VALET BUDDY

Airport Valet Car Locator Mobile App

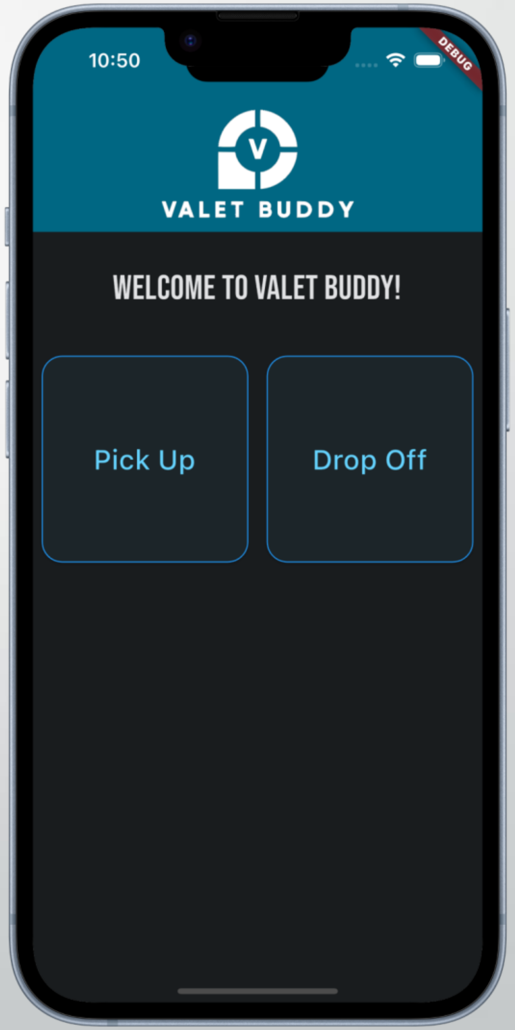
Jordan Treutel, Richard Hoehn, Ian Hurd, Patrick Burnett

December 6th, 2023

RH

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



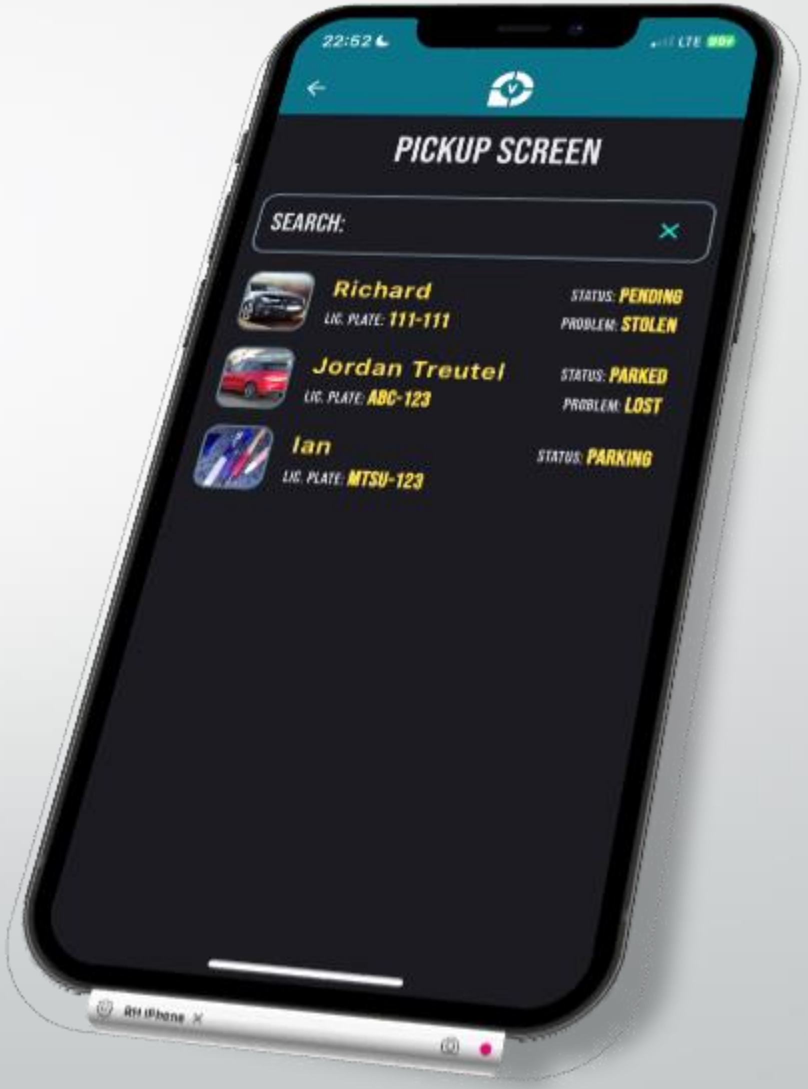
WELCOME TO VALET BUDDY!

Flutter python TinyDB

RH

Problem Statement

- Using this app and GPS technology, the Airport Car parking owner will no longer have to physically paint numbers on parking spaces anymore!
- Which Saves:
  - Time
  - Money
  - and Improves Scalability of expanding to more Parking Lots
- It allows to faster find the cars and move them around.
- Plus enabling taking pictures allows for quicker retrieval



22:52

PICKUP SCREEN

SEARCH: ✕

Richard
LIC. PLATE: 111-111
STATUS: PENDING
PROBLEM: STOLEN

Jordan Treutel
LIC. PLATE: ABC-123
STATUS: PARKED
PROBLEM: LOST

Ian
LIC. PLATE: MTSU-123
STATUS: PARKING

RH

Project Scope & Stakeholders/Users

App

- Input the car's license plate number
- Upload a picture of their car to the app
- Save GPS coordinates of a car's location
- Access / Filter list of cars and select one to retrieve
- Report problems with cars (Lost, Stolen, Scratch)

Server

- RESTful API
- JSON
- Flask with TinyDB as database

- 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

RH

Project Scope (cont.)
(Non Functional Requirements)

In Scope

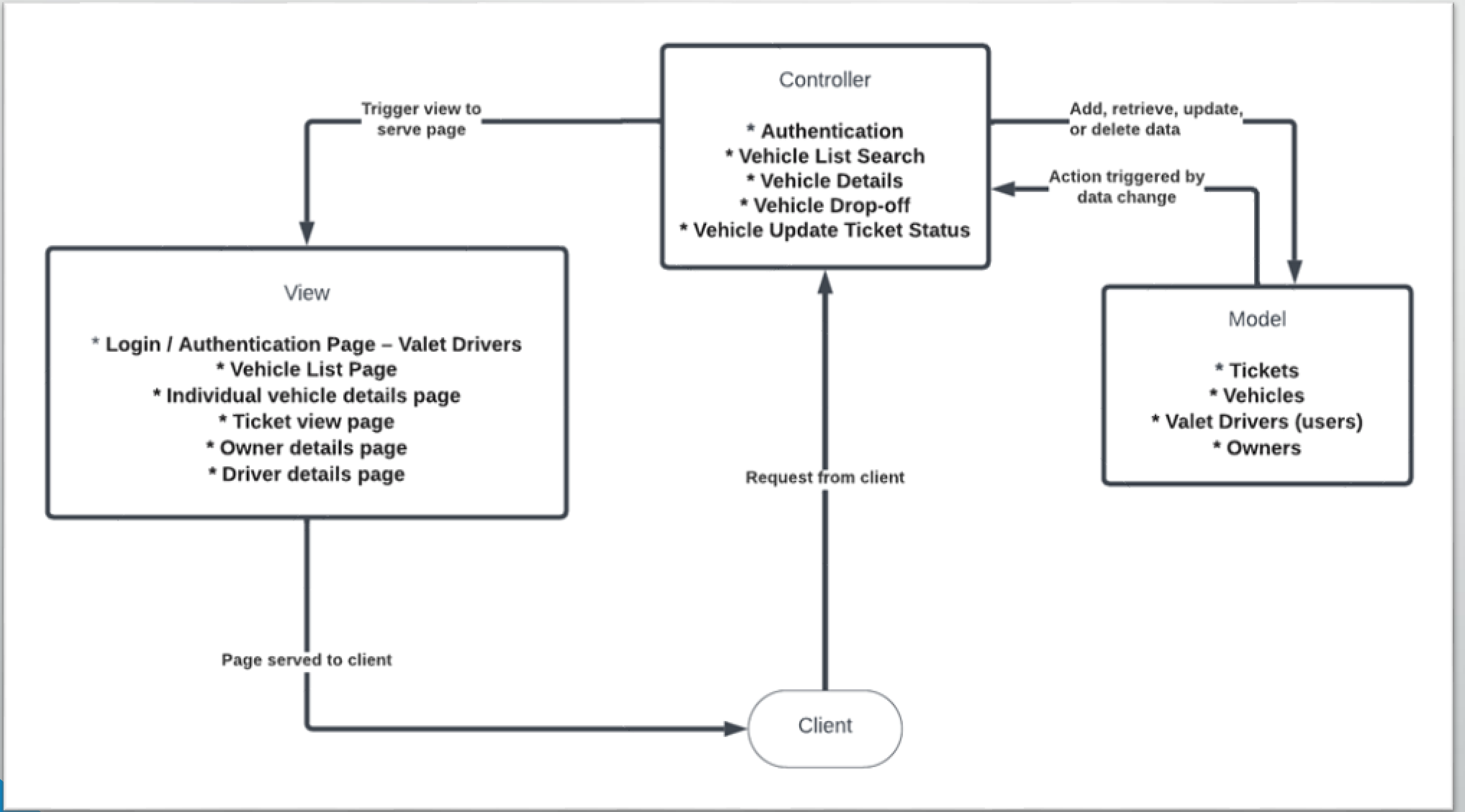
- JSON for data transfer
- Flutter - mobile framework
- Dart – mobile programming language
- Google Maps API
- Python
- Flask (REST)
- Database – TinyDB (native python)
- Image Capture in JPEG

