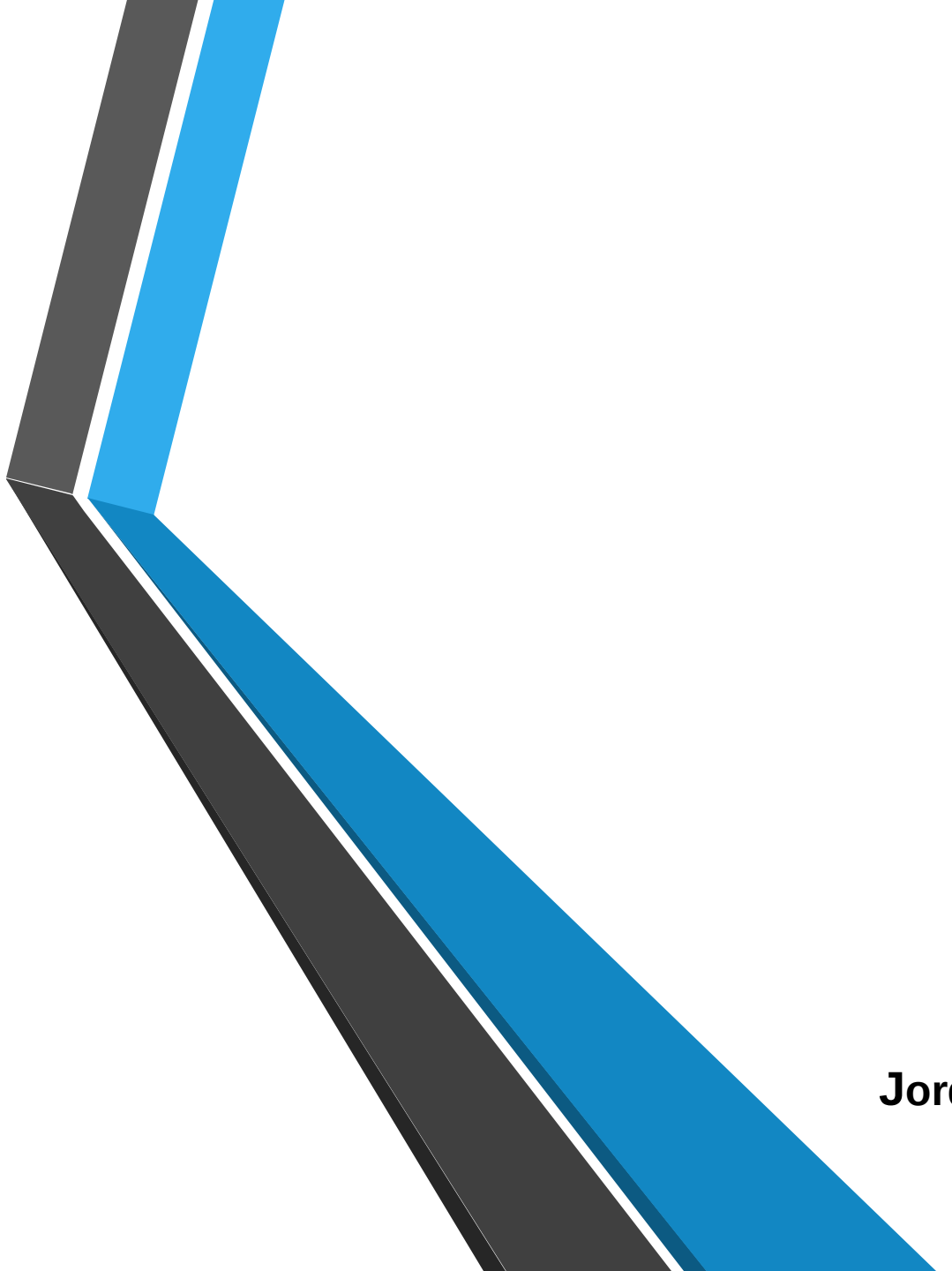


VALET BUDDY

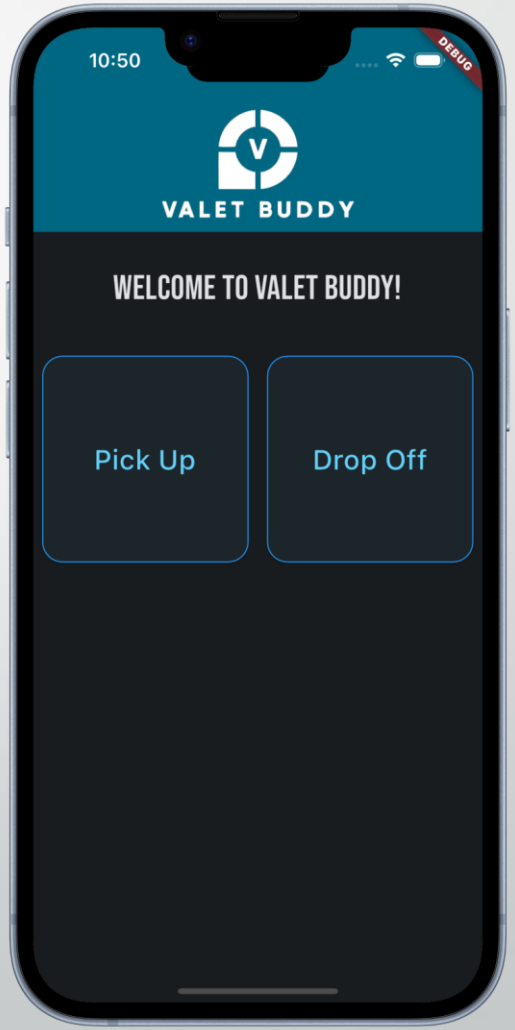
Airport Valet Car Locator Mobile App

Jordan Treutel, Richard Hoehn, Ian Hurd, Patrick Burnett

November 6th, 2023

Introduction

- Mobile App for Valet Parking at Airports
- App is solely used by the Valets
- Business Case of App is Simple
  - GPS Location Tagging instead of parking field numbers
  - Scalable Parking Lots
  - No cost for painting numbers
- Client / Server Architecture
  - Server – Python Flask Server with TinyDB as the datastore
  - Client – Flutter - iOS & Android Cross Platform Development



Flutter python TinyDB

Requirements Gathering – Stakeholders/Users

- Airport administration
- Valets (drivers) - main users of the app
- Airport travelers (car owners)
- Owner of airport parking lot
- Business and financial staff at Valet Buddy company

Requirements Gathering
Functional Requirements

- Users can input the car's license plate number
- Users can upload a picture of their car to the app
- User can save GPS coordinates of a car's location
- Users can access a list of cars and select one to retrieve its coordinates
- Users can only see cars from their own airport, unless they are an airport administrator
- Valet drivers can report problems with cars, such as cars being damaged or stolen
- Each car is given a "ticket" which tracks all data about its drop-off and pick-up
- There exists a list of tickets, each entry storing a picture of the car, the car's license plate number, the name of the car's owner, and the coordinates of the car's location

