Team members:

1. **Kamal Abdullayev**
2. **Feray Gulu-zada**
3. **Nika Mgaloblishvili**

Task #1 (1.5 points): Provide a description of the system you will be modelling in this course.

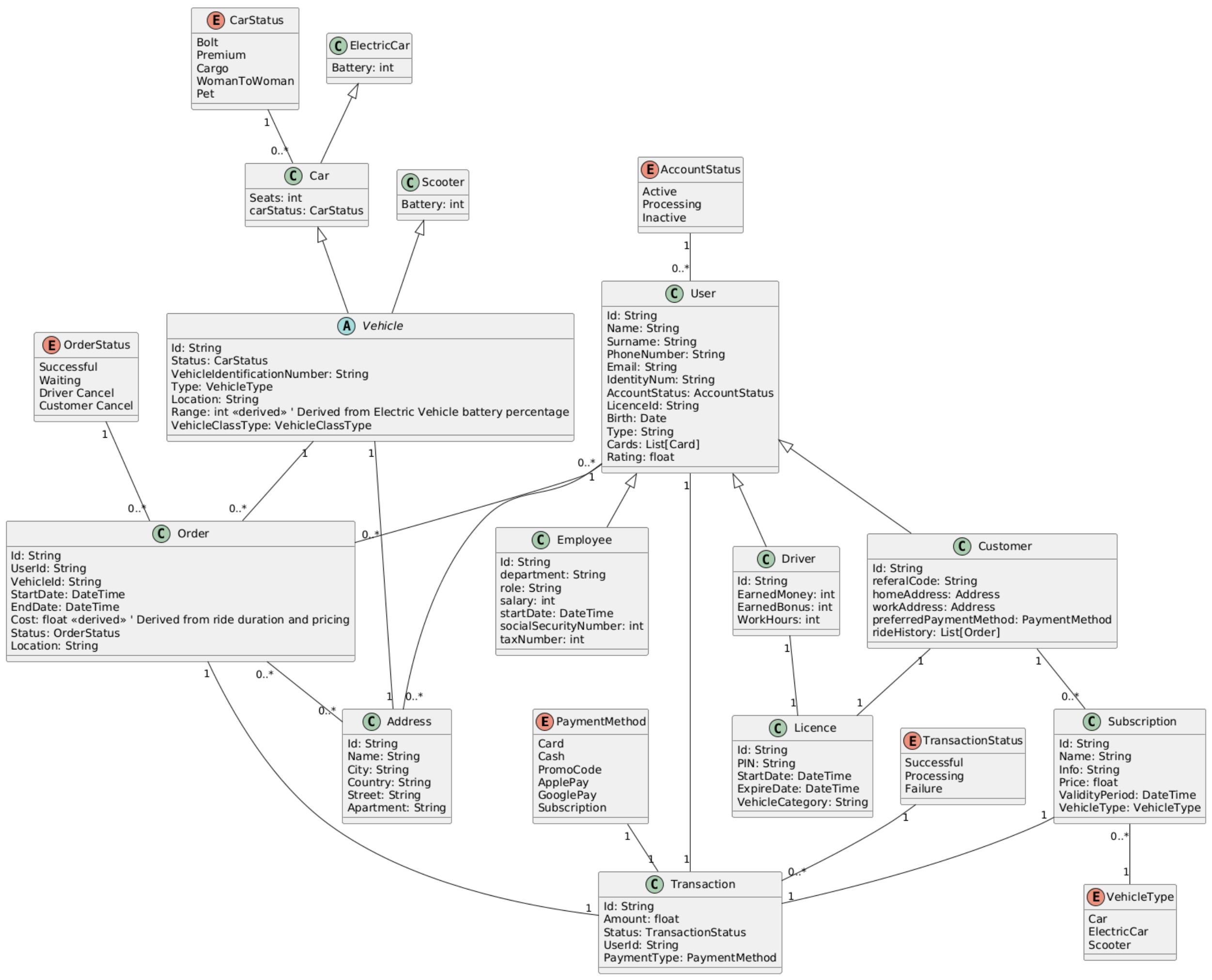
The system being modeled is a ride-hailing platform, similar to Bolt, comprising users (customers and drivers), vehicles, orders (rides), transactions, and related elements.