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
- User Registration & Mgmt.

PB

Initial Design Architecture - MVC



Controller

* Authentication
* Vehicle List Search
* Vehicle Details
* Vehicle Drop-off
* Vehicle Update Ticket Status

View

* Login / Authentication Page – Valet Drivers
* Vehicle List Page
* Individual vehicle details page
* Ticket view page
* Owner details page
* Driver details page

Model

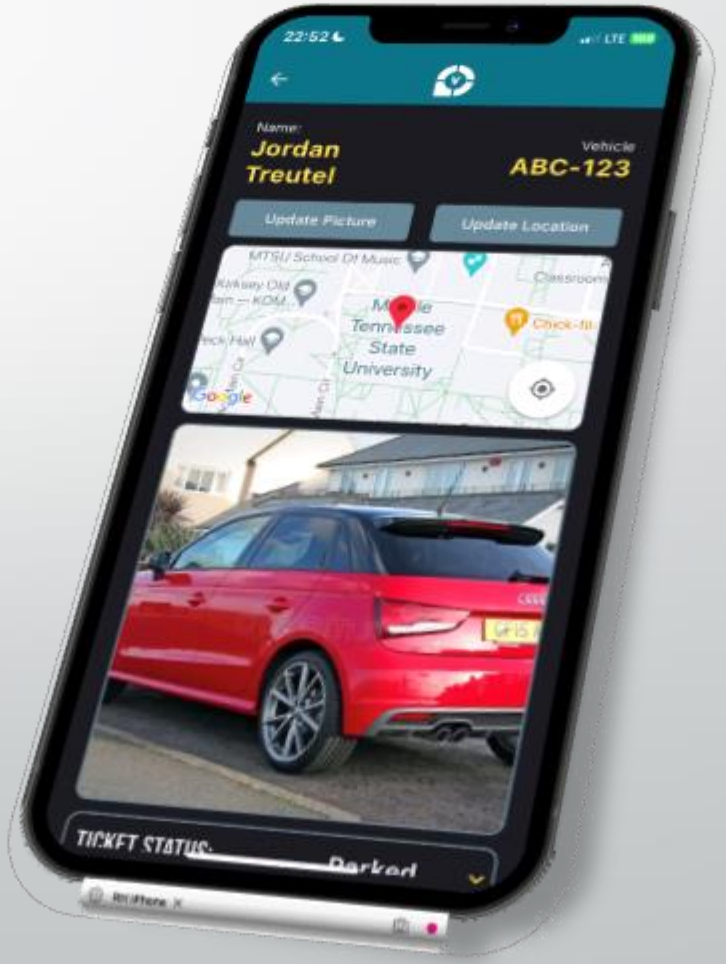
* Tickets
* Vehicles
* Valet Drivers (users)
* Owners

Trigger view to serve page
Add, retrieve, update, or delete data
Action triggered by data change
Request from client
Page served to client
Client

PB

Implementation: MVC and Flask

- Despite the initial plan being model-view-controller framework for the application, the actual implementation only made use of **<u>models</u>** and **<u>views</u>**, with views filling the role of what would otherwise be the controllers' job.
- Presentation is handled by client-side Flutter framework



Name: Jordan Treutel

Vehicle ABC-123

TICKET STATUS: Parked

JT

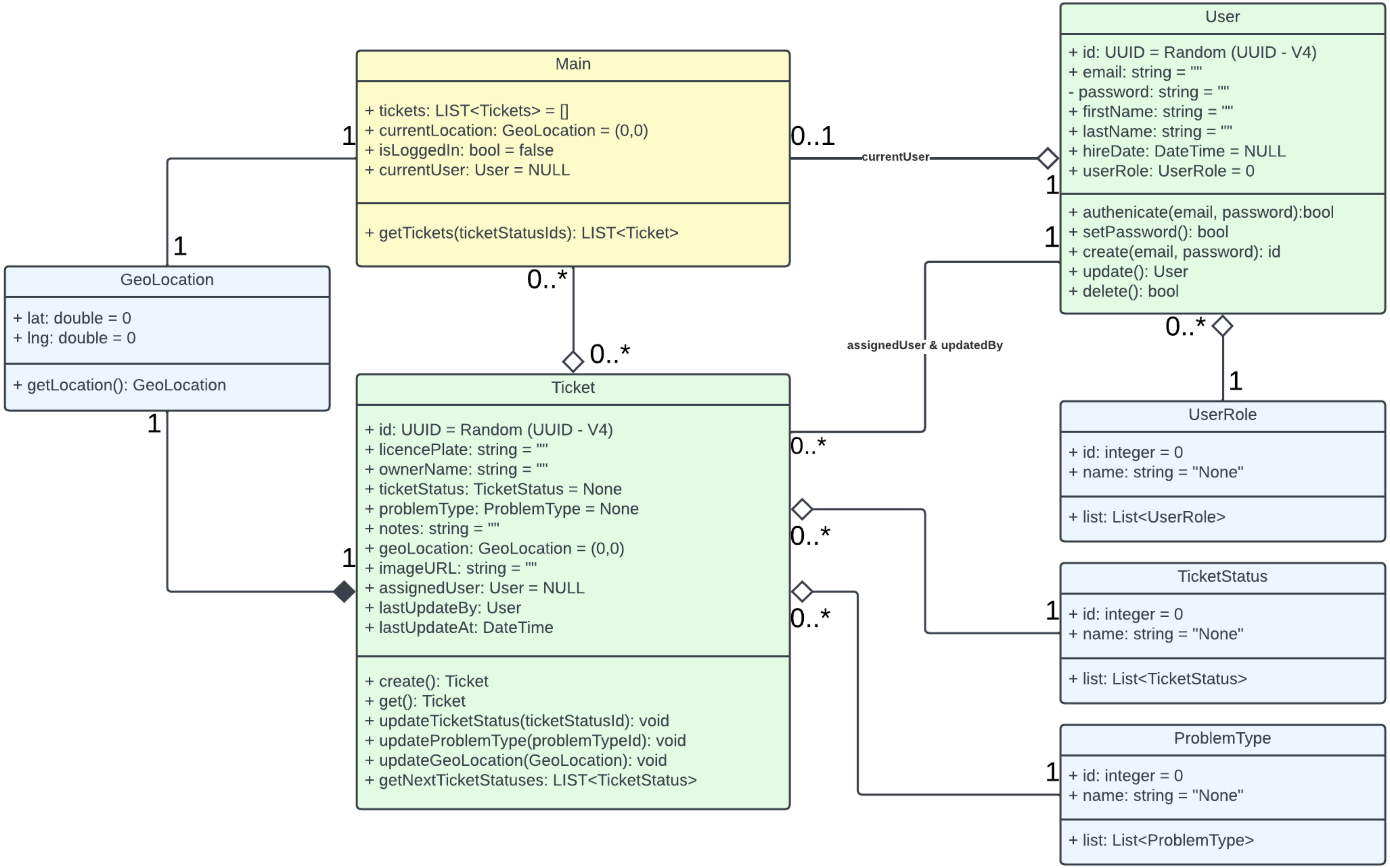
Key Object-Oriented Principles

- 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.