Requirements Gathering – Non Functional Requirement

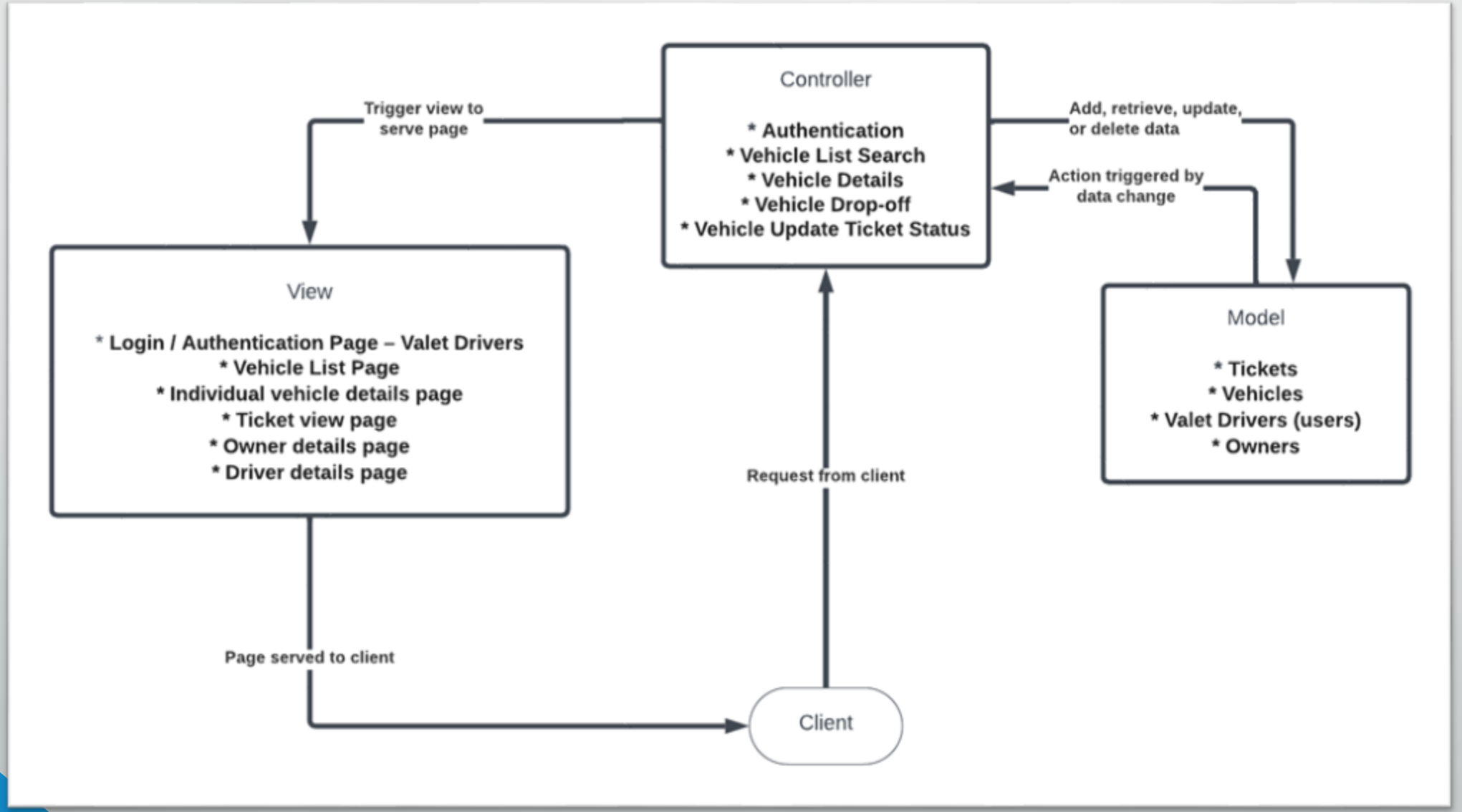
In Scope

- JSON for data transfer
- Flutter - mobile framework
- Dart – mobile programming language
- Google Maps API
- Python
- Flask (REST)
- Database – TinyDB (native python)

Out of Scope

- All communications done via 'https'
- Backing up database and car images to AWS S3 on a daily basis
- For demo, will store files on server – Usually this would be in AWS S3
- Monetization
- Email Weekly map (PDF) of all location of the cars