User class includes attributes like ID, name, contact details, and account status. Users can either be Customers or Drivers, with drivers having specific attributes like earned money and work hours. Each driver must have a License and own vehicle.

Vehicle types include Cars, Scooters, and ElectricCars, each with specific attributes like seats or battery level. Orders are linked to customers, vehicles, and transactions, with order and transaction statuses defined by enumerations.

Payment is handled via Transactions linked to a PaymentMethod (e.g., card or cash) or subscription. Addresses are linked to users, and relationships between classes are defined by multiplicity, such as one user having many orders but one order belonging to one user.

Task #2 (3 points): Provide a class model for the system you described in Task #1.



Task #3 (0.5 point): What is the environment, which you choose to design your class diagram? What were the decisive factors to give the preference to this environment?

We initially worked on Excalidraw to create quick and intuitive drafts of the class diagrams. Excalidraw offers an easy drawing interface, which makes it comfortable for brainstorming and rough sketching of the class structure without any technical constraints. However, for more formal and detailed class diagrams,we later transitioned to PlantUML.

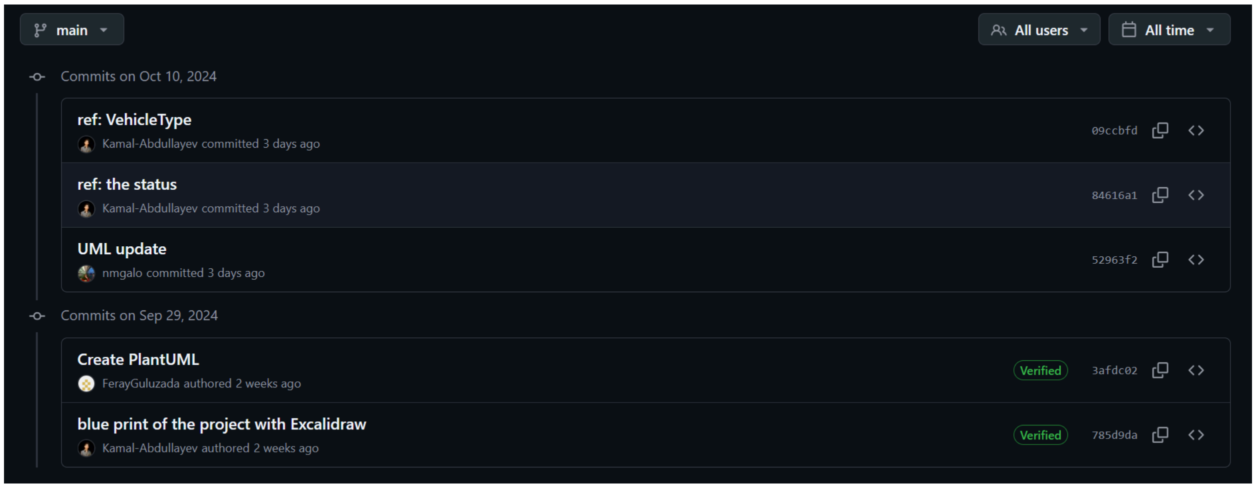
The decisive factors for choosing **Excalidraw** were:

1. Simple and intuitive interface
2. Fast for initial drafts and rough ideas
3. Easy to adjust or iterate during the early phases of design

When the class diagrams required more structure and detail, we moved to **PlantUML** due to:

1. Text-based modeling, which allows version control
2. Ability to generate diagrams directly from textual descriptions
3. Easier to maintain and modify as the model evolves

Task #4 (1 point): Provide evidence that the work was conducted in a team (e.g., in the form of a link to the VCS; the screenshot of the above etc.).



