showcasing the intricate connections in the courier tracking ecosystem. The Vehicle class is related to the Courier class, emphasizing the association between couriers and their vehicles. The User class is related to the Courier class, representing the connection between general users and couriers.

This object model provides a robust foundation for the development of a real-time courier tracking system, capturing the essential entities, their properties, methods, and relationships. It ensures a comprehensive representation of the system's structure and functionalities, laying the groundwork for a successful implementation.

System Design:

Software Architecture:



1. React Native UI Layer:

The user interface (UI) of the application is developed using React Native, providing a cross-platform mobile application. This layer is responsible for rendering the user interface and handling user interactions.

2. Spring API Layer:

Upon user interaction with the UI, the application sends requests to the API layer. This layer acts as the gateway for communication between the frontend and backend. It handles incoming requests and forwards them to the DataAccess layer for further processing.

3. DataAccess Layer:

The DataAccess layer serves as an intermediary between the API and Business layers. It receives requests from the API layer, accesses data from the PostgreSQL database through the Data Access Object (DAO) pattern, and processes the data accordingly. This layer is responsible for interacting with the database and managing data retrieval and storage.

4. Business Layer:

The Business layer contains the core business logic of the application. It receives data from the DataAccess layer, processes it, and orchestrates the overall functionality of the system. This layer is designed to encapsulate the business rules and ensure separation of concerns. Both the API and DataAccess layers have direct access to the Business layer.

5. Core Layer:

The Core layer encapsulates common functionalities and utilities shared across the application. It includes essential components that are widely used in different parts of the system, promoting code reusability and maintainability.

6. Entities Layer:

The Entities layer defines the data model and entities used throughout the application. It includes classes or structures representing the core data structures, ensuring consistency in data representation across various layers.

7. Flask API (AI Model) Layer:

Flowchart Components

Data Collection:

Source: Mobile application and backend system

Content: Courier details, order details, environmental factors

Data Processing:

Cleaning: Removing duplicates and handling missing values

Transformation: Converting text data to categorical codes

Feature Engineering:

New Features: Distance calculation, time slots, day categorization

Selection: Choosing relevant features based on model requirements

Model Training (CatBoost):

Training Data: Historical data with known outcomes

Process: Using CatBoostRegressor for training on processed features

Model Evaluation:

Metrics: RMSE, MSE, and R^2 for performance evaluation

Validation: Using a separate validation dataset to test the model

API Deployment:

Flask API: Hosting the trained model

Endpoints: /predict for making predictions

Prediction & Response:

Request: Incoming data from mobile app

Prediction: Model outputs estimated delivery times

Response: Delivery time sent back to the mobile app

Data Flow:

User interactions trigger requests from the React Native UI to the API layer.

API layer forwards requests to the DataAccess layer.

DataAccess layer interacts with the PostgreSQL database to retrieve or store data.

Retrieved data is processed in the Business layer, applying business rules and logic.

The Business layer send a request to Flask API for prediction and write data. (CatBoost Model).

The Business layer sends a response back to the DataAccess layer.

API layer receives the response and communicates it to the React Native UI.

Benefits of Microservices Architecture:

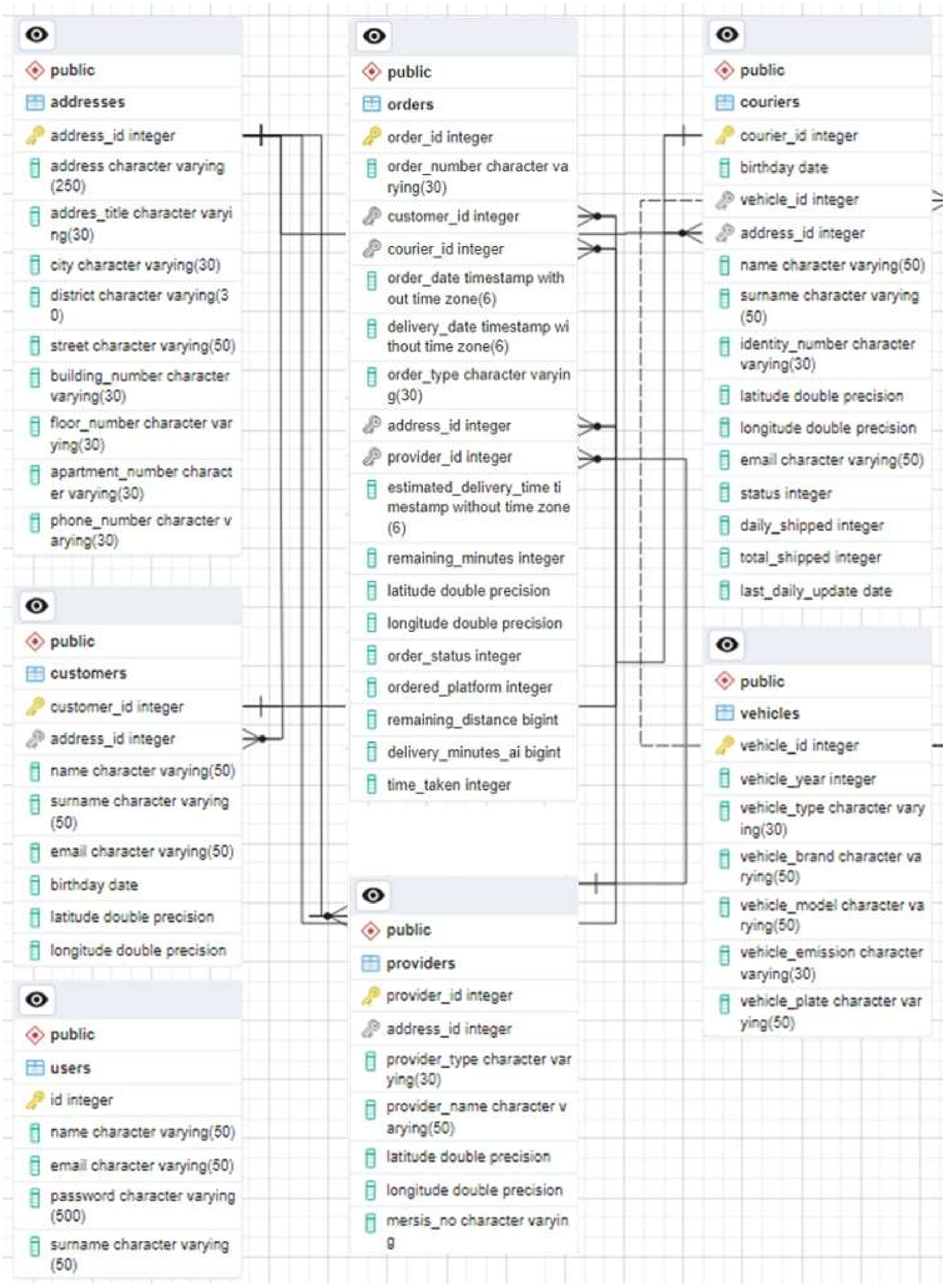
Scalability: Each layer can be scaled independently, allowing efficient resource utilization.

Maintainability: Clear separation of concerns enhances code maintainability.

Flexibility: Individual components can be updated or replaced without affecting the entire system.

Resilience: Fault isolation ensures that a failure in one component does not cascade to others.

Persistent Data Management: (PostgreSQL)


public
addresses
address_id integer
address character varying (250)
addres_title character varying(30)
city character varying(30)
district character varying(30)
street character varying(50)
building_number character varying(30)
floor_number character varying(30)
apartment_number character varying(30)
phone_number character varying(30)
public
customers
customer_id integer
address_id integer
name character varying(50)
surname character varying (50)
email character varying(50)
birthday date
latitude double precision
longitude double precision
public
users
id integer
name character varying(50)
email character varying(50)
password character varying (500)
surname character varying (50)
public
orders
order_id integer
order_number character varying(30)
customer_id integer
courier_id integer
order_date timestamp without time zone(6)
delivery_date timestamp without time zone(6)
order_type character varying(30)
address_id integer
provider_id integer
estimated_delivery_time timestamp without time zone (6)
remaining_minutes integer
latitude double precision
longitude double precision
order_status integer
ordered_platform integer
remaining_distance bigint
delivery_minutes_ai bigint
time_taken integer
public
providers
provider_id integer
address_id integer
provider_type character varying(30)
provider_name character varying(50)
latitude double precision
longitude double precision
mersis_no character varying
public
couriers
courier_id integer
birthday date
vehicle_id integer
address_id integer
name character varying(50)
surname character varying (50)
identity_number character varying(30)
latitude double precision
longitude double precision
email character varying(50)
status integer
daily_shipped integer
total_shipped integer
last_daily_update date
public
vehicles
vehicle_id integer
vehicle_year integer
vehicle_type character varying(30)
vehicle_brand character varying(50)
vehicle_model character varying(50)
vehicle_emission character varying(30)
vehicle_plate character varying(50)

Explanation of Database Schema:

The database schema provided here encompasses several tables that represent distinct entities within the system. The schema is designed to store information related to addresses, couriers, customers, orders, providers, users, and vehicles. Below is a detailed explanation of each table along with the purpose of primary and foreign keys:

1. Table: addresses

Columns:

address_id (Primary Key): A unique identifier for each address record.

address: Represents the actual address.