OO Principles & Modular Code

- Code Reusability
  - Ticket, TicketStatus, ProblemType classes
- 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)
  - Custom Widgets used to build screens

Class Diagram - App



Main
+ tickets: LIST<Tickets> = []
+ currentLocation: GeoLocation = (0,0)
+ isLoggedIn: bool = false
+ currentUser: User = NULL
+ getTickets(ticketStatusIds): LIST<Ticket>
User
+ id: UUID = Random (UUID - V4)
+ email: string = ""
- password: string = ""
+ firstName: string = ""
+ lastName: string = ""
+ hireDate: DateTime = NULL
+ userRole: UserRole = 0
+ authenicate(email, password):bool
+ setPassword(): bool
+ create(email, password): id
+ update(): User
+ delete(): bool
GeoLocation
+ lat: double = 0
+ lng: double = 0
+ getLocation(): GeoLocation
Ticket
+ id: UUID = Random (UUID - V4)
+ licencePlate: string = ""
+ ownerName: string = ""
+ ticketStatus: TicketStatus = None
+ problemType: ProblemType = None
+ notes: string = ""
+ geoLocation: GeoLocation = (0,0)
+ imageURL: string = ""
+ assignedUser: User = NULL
+ lastUpdateBy: User
+ lastUpdateAt: DateTime
+ create(): Ticket
+ get(): Ticket
+ updateTicketStatus(ticketStatusId): void
+ updateProblemType(problemTypeId): void
+ updateGeoLocation(GeoLocation): void
+ getNextTicketStatuses: LIST<TicketStatus>
UserRole
+ id: integer = 0
+ name: string = "None"
+ list: List<UserRole>
TicketStatus
+ id: integer = 0
+ name: string = "None"
+ list: List<TicketStatus>
ProblemType
+ id: integer = 0
+ name: string = "None"
+ list: List<ProblemType>
currentUser
assignedUser & updatedBy
1
0..1
1
1
1
0..*
0..*
1
1
0..*
1
0..*
0..*
0..*
1
1