Commits on Oct 10, 2024

ref: VehicleType
Kamal-Abdullayev committed 3 days ago
09ccbfd

ref: the status
Kamal-Abdullayev committed 3 days ago
84616a1

UML update
nmgalo committed 3 days ago
52963f2

Commits on Sep 29, 2024

Create PlantUML
FerayGuluzada authored 2 weeks ago
Verified
3afdc02

blue print of the project with Excalidraw
Kamal-Abdullayev authored 2 weeks ago
Verified
785d9da

Task #5 (3 points*): Explore and document Generative AI capabilities in modelling class model (in addition to Task#2, but not instead!).

Design the class diagram for your system using any Generative AI tools. When selecting the tool, seek for specialized tools for the given purpose. Justify your choice.

Provide the prompt you have used to obtain the result.

Compare the model produced by Generative AI tools with the one designed by you - assess both critically. Was it possible to achieve the same result that you obtained in Task #2? If not, what were the differences? What are strengths and weaknesses of GenAI tools in addressing this task?

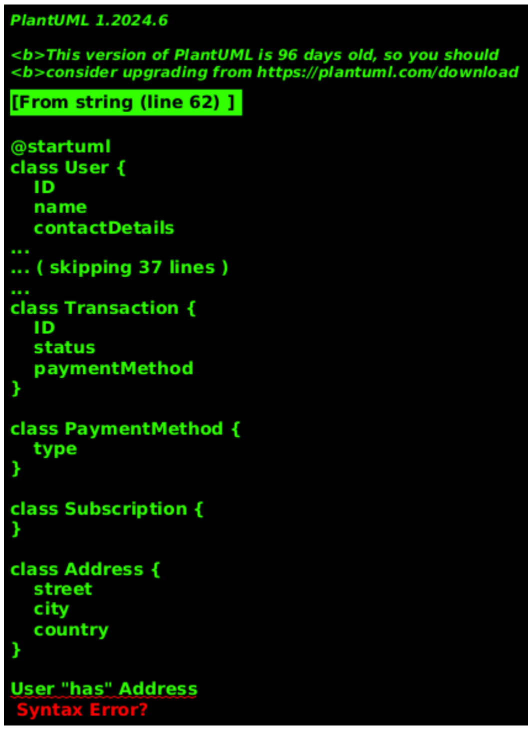
Had the use of GenAI tools allowed you to improve either the description (Task#1) or the model (Task#2)?

(*+1 point (bonus point)): if more than one tool has been tried and critically assessed

To explore the capabilities of Generative AI in systems modeling we were curious if AI like ChatGPT and Gemini could generate a PlantUML code good enough to compare to the one we did. It shouldn't be too difficult since PlantUML can be generated by typing but the AIs did not do their job so well.

The Prompt: We used our description from Task 1 as prompt for both Gemini and ChatGPT

Gemini provided the code but it wasn't even correct enough to run:



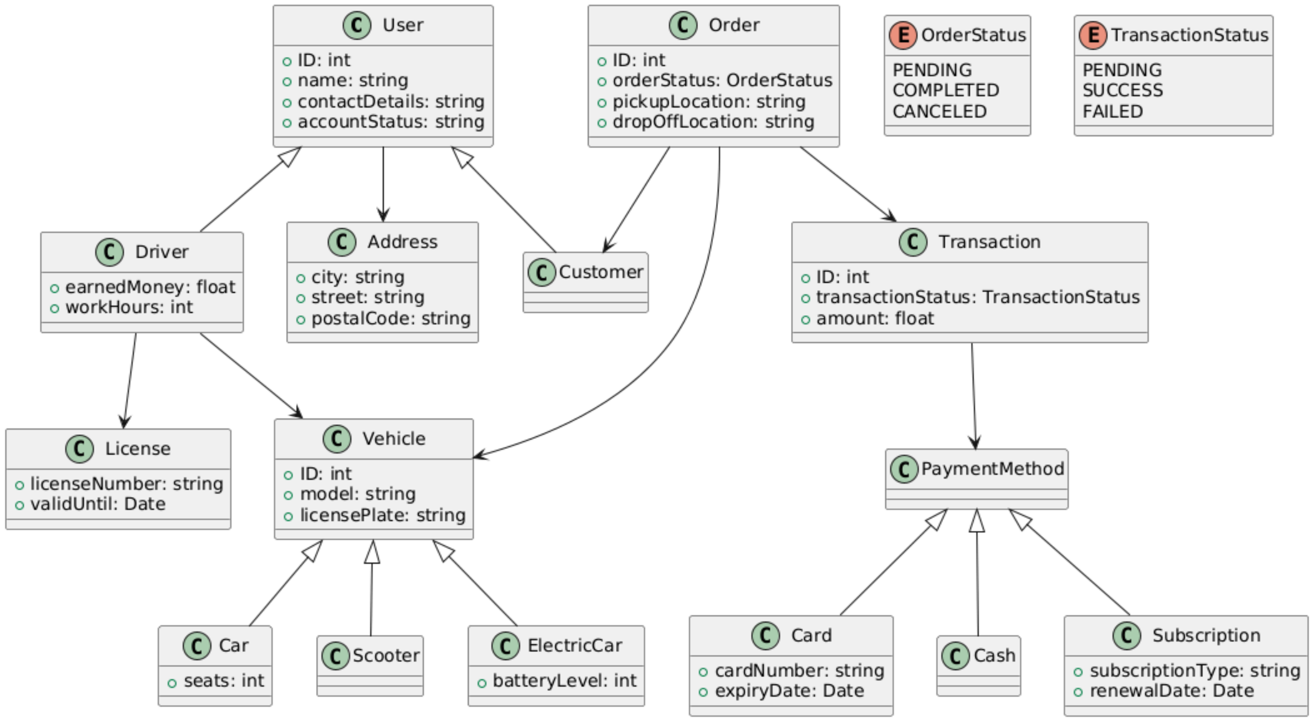
```
PlantUML 1.2024.6

<b>This version of PlantUML is 96 days old, so you should
<b>consider upgrading from https://plantuml.com/download
[From string (line 62) ]

@startuml
class User {
   ID
   name
   contactDetails
...
... ( skipping 37 lines )
...
class Transaction {
   ID
   status
   paymentMethod
}

class PaymentMethod {
   type
}

class Subscription {
}

class Address {
   street
   city
   country
}

User "has" Address
 Syntax Error?
```

```
@startuml
class User {
    ID
    name
    contactDetails
    accountStatus
}

class Customer extends User {
}

class Driver extends User {
    earnedMoney
    workHours
}

class Vehicle {
    ID
    type
}

class Car extends Vehicle {
    seats
}

class Scooter extends Vehicle {
}

class ElectricCar extends Vehicle {
    batteryLevel
}
```

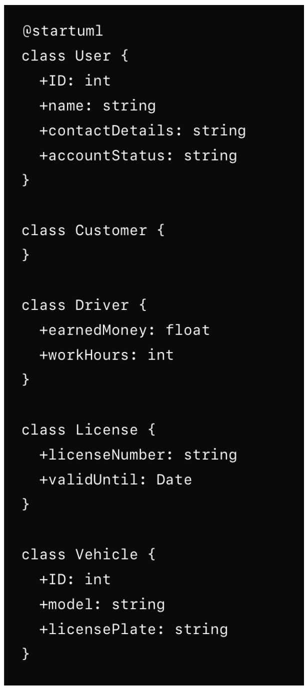
```
class License {
    number
    expiryDate
}

class Order {
    ID
    status
}

class Transaction {
    ID
    status
    paymentMethod
}

class PaymentMethod {
    type
}

class Subscription {
}

class Address {
    street
    city
    country
}
```

```
User "has" Address
Driver "has" License
Driver "owns" Vehicle
Order "belongs to" Customer
Order "is assigned to" Vehicle
Order "has" Transaction
Transaction "uses" PaymentMethod
Transaction "is part of" Subscription
```