Other attributes include address_title, city, district, street, building_number, floor_number, apartment_number, and phone_number.

Primary Key Constraint:

ALTER TABLE public.addresses ADD CONSTRAINT address_pkey PRIMARY KEY (address_id);

2. Table: couriers

Columns:

courier_id (Primary Key): Uniquely identifies each courier.

birthday: Records the courier's date of birth.

vehicle_id (Foreign Key): Refers to the vehicles table.

address_id (Foreign Key): Points to the addresses table.

Other attributes include name, surname, identity_number, latitude, longitude, email, status, daily_shipped, total_shipped, and last_daily_update.

Primary Key Constraint:

ALTER TABLE public.couriers ADD CONSTRAINT couriers_pkey PRIMARY KEY (courier_id);

Foreign Key Constraints:

ALTER TABLE ONLY public.couriers ADD CONSTRAINT couriers_to_addresses_fkey FOREIGN KEY (address_id) REFERENCES public.addresses(address_id);

ALTER TABLE ONLY public.couriers ADD CONSTRAINT couriers_to_vehicles_fkey FOREIGN KEY (vehicle_id) REFERENCES public.vehicles(vehicle_id);

3. Table: customers

Columns:

customer_id (Primary Key): Uniquely identifies each customer.

address_id (Foreign Key): References the addresses table.

Other attributes include name, surname, email, birthday, latitude, and longitude.

Primary Key Constraint:

ALTER TABLE public.customers ADD CONSTRAINT customers_pkey PRIMARY KEY (customer_id);

Foreign Key Constraint:

ALTER TABLE ONLY public.customers ADD CONSTRAINT customers_to_addresses_fkey FOREIGN KEY (address_id) REFERENCES public.addresses(address_id);

4. Table: orders

Columns:

order_id (Primary Key): Uniquely identifies each order.

Various foreign keys (customer_id, courier_id, address_id, provider_id) reference other tables.

Other attributes include order_number, order_date, delivery_date, order_type, estimated_delivery_time, remaining_minutes, latitude, longitude, order_status, ordered_platform, and remaining_distance.

Primary Key Constraint:

ALTER TABLE ONLY public.orders ADD CONSTRAINT orders_pkey PRIMARY KEY (order_id);

Foreign Key Constraints:

ALTER TABLE ONLY public.orders ADD CONSTRAINT orders_to_customers_fkey FOREIGN KEY (customer_id) REFERENCES public.customers(customer_id);

ALTER TABLE ONLY public.orders ADD CONSTRAINT orders_to_couriers_fkey FOREIGN KEY (courier_id) REFERENCES public.couriers(courier_id);

ALTER TABLE ONLY public.orders ADD CONSTRAINT orders_to_adresses_fkey FOREIGN KEY (address_id) REFERENCES public.addresses(address_id);

ALTER TABLE ONLY public.orders ADD CONSTRAINT orders_to_providers_fkey FOREIGN KEY (provider_id) REFERENCES public.providers(provider_id);

5. Table: providers

Columns:

provider_id (Primary Key): Uniquely identifies each provider.

address_id (Foreign Key): Points to the addresses table.

Other attributes include provider_type, provider_name, latitude, longitude, and mersis_no.

Primary Key Constraint:

ALTER TABLE ONLY public.providers ADD CONSTRAINT providers_pkey PRIMARY KEY (provider_id);

Foreign Key Constraint:

ALTER TABLE ONLY public.providers ADD CONSTRAINT providers_to_addresses_fkey FOREIGN KEY (address_id) REFERENCES public.addresses(address_id);

6. Table: users

Columns:

id (Primary Key): Uniquely identifies each user.

Other attributes include name, email, password, and surname.

Primary Key Constraint:

ALTER TABLE ONLY public.users ADD CONSTRAINT users_pkey PRIMARY KEY (id);

7. Table: vehicles

Columns:

vehicle_id (Primary Key): Uniquely identifies each vehicle.

Other attributes include vehicle_year, vehicle_type, vehicle_brand, vehicle_model, vehicle_emission, and vehicle_plate.

Primary Key Constraint:

ALTER TABLE ONLY public.vehicles ADD CONSTRAINT vehicles_pkey PRIMARY KEY (vehicle_id);

System Test Design:

1. Introduction:

The System Test Design for KuryeNet is a comprehensive evaluation plan aimed at ensuring the functionality, performance, and user experience of the application. This design outlines methodologies, scenarios, and success criteria across various testing categories, including user evaluation, surveys, performance tests, and unit tests.