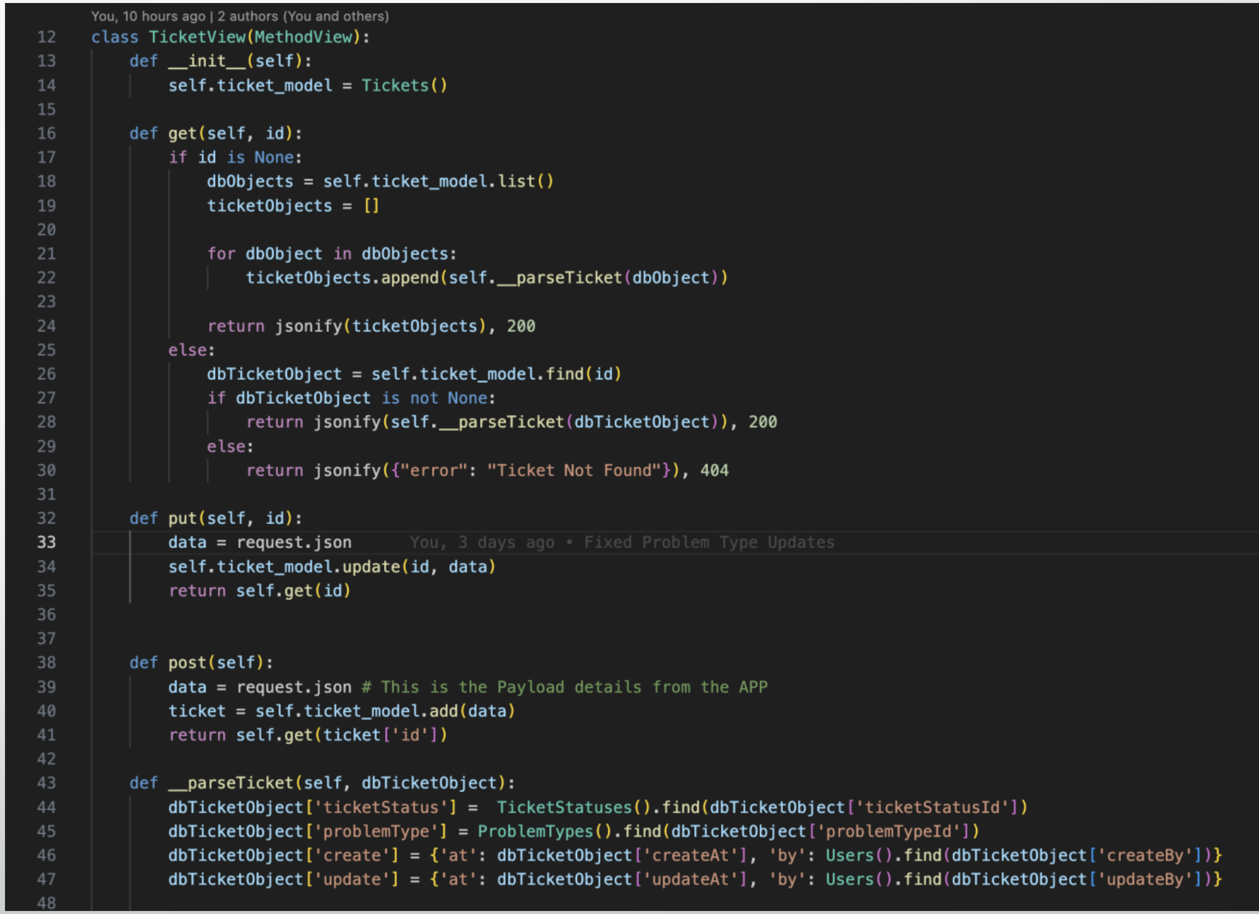
JT

Design 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

Server Implementation – Ticket View

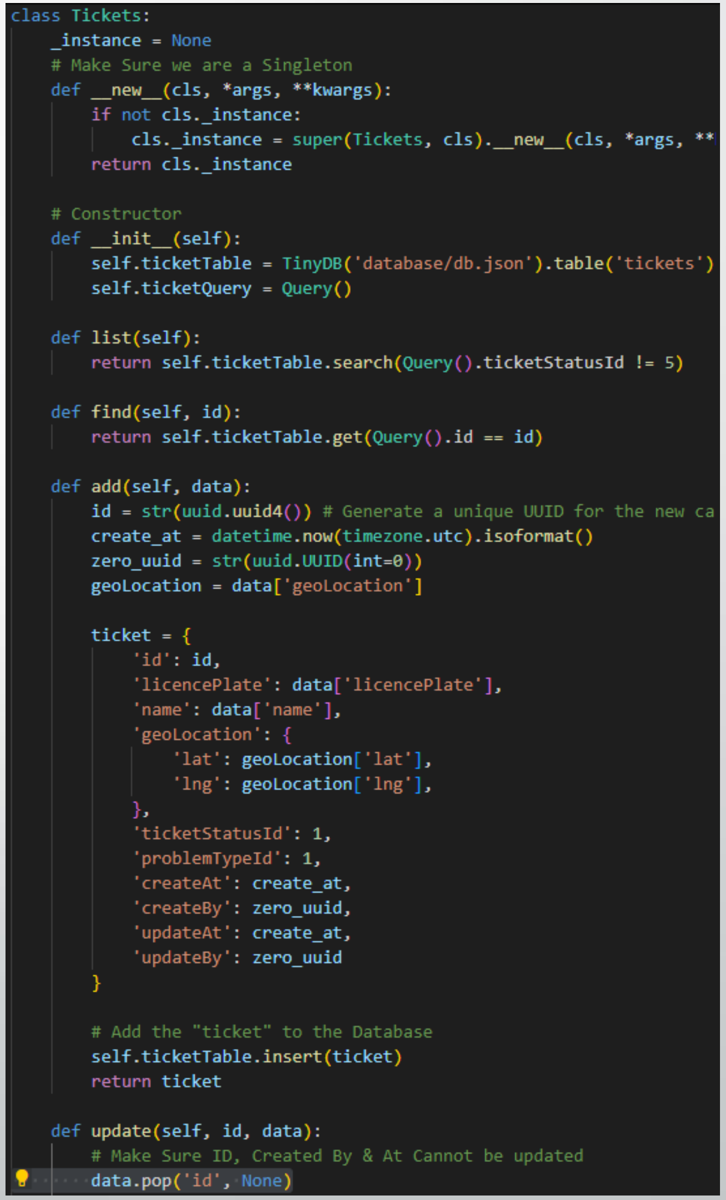
- Get request for ticket is passed ticket ID and retrieves ticket.
- Certain fields receive additional formatting before output
- Lack of ID passed to get returns list of all tickets
- POST and PUT create and update tickets respectively



```python
You, 10 hours ago | 2 authors (You and others)
class TicketView(MethodView):
    def __init__(self):
        self.ticket_model = Tickets()

    def get(self, id):
        if id is None:
            dbObjects = self.ticket_model.list()
            ticketObjects = []

            for dbObject in dbObjects:
                ticketObjects.append(self.__parseTicket(dbObject))

            return jsonify(ticketObjects), 200
        else:
            dbTicketObject = self.ticket_model.find(id)
            if dbTicketObject is not None:
                return jsonify(self.__parseTicket(dbTicketObject)), 200
            else:
                return jsonify({"error": "Ticket Not Found"}), 404

    def put(self, id):
        data = request.json      You, 3 days ago • Fixed Problem Type Updates
        self.ticket_model.update(id, data)
        return self.get(id)


    def post(self):
        data = request.json # This is the Payload details from the APP
        ticket = self.ticket_model.add(data)
        return self.get(ticket['id'])

    def __parseTicket(self, dbTicketObject):
        dbTicketObject['ticketStatus'] =  TicketStatuses().find(dbTicketObject['ticketStatusId'])
        dbTicketObject['problemType'] = ProblemTypes().find(dbTicketObject['problemTypeId'])
        dbTicketObject['create'] = {'at': dbTicketObject['createAt'], 'by': Users().find(dbTicketObject['createBy'])}
        dbTicketObject['update'] = {'at': dbTicketObject['updateAt'], 'by': Users().find(dbTicketObject['updateBy'])}

```

Server Implementation – Ticket Model

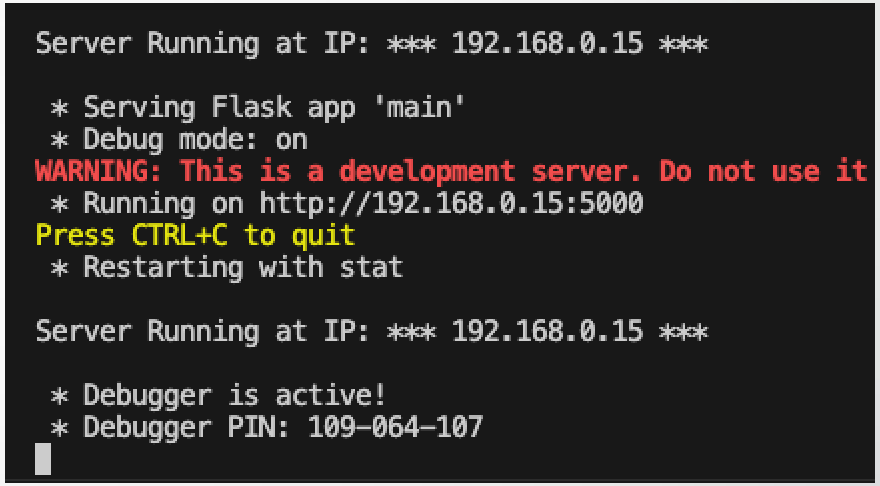
- Model handling direct database access for ticket retrieval and manipulation
- Contains methods for creating, modifying, searching, and listing tickets
- Important to know is that the Model – Is a singleton – since it interfaces with the Database.



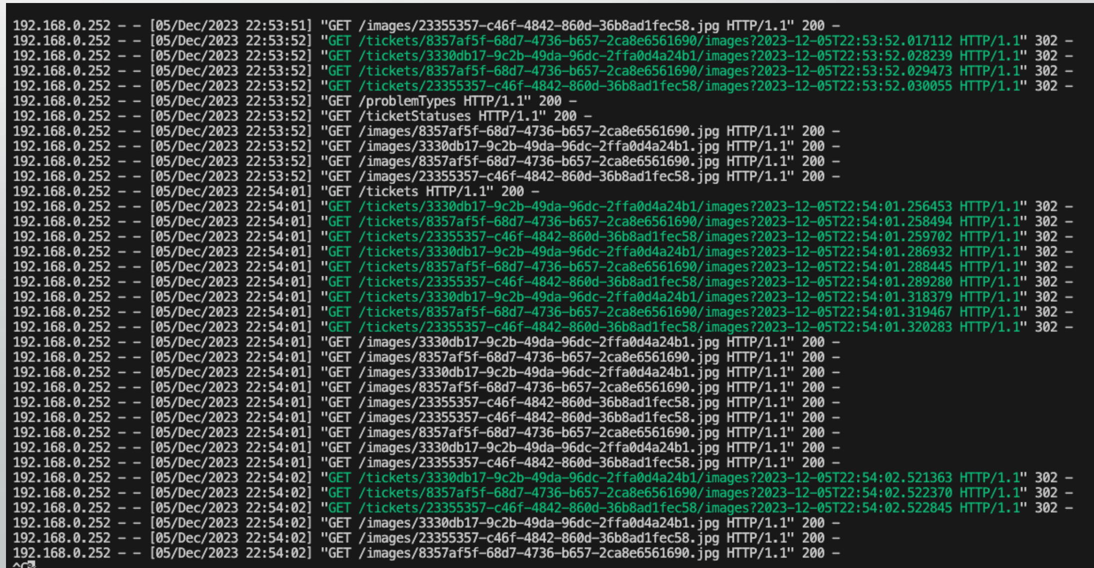
```
class Tickets:
    _instance = None
    # Make Sure we are a Singleton
    def __new__(cls, *args, **kwargs):
        if not cls._instance:
            cls._instance = super(Tickets, cls).__new__(cls, *args, **
        return cls._instance

    # Constructor
    def __init__(self):
        self.ticketTable = TinyDB('database/db.json').table('tickets')
        self.ticketQuery = Query()

    def list(self):
        return self.ticketTable.search(Query().ticketStatusId != 5)

    def find(self, id):
        return self.ticketTable.get(Query().id == id)

    def add(self, data):
        id = str(uuid.uuid4()) # Generate a unique UUID for the new ca
        create_at = datetime.now(timezone.utc).isoformat()
        zero_uuid = str(uuid.UUID(int=0))
        geoLocation = data['geoLocation']

        ticket = {
            'id': id,
            'licencePlate': data['licencePlate'],
            'name': data['name'],
            'geoLocation': {
                'lat': geoLocation['lat'],
                'lng': geoLocation['lng'],
            },
            'ticketStatusId': 1,
            'problemTypeId': 1,
            'createAt': create_at,
            'createBy': zero_uuid,
            'updateAt': create_at,
            'updateBy': zero_uuid
        }

        # Add the "ticket" to the Database
        self.ticketTable.insert(ticket)
        return ticket

    def update(self, id, data):
        # Make Sure ID, Created By & At Cannot be updated
        data.pop('id', None)
```