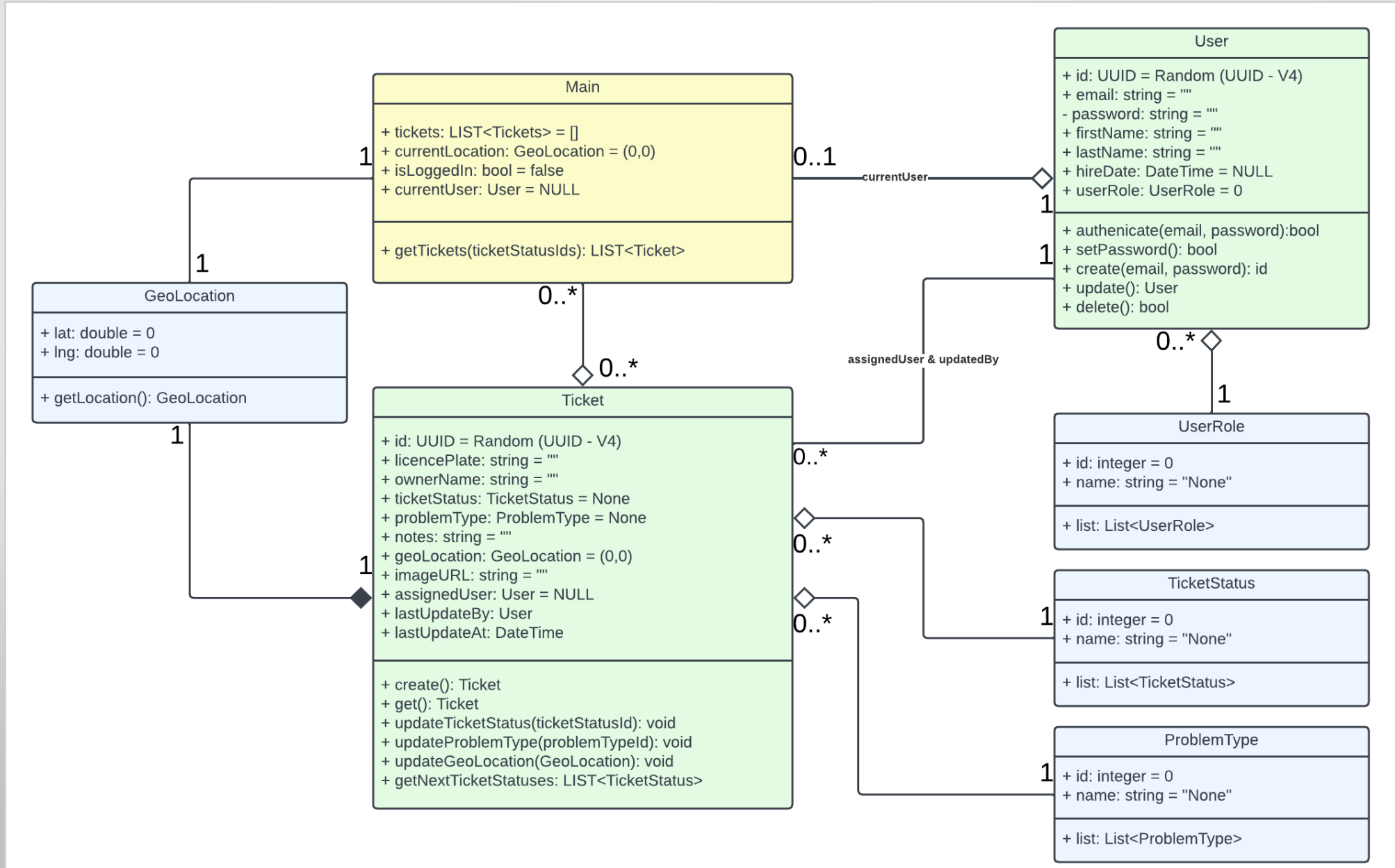
Design Architecture - MVC



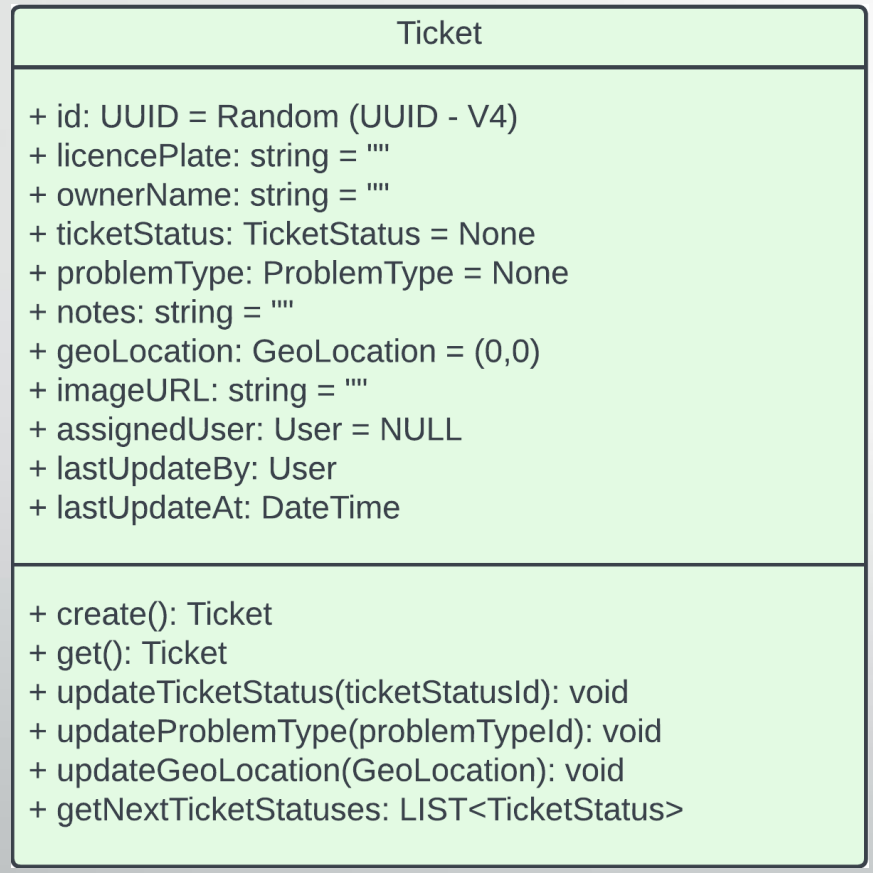
Key Object-Oriented Principles

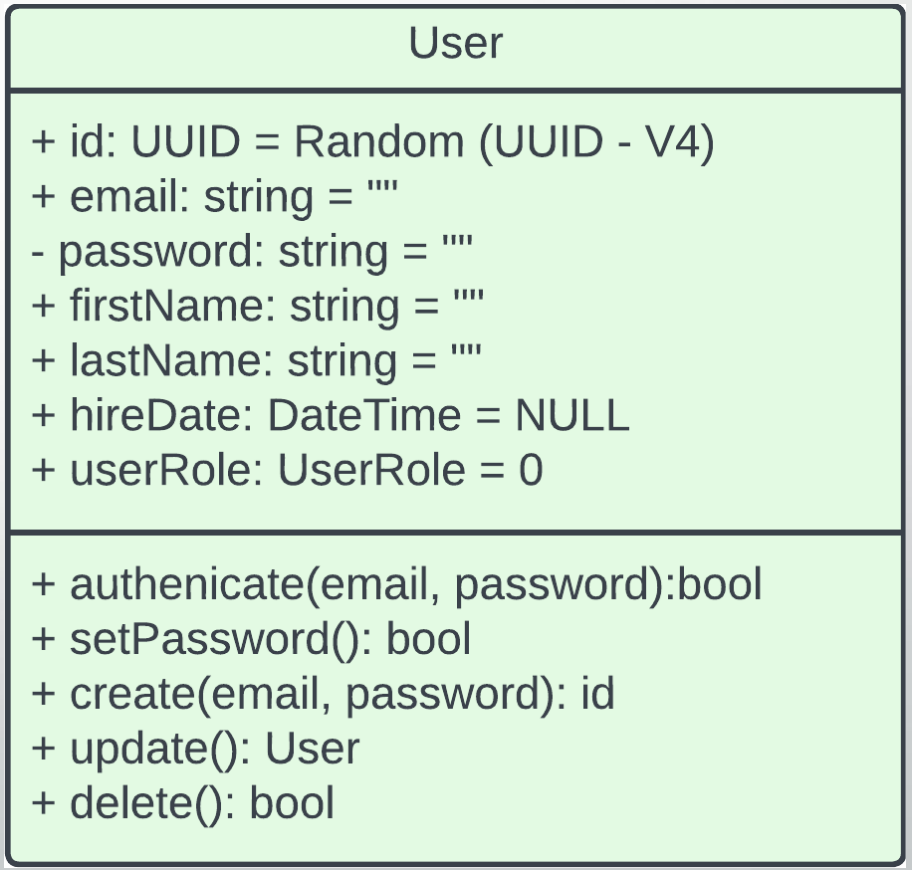
- Inheritance (Flutter)
  - Widget inheritance is one of the design patterns in Flutter.
  - Most all UI frameworks (flutter, angular, react) use inheritance to manage the UI state.
- Composition
  - In Flutter we regularly use composition, in that a widget (Expl: Container) is passed a "child" widget of type TextBox() - This means that the Container is composable.
- Encapsulation
  - Our Valet Buddy app is heavily reliant on encapsulation. This can be observed by our class diagram implementing methods that only that class can perform.
  - This is important since it places restrictions on accessing methods directly and prevents accidental modification of data and states of the Tickets.