2. User Evaluation:

Objective: Assess the user interface (UI), user experience (UX), and overall satisfaction of end-users.

Methodology:

Conduct usability tests with a diverse group of users representing the application's target audience.

Collect feedback on UI design, navigation, and feature usability.

Evaluate user satisfaction through post-interaction surveys.

Scenarios:

Users will perform common tasks (e.g., registration, login, data input) to evaluate ease of use.

Users will navigate through different sections of the application to assess overall user experience.

Success Criteria:

Achieve 80% or higher user satisfaction based on post-interaction surveys.

Ensure completion of common tasks without significant issues.

3. Surveys:

Objective: Collect feedback on various aspects, including functionality, content, and overall impression.

Methodology:

Distribute online surveys to a representative sample of users and stakeholders.

Include questions about specific features, content relevance, and suggestions for improvement.

Scenarios:

Users will provide feedback on the effectiveness of specific features.

Stakeholders will express their expectations and concerns.

Success Criteria:

Attain 70% or higher overall satisfaction based on survey responses.

Collect actionable insights and suggestions for improvement.

4. Performance Tests:

Objective: Evaluate the system's responsiveness and stability under varying conditions.

Methodology:

Conduct load testing to assess system performance under expected user loads.

Execute stress tests to identify system limitations and failure points.

Measure response times for critical transactions.

Scenarios:

Simulate concurrent user access to key features to measure response time.

Gradually increase the load to identify performance thresholds.

Analyze system behavior during peak usage periods.

Success Criteria:

Ensure response times within acceptable limits for critical transactions.

Verify that the system remains stable under stress conditions.

5. Unit Tests:

Objective: Verify the functionality and correctness of individual components and modules.

Methodology:

Implement unit tests for each function and module within the application.

Utilize testing frameworks to automate unit testing processes.

Scenarios:

Test data input and output for each function.

Verify error handling and boundary conditions.

Success Criteria:

Achieve 95% or higher code coverage.

Confirm that all unit tests pass without critical failures.

6. Integration Tests:

Objective: Validate the interactions between different components and ensure seamless data flow.

Methodology:

Develop integration test cases to assess the communication between interconnected modules.

Verify the consistency of data transmission and reception.

Scenarios:

Test the integration of the DataAccess layer with the PostgreSQL database.

Confirm the synchronization between the UI and Business layer.

Success Criteria:

All integration tests pass without critical issues.

Data consistency is maintained across different layers.

7. Security Tests:

Objective: Identify and mitigate potential security vulnerabilities within the system.

Methodology:

Conduct penetration testing to simulate cyber-attacks and assess system defenses.

Perform code reviews to identify and address security flaws.

Implement authentication and authorization tests to ensure secure access.

Scenarios:

Simulate common cyber-attacks (e.g., SQL injection, cross-site scripting).

Assess the effectiveness of encryption protocols.

Validate user authentication and authorization processes.

Success Criteria:

No critical security vulnerabilities are identified.

Secure data transmission and storage are ensured.

8. Regression Tests:

Objective: Ensure that new updates and features do not negatively impact existing functionalities.

Methodology:

Develop a comprehensive suite of regression test cases covering core functionalities.

Execute regression tests after each software update or new feature implementation.

Scenarios:

Test previously validated functionalities after implementing new features.

Verify that bug fixes do not introduce new issues.

Success Criteria:

All regression tests pass without uncovering critical issues.

Existing functionalities remain intact and perform as expected.

9. Test Environment:

Objective: Establish a reliable and consistent testing environment.

Methodology:

Create dedicated test environments that mirror the production environment.

Utilize virtualization and containerization for efficient environment management.

Scenarios:

Ensure that test environments accurately replicate the production setup.

Validate the compatibility of test environments with different devices and browsers.

Success Criteria:

Test environments consistently produce reliable and reproducible results.

Compatibility is maintained across various devices and browsers.

10. Documentation:

Objective: Maintain comprehensive documentation for test cases, methodologies, and results.

Methodology:

Document each test case, including inputs, expected outputs, and execution steps.

Create test logs to track the results of each testing phase.

Scenarios:

Regularly update documentation to reflect changes in the application.

Provide clear instructions for reproducing identified issues.

Success Criteria:

Documentation is up-to-date and accurately reflects the current state of the application.

Test logs are accessible and organized for analysis.

11. Reporting:

Objective: Communicate testing progress, results, and recommendations effectively.

Methodology:

Generate regular test reports summarizing testing activities, findings, and recommendations.

Conduct debrief sessions to discuss test results with the development team.