Server Implementation – User Model and View

- Similar to ticket model and view
- GET finds specific instance or lists them all, as with tickets
- Lacks PUT method
- User authenticated with POST
- Users hardcoded for development; software interface for user registration out of scope for project



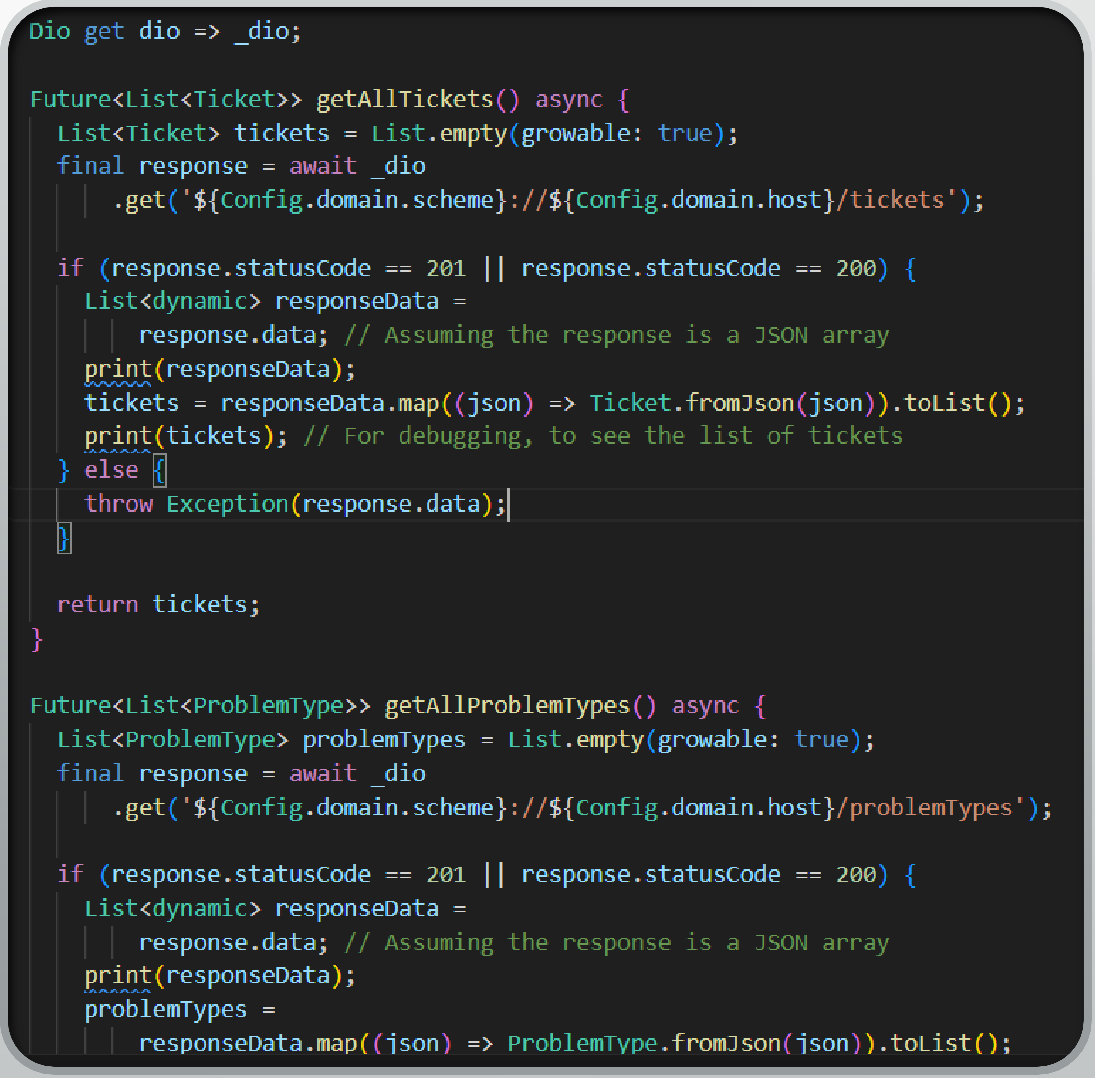
```
Server Running at IP: *** 192.168.0.15 ***

 * Serving Flask app 'main'
 * Debug mode: on
WARNING: This is a development server. Do not use it
 * Running on http://192.168.0.15:5000
Press CTRL+C to quit
 * Restarting with stat

Server Running at IP: *** 192.168.0.15 ***

 * Debugger is active!
 * Debugger PIN: 109-064-107
```



```
192.168.0.252 - - [05/Dec/2023 22:53:51] "GET /images/23355357-c46f-4842-860d-36b8ad1fec58.jpg HTTP/1.1" 200 -
192.168.0.252 - - [05/Dec/2023 22:53:52] "GET /tickets/8357af5f-68d7-4736-b657-2ca8e6561690/images?2023-12-05T22:53:52.017112 HTTP/1.1" 302 -
192.168.0.252 - - [05/Dec/2023 22:53:52] "GET /tickets/3330db17-9c2b-49da-96dc-2ffa0d4a24b1/images?2023-12-05T22:53:52.028239 HTTP/1.1" 302 -
192.168.0.252 - - [05/Dec/2023 22:53:52] "GET /tickets/8357af5f-68d7-4736-b657-2ca8e6561690/images?2023-12-05T22:53:52.029473 HTTP/1.1" 302 -
192.168.0.252 - - [05/Dec/2023 22:53:52] "GET /tickets/23355357-c46f-4842-860d-36b8ad1fec58/images?2023-12-05T22:53:52.030055 HTTP/1.1" 302 -
192.168.0.252 - - [05/Dec/2023 22:53:52] "GET /problemTypes HTTP/1.1" 200 -
192.168.0.252 - - [05/Dec/2023 22:53:52] "GET /ticketStatuses HTTP/1.1" 200 -
192.168.0.252 - - [05/Dec/2023 22:53:52] "GET /images/8357af5f-68d7-4736-b657-2ca8e6561690.jpg HTTP/1.1" 200 -
192.168.0.252 - - [05/Dec/2023 22:53:52] "GET /images/3330db17-9c2b-49da-96dc-2ffa0d4a24b1.jpg HTTP/1.1" 200 -
192.168.0.252 - - [05/Dec/2023 22:53:52] "GET /images/8357af5f-68d7-4736-b657-2ca8e6561690.jpg HTTP/1.1" 200 -
192.168.0.252 - - [05/Dec/2023 22:53:52] "GET /images/23355357-c46f-4842-860d-36b8ad1fec58.jpg HTTP/1.1" 200 -
192.168.0.252 - - [05/Dec/2023 22:54:01] "GET /tickets HTTP/1.1" 200 -
192.168.0.252 - - [05/Dec/2023 22:54:01] "GET /tickets/3330db17-9c2b-49da-96dc-2ffa0d4a24b1/images?2023-12-05T22:54:01.256453 HTTP/1.1" 302 -
192.168.0.252 - - [05/Dec/2023 22:54:01] "GET /tickets/8357af5f-68d7-4736-b657-2ca8e6561690/images?2023-12-05T22:54:01.258494 HTTP/1.1" 302 -
192.168.0.252 - - [05/Dec/2023 22:54:01] "GET /tickets/23355357-c46f-4842-860d-36b8ad1fec58/images?2023-12-05T22:54:01.259702 HTTP/1.1" 302 -
192.168.0.252 - - [05/Dec/2023 22:54:01] "GET /tickets/3330db17-9c2b-49da-96dc-2ffa0d4a24b1/images?2023-12-05T22:54:01.286932 HTTP/1.1" 302 -
192.168.0.252 - - [05/Dec/2023 22:54:01] "GET /tickets/8357af5f-68d7-4736-b657-2ca8e6561690/images?2023-12-05T22:54:01.288445 HTTP/1.1" 302 -
192.168.0.252 - - [05/Dec/2023 22:54:01] "GET /tickets/23355357-c46f-4842-860d-36b8ad1fec58/images?2023-12-05T22:54:01.289280 HTTP/1.1" 302 -
192.168.0.252 - - [05/Dec/2023 22:54:01] "GET /tickets/3330db17-9c2b-49da-96dc-2ffa0d4a24b1/images?2023-12-05T22:54:01.318379 HTTP/1.1" 302 -
192.168.0.252 - - [05/Dec/2023 22:54:01] "GET /tickets/8357af5f-68d7-4736-b657-2ca8e6561690/images?2023-12-05T22:54:01.319467 HTTP/1.1" 302 -
192.168.0.252 - - [05/Dec/2023 22:54:01] "GET /tickets/23355357-c46f-4842-860d-36b8ad1fec58/images?2023-12-05T22:54:01.320283 HTTP/1.1" 302 -
192.168.0.252 - - [05/Dec/2023 22:54:01] "GET /images/3330db17-9c2b-49da-96dc-2ffa0d4a24b1.jpg HTTP/1.1" 200 -
192.168.0.252 - - [05/Dec/2023 22:54:01] "GET /images/8357af5f-68d7-4736-b657-2ca8e6561690.jpg HTTP/1.1" 200 -
192.168.0.252 - - [05/Dec/2023 22:54:01] "GET /images/3330db17-9c2b-49da-96dc-2ffa0d4a24b1.jpg HTTP/1.1" 200 -
192.168.0.252 - - [05/Dec/2023 22:54:01] "GET /images/8357af5f-68d7-4736-b657-2ca8e6561690.jpg HTTP/1.1" 200 -
192.168.0.252 - - [05/Dec/2023 22:54:01] "GET /images/23355357-c46f-4842-860d-36b8ad1fec58.jpg HTTP/1.1" 200 -
192.168.0.252 - - [05/Dec/2023 22:54:01] "GET /images/23355357-c46f-4842-860d-36b8ad1fec58.jpg HTTP/1.1" 200 -
192.168.0.252 - - [05/Dec/2023 22:54:01] "GET /images/8357af5f-68d7-4736-b657-2ca8e6561690.jpg HTTP/1.1" 200 -
192.168.0.252 - - [05/Dec/2023 22:54:01] "GET /images/3330db17-9c2b-49da-96dc-2ffa0d4a24b1.jpg HTTP/1.1" 200 -
192.168.0.252 - - [05/Dec/2023 22:54:01] "GET /images/23355357-c46f-4842-860d-36b8ad1fec58.jpg HTTP/1.1" 200 -
192.168.0.252 - - [05/Dec/2023 22:54:02] "GET /tickets/3330db17-9c2b-49da-96dc-2ffa0d4a24b1/images?2023-12-05T22:54:02.521363 HTTP/1.1" 302 -
192.168.0.252 - - [05/Dec/2023 22:54:02] "GET /tickets/8357af5f-68d7-4736-b657-2ca8e6561690/images?2023-12-05T22:54:02.522370 HTTP/1.1" 302 -
192.168.0.252 - - [05/Dec/2023 22:54:02] "GET /tickets/23355357-c46f-4842-860d-36b8ad1fec58/images?2023-12-05T22:54:02.522845 HTTP/1.1" 302 -
192.168.0.252 - - [05/Dec/2023 22:54:02] "GET /images/3330db17-9c2b-49da-96dc-2ffa0d4a24b1.jpg HTTP/1.1" 200 -
192.168.0.252 - - [05/Dec/2023 22:54:02] "GET /images/23355357-c46f-4842-860d-36b8ad1fec58.jpg HTTP/1.1" 200 -
192.168.0.252 - - [05/Dec/2023 22:54:02] "GET /images/8357af5f-68d7-4736-b657-2ca8e6561690.jpg HTTP/1.1" 200 -
^C
```