Class Diagram

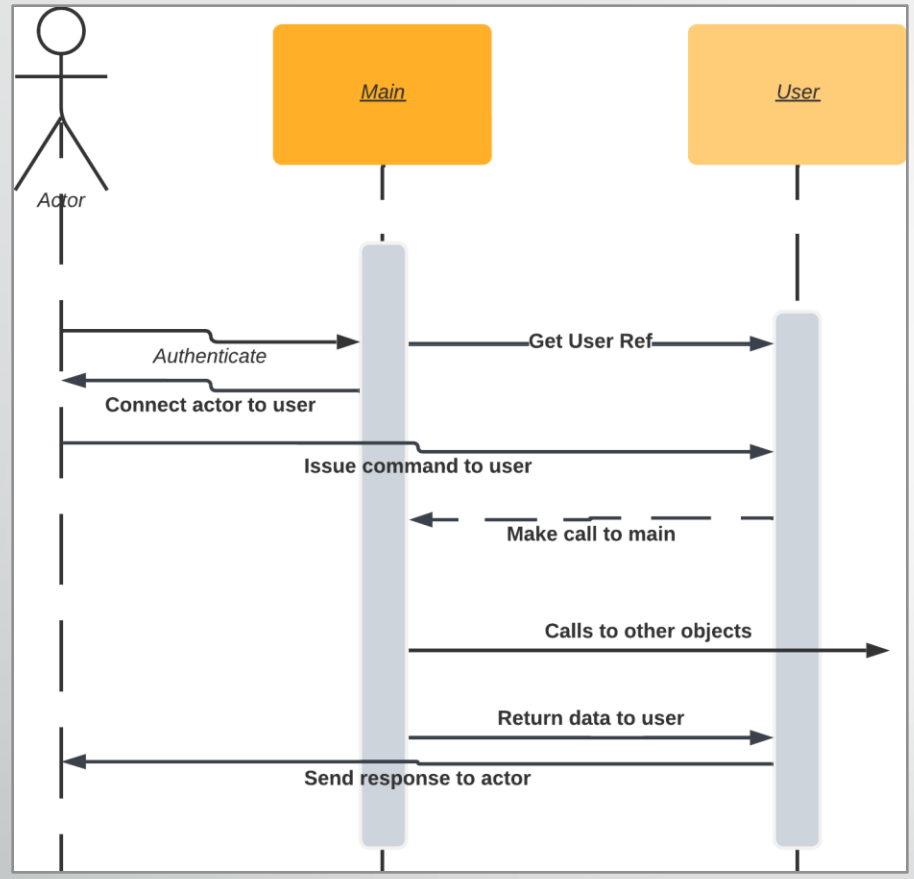


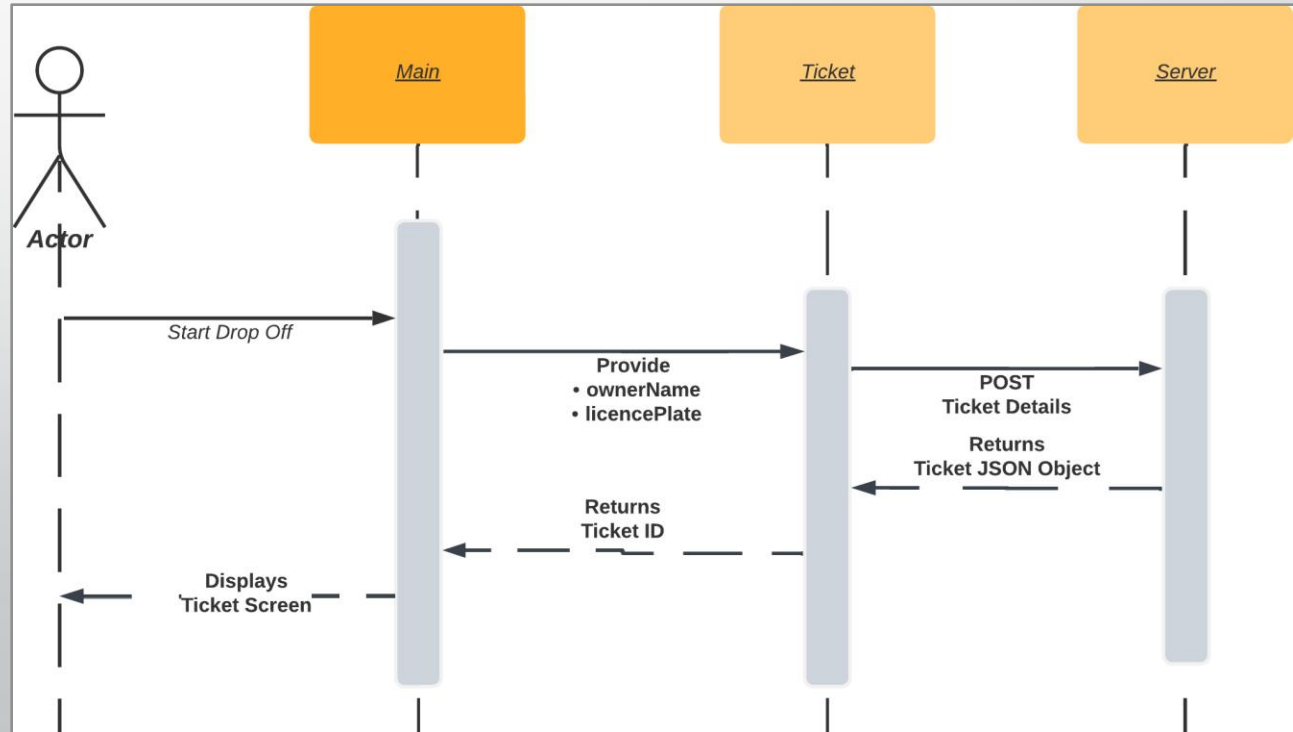
Class Diagram – Ticket & User





Sequence Diagram – Auth & Ticket





Unit Testing

Valet Buddy Unit Test Strategy

- Test Early & Often
- Prioritize the Critical Path
- Implement Using Flutter's Test Package Runners

Main

Seq	Method	Parameter	Response	
			Object Type	*Details*
1	Main<<Constructor>> Singelton - Approach We only have one isntance of Main for the Application	*NONE*	Main	main.tickets = LIST<<EMPTY>> main.currentLocation = <<GeoLocation(0,0)>> main.isLoggedIn = FALSE main.currentUser = NULL
2	main getTickets()	*NONE*	<<LIST>> Ticket	ticket.id = 1 ticket.ownerName = "John Doe" ticket.licencePlate = "123 ABC" ticket.lastUpdateAt = <<Current Date & Time>> ticket.titcketStatus = <<TicketStatus.Parking>> ticket.problemType= <<ProblemType.None>> ticket.geoLocation = <<GeoLocation(0,0)>> * * *