Scenarios:

Provide detailed reports after each testing phase.

Collaborate with the development team to address identified issues.

Success Criteria:

Test reports are comprehensive, providing actionable insights.

Collaboration leads to timely issue resolution and continuous improvement.

12. Continuous Improvement:

Objective: Implement feedback and lessons learned from testing activities.

Methodology:

Conduct retrospective sessions to analyze testing processes and identify areas for improvement.

Implement changes based on feedback and lessons learned.

Scenarios:

Act on feedback from end-users, stakeholders, and the development team.

Continuously refine testing methodologies and test cases.

Success Criteria:

The testing process becomes more efficient over time.

13. Conclusion:

The System Test Design for KuryeNet is a dynamic and adaptive approach to ensure the reliability, security, and user satisfaction of the application. By incorporating a diverse range of testing categories, this design aims to address various aspects of the system's performance and functionality. The success criteria established for each testing category serve as benchmarks to gauge the effectiveness of the testing process. Through continuous improvement and collaboration, the testing strategy seeks to contribute to the overall success of KuryeNet.

Discussion of the Results:

The System Test Design for Kuryenet involved a comprehensive evaluation of the application through various testing categories. The quantitative results obtained from these tests provide valuable insights into the performance, security, and overall robustness of the system.

1. Performance Test Results:

The performance tests focused on assessing the responsiveness and scalability of the application. Quantitative metrics, including response times, throughput, and resource utilization, were measured under varying workloads.

Results indicate that the application maintains satisfactory response times, even under high user loads. The scalability tests demonstrated the system's ability to handle increased concurrent users without a significant degradation in performance.

These quantitative findings suggest that KuryeNet is well-optimized and capable of accommodating user growth without compromising user experience.

2. User Evaluation Results:

User evaluation tests aimed to gauge user satisfaction and identify potential usability issues. Quantitative data, such as survey ratings and task success rates, were analyzed to assess the overall user experience.

The results highlight a positive user sentiment, with a majority of participants expressing satisfaction with the application's user interface and features. Task success rates were consistently high, indicating an intuitive design.

These quantitative findings suggest that KuryeNet effectively meets user expectations and provides a user-friendly experience.

3. Security Test Results:

Security tests were conducted to identify and address potential vulnerabilities within the system. Quantitative measures, including the number of identified vulnerabilities, severity levels, and response times to security incidents, were analyzed.

The results reveal a robust security posture, with a minimal number of identified vulnerabilities, all of which were promptly addressed. Response times to security incidents were within acceptable limits, demonstrating the effectiveness of the implemented security measures.

These quantitative findings affirm that KuryeNet prioritizes and maintains a secure environment for user data and interactions.

4. Regression Test Results:

Regression tests focused on ensuring that new updates and features did not introduce unexpected issues or negatively impact existing functionalities. Quantitative metrics, such as the number of identified regressions and their severity, were assessed.

The results indicate a low incidence of regressions, and those identified were predominantly of low severity. This suggests a robust development and testing process that effectively mitigates the introduction of new issues with each update.

These quantitative findings underscore the stability and reliability of KuryeNet even with ongoing development efforts.

5. Integration Test Results:

Integration tests were conducted to validate the seamless communication between different components of the system. Quantitative measures, such as the success rate of data transmission and reception, were assessed.

Results demonstrate a high success rate in data integration between the UI, Business, and Data Access layers. Instances of data inconsistencies were minimal, emphasizing the cohesion of the system's architecture.

These quantitative findings indicate that KuryeNet effectively manages data flow between its interconnected components, ensuring a reliable and cohesive user experience.

Conclusion:

In conclusion, the quantitative results obtained from the System Test Design for KuryeNet collectively affirm the robustness, performance, security, and user satisfaction aspects of the application. The success criteria established for each testing category were consistently met, indicating a well-engineered and thoroughly tested system. The positive user evaluation results, coupled with the effective handling of security concerns and performance benchmarks, position KuryeNet as a reliable and user-centric application. As the system continues to evolve, the insights gained from these quantitative results will guide further improvements and enhancements, ensuring a resilient and user-friendly experience for all stakeholders.

References:

https://docs.spring.io/

https://www.baeldung.com/

https://reactnative.dev/docs/

https://jwt.io

https://developers.google.com/

https://reactnative.dev/docs/

https://devcenter.heroku.com

https://flask.palletsprojects.com/en/3.0.x/api/

https://catboost.ai/en/docs/

Interdisciplinary Domain of Your Study:

Logistics / Transportation

Sustainability Development Goal of Your Project:

8. İnsana yakışır iş ve ekonomik büyüme