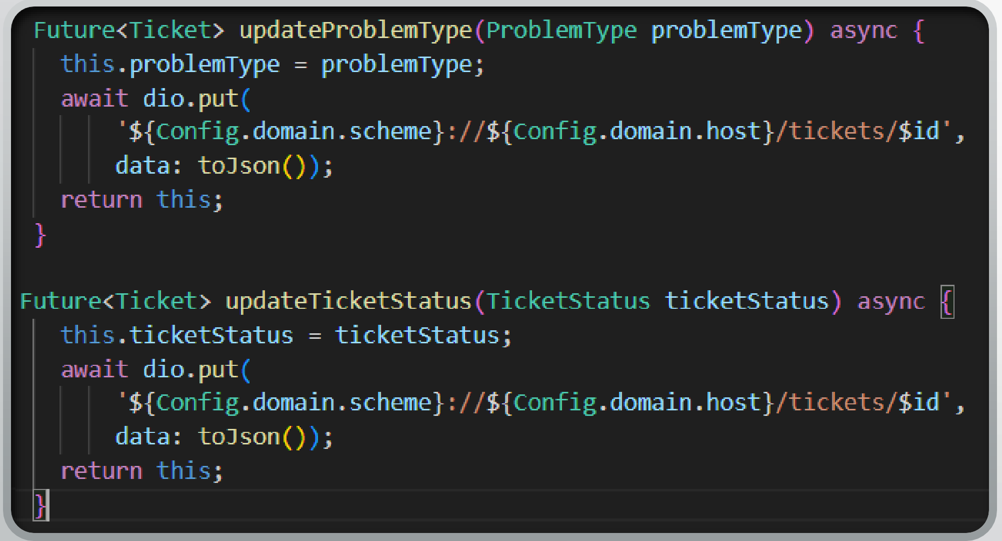
Client Implementation – App <-> Server Interface

- Establishes a single Dio connection to the server (HTTP networking package for Dart/Flutter)
- Each method defined allows for the client/app to interact with the server
- Example: getAllTickets() returns all of the tickets so they can be displayed on the Pick-Up Screen
- Retrieving data from the server:
  - Uses dio to pull from the /tickets endpoint
  - Waits for a response code of 201 or 200
  - Maps each ticket to a Ticket object
  - Returns a list of all Tickets
- Updating server done through the Ticket class itself (next slide)



```
Dio get dio => _dio;

Future<List<Ticket>> getAllTickets() async {
  List<Ticket> tickets = List.empty(growable: true);
  final response = await _dio
      .get('${Config.domain.scheme}://${Config.domain.host}/tickets');

  if (response.statusCode == 201 || response.statusCode == 200) {
    List<dynamic> responseData =
        response.data; // Assuming the response is a JSON array
    print(responseData);
    tickets = responseData.map((json) => Ticket.fromJson(json)).toList();
    print(tickets); // For debugging, to see the list of tickets
  } else {
    throw Exception(response.data);
  }

  return tickets;
}

Future<List<ProblemType>> getAllProblemTypes() async {
  List<ProblemType> problemTypes = List.empty(growable: true);
  final response = await _dio
      .get('${Config.domain.scheme}://${Config.domain.host}/problemTypes');

  if (response.statusCode == 201 || response.statusCode == 200) {
    List<dynamic> responseData =
        response.data; // Assuming the response is a JSON array
    print(responseData);
    problemTypes =
        responseData.map((json) => ProblemType.fromJson(json)).toList();
```