Ticket

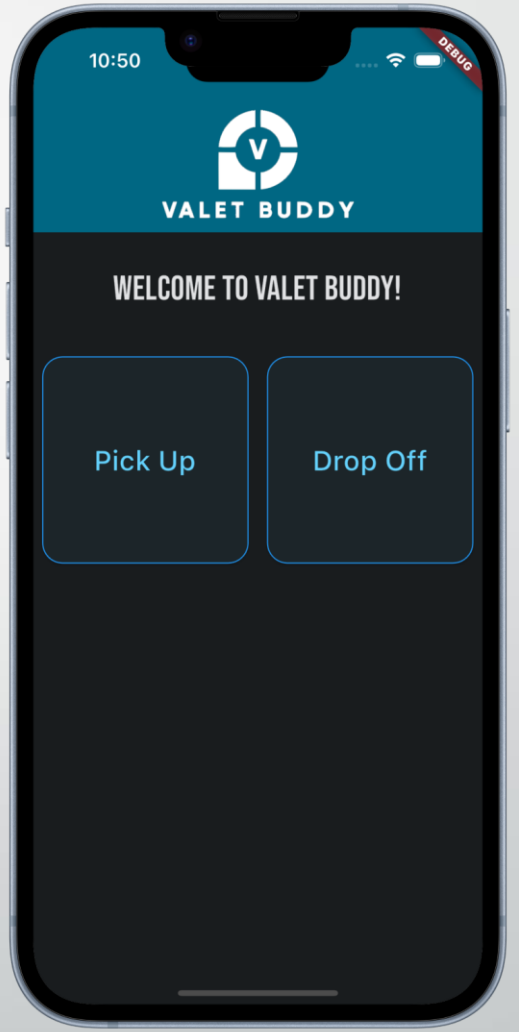
Seq	Method	Parameter	Response	
			Object Type	*Details*
1	Ticket<<Constructor>> create()	"ownerName" = "John Doe" "licencePlate" = "123 ABC"	Ticket	ticket.id = 1 (dynamic) ticket.ownerName = "John Doe" ticket.licencePlate = "123 ABC" ticket.lastUpdateAt = <<Current Date & Time>> ticket.titcketStatus = <<TicketStatus.None>> ticket.problemType= <<ProblemType.None>> ticket.geoLocation = <<GeoLocation(0,0)>>
2	ticket. get()	*NONE*	Ticket	ticket.id = 1 ticket.ownerName = "John Doe" ticket.licencePlate = "123 ABC" ticket.lastUpdateAt = <<Current Date & Time>> ticket.titcketStatus = <<TicketStatus.None>> ticket.problemType= <<ProblemType.None>> ticket.geoLocation = <<GeoLocation(0,0)>>
3	ticket. updateTicketStatus()	"ticketStatusId" = 2	Ticket	ticket.id = 1 ticket.ownerName = "John Doe" ticket.licencePlate = "123 ABC" ticket.lastUpdateAt = <<Current Date & Time>> ticket.titcketStatus = <<TicketStatus.Parking>> ticket.problemType= <<ProblemType.None>> ticket.geoLocation = <<GeoLocation(0,0)>>
4	ticket. updateProblemType()	"problemTypeId" = 2	Ticket	ticket.id = 1 ticket.ownerName = "John Doe" ticket.licencePlate = "123 ABC" ticket.lastUpdateAt = <<Current Date & Time>> ticket.titcketStatus = <<TicketStatus.Parking>> ticket.problemType= <<ProblemType.Lost>> ticket.geoLocation = <<GeoLocation(0,0)>>
5	ticket. updateGeoLocation()	"geoLocation" = (12.345, 67.890)	Ticket	ticket.id = 1 ticket.ownerName = "John Doe" ticket.licencePlate = "123 ABC" ticket.lastUpdateAt = <<Current Date & Time>> ticket.titcketStatus = <<TicketStatus.Parking>> ticket.problemType= <<ProblemType.Lost>> ticket.geoLocation = <<GeoLocation(12.345, 67.890)>>
6	ticket. getNextTicketStatuses()	*NONE*	<<LIST>> TicketStatus	ticketStatus.id = 1 ticketStatus.name = "None" * * *

OO Principles & Modular Code

- Code Reusability
- Maintainability (changing one part of code doesn't mess with unrelated functionality)
  - Config file for changing global values across the app (IP, port, host name, color themes, and other constants)
- Readability (organized and easy to compartmentalize in your head)

Summary

- Valet Buddy is a mobile app that helps valet drivers record parking of vehicles and eliminates need to maintain physical numbering of parking spaces by GPS location tagging.
- Client / Server Architecture
  - Server – Python Flask Server with TinyDB as the datastore
  - Client – Flutter - iOS & Android Cross Platform Development
- Object-Oriented design (Python & Flutter)
- Heavy use of Inheritance (essence of cross platform development)
- Unit Testing via Flutter Test Package Runners



Flutter python TinyDB

Key Takeaways

- Adding or modifying any project or feature can grow in complexity very quickly
  - Expl: Ticket Status + Problem Types etc…
- Design Architecture of choice = foundation for development
- Class Diagrams = transition from design to implementation
  - Lots of time spent on defining the methods
- Challenges:
  - Fitting Design Architecture to MVC
    - Do model and view directly communicate?
    - Valet drivers send requests directly to controller, not the view
  - Deciding relationships between classes
  - Mobile App vs. Server might require two different Architectures

Questions?