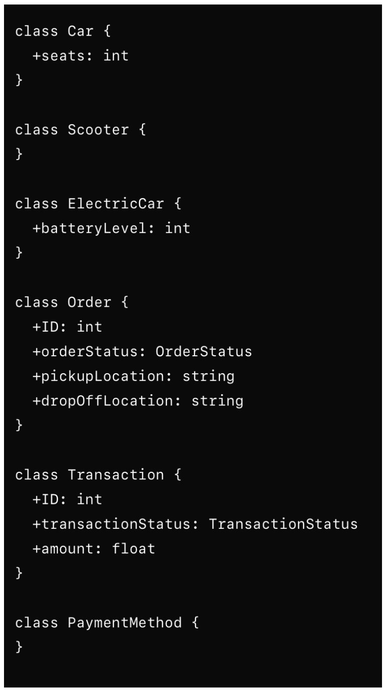
And this is the PlantUMLl generated by ChatGPT.



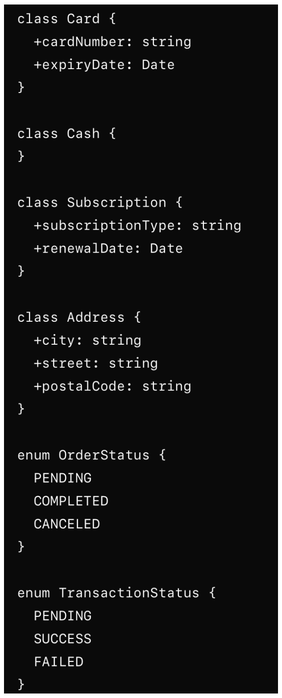
Below is the code:



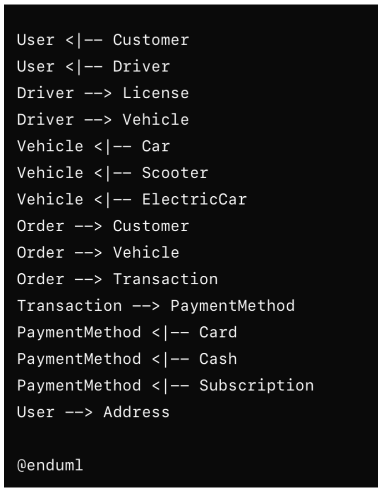
```
@startuml
class User {
  +ID: int
  +name: string
  +contactDetails: string
  +accountStatus: string
}

class Customer {
}

class Driver {
  +earnedMoney: float
  +workHours: int
}

class License {
  +licenseNumber: string
  +validUntil: Date
}

class Vehicle {
  +ID: int
  +model: string
  +licensePlate: string
}
```



```
class Car {
  +seats: int
}

class Scooter {
}

class ElectricCar {
  +batteryLevel: int
}

class Order {
  +ID: int
  +orderStatus: OrderStatus
  +pickupLocation: string
  +dropOffLocation: string
}

class Transaction {
  +ID: int
  +transactionStatus: TransactionStatus
  +amount: float
}

class PaymentMethod {
}
```



```
class Card {
  +cardNumber: string
  +expiryDate: Date
}

class Cash {
}

class Subscription {
  +subscriptionType: string
  +renewalDate: Date
}

class Address {
  +city: string
  +street: string
  +postalCode: string
}

enum OrderStatus {
  PENDING
  COMPLETED
  CANCELED
}

enum TransactionStatus {
  PENDING
  SUCCESS
  FAILED
}
```



```
User <|-- Customer
User <|-- Driver
Driver --> License
Driver --> Vehicle
Vehicle <|-- Car
Vehicle <|-- Scooter
Vehicle <|-- ElectricCar
Order --> Customer
Order --> Vehicle
Order --> Transaction
Transaction --> PaymentMethod
PaymentMethod <|-- Card
PaymentMethod <|-- Cash
PaymentMethod <|-- Subscription
User --> Address

@enduml
```

Although the code generated by ChatGPT provides some general structure and relations, it is not nearly detailed enough. It could be said that by providing multiple “guiding prompts” to ChatGPT it could be possible to build a good model. We certainly found it useful to use ChatGPT to fasten up the work by guiding it, making it help us build our vision. However we believe that without the humans’ vision ChatGPT is not good enough to build models, it doesn’t really have a vision.