Client Implementation – Ticket Class

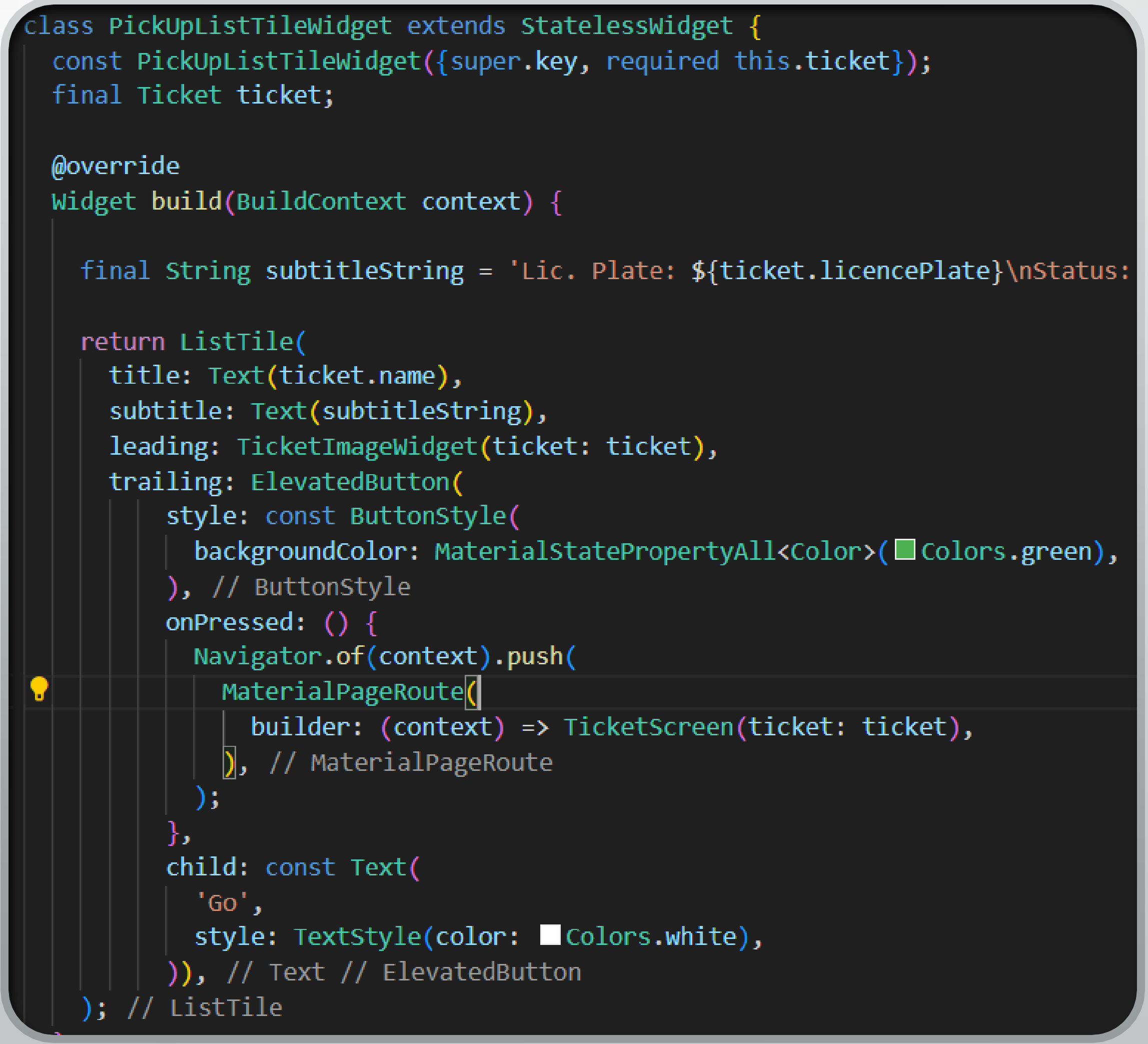


```
Future<Ticket> updateProblemType(ProblemType problemType) async {
  this.problemType = problemType;
  await dio.put(
      '${Config.domain.scheme}://${Config.domain.host}/tickets/$id',
      data: toJson());
  return this;
}

Future<Ticket> updateTicketStatus(TicketStatus ticketStatus) async {
  this.ticketStatus = ticketStatus;
  await dio.put(
      '${Config.domain.scheme}://${Config.domain.host}/tickets/$id',
      data: toJson());
  return this;
}
```

- Contains several attributes that comprise any Ticket:
  - Geolocation, id, licensePlate,
    name, problemType, ticketStatus, updatedBy
- Also contains methods that can update the server:
  - Directly update the server from inside the actual Ticket itself
  - Example: updateProblemType(problemType) will update the server to reflect the new problemType for *this* Ticket instance.
- Can also map a Ticket's Dart representation to or from JSON (so it can be stored/retrieved from the TinyDB on the server)

Client Implementation – Custom Widgets & Screens

- Stateful: A widget or screen that stores variable data that is used to build or rebuild the UI at any moment in time
- Stateless: A widget or screen that has no mutable state that needs to be updated; the UI does not dynamically update
- Stateful widgets/screens in our app often store a specific Ticket or a List<Ticket> relevant to that screen
    - Ex: Pick-Up Screen contains state that stores a list of all tickets for displaying.
- The screen itself is built using a hierarchical structure of widgets (both custom and built-in)

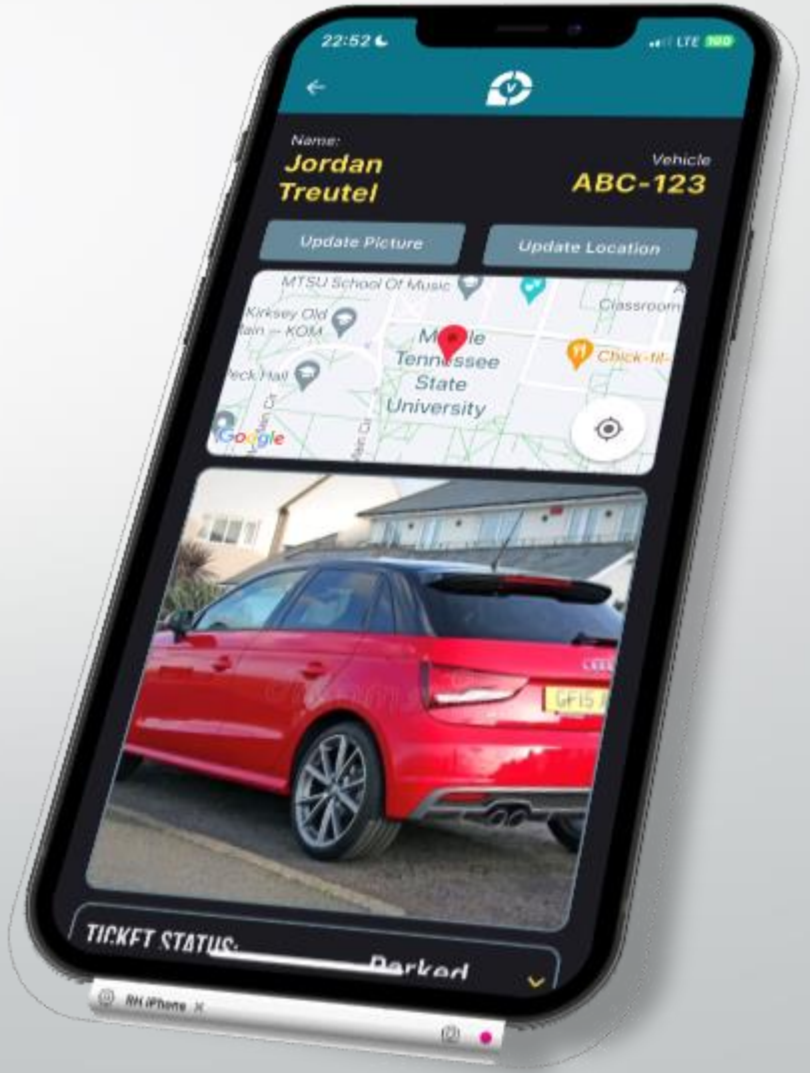


```
class PickUpListTileWidget extends StatelessWidget {
  const PickUpListTileWidget({super.key, required this.ticket});
  final Ticket ticket;

  @override
  Widget build(BuildContext context) {

    final String subtitleString = 'Lic. Plate: ${ticket.licencePlate}\nStatus:

    return ListTile(
      title: Text(ticket.name),
      subtitle: Text(subtitleString),
      leading: TicketImageWidget(ticket: ticket),
      trailing: ElevatedButton(
          style: const ButtonStyle(
            backgroundColor: MaterialStatePropertyAll<Color>(🟩Colors.green),
          ), // ButtonStyle
          onPressed: () {
            Navigator.of(context).push(
              MaterialPageRoute(
                builder: (context) => TicketScreen(ticket: ticket),
              ), // MaterialPageRoute
            );
          },
          child: const Text(
            'Go',
            style: TextStyle(color: ⬜Colors.white),
          )), // Text // ElevatedButton
    ); // ListTile
```

PB

Results & Achievements

- Successfully implemented planned design
- Working app that can connect to back end server
- Able to use Google Maps API



Name: Jordan Treutel

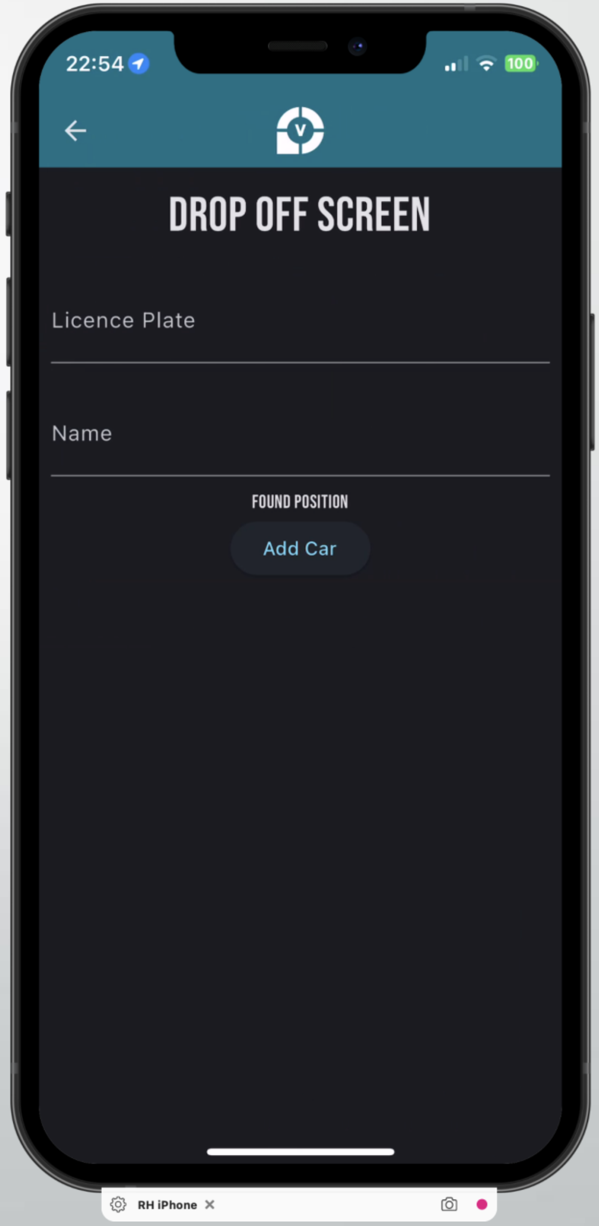
Vehicle ABC-123

TICKET STATUS: Parked

RH

Challenges

- Data type int/str issues during server/client interactions
- GPS location (by use of simulated or virtualized devices)
- Transitioning from MVC design plan to View/Model architecture
- Creating separate architectures for App and Server



PB

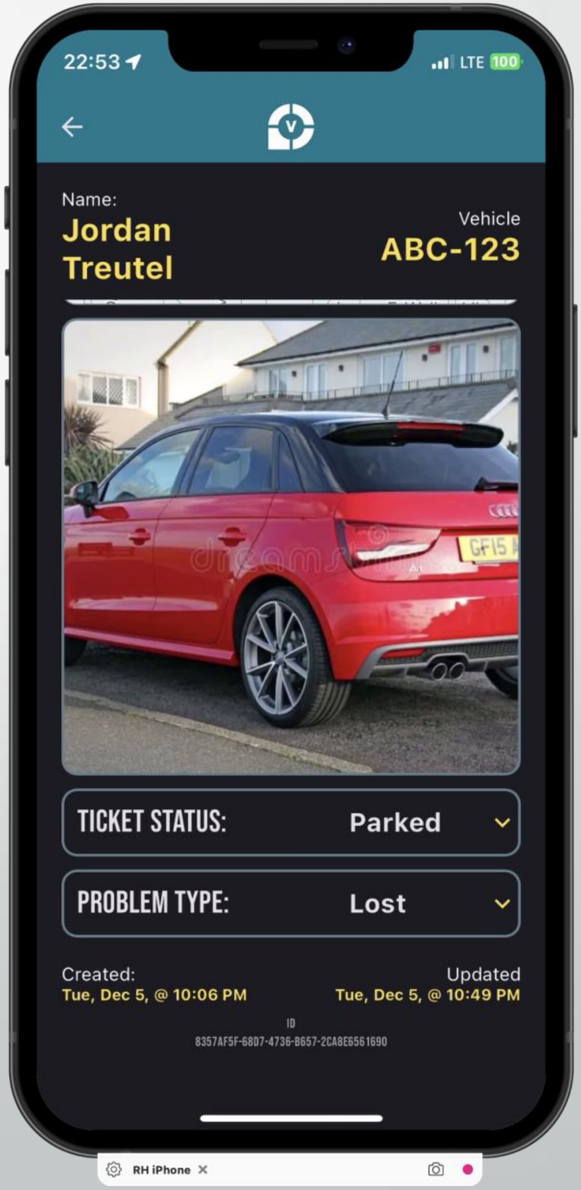
Future Improvements

- Daily database backups
- Switch from locally hosted server to cloud server
- Create map of all car locations
- Have all communications done via 'https'

PB

Conclusion

- Initial MVC architecture as a jumping off point
- Extensively planning out classes and relations helped identify errors and points of confusion before any code was written



Name:
Jordan Treutel

Vehicle
ABC-123

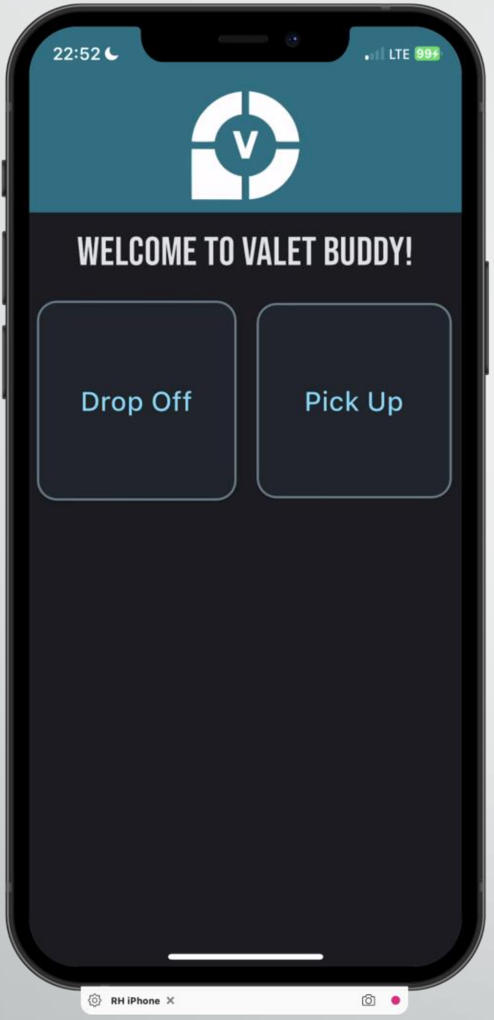
TICKET STATUS: Parked

PROBLEM TYPE: Lost

Created:
Tue, Dec 5, @ 10:06 PM

Updated
Tue, Dec 5, @ 10:49 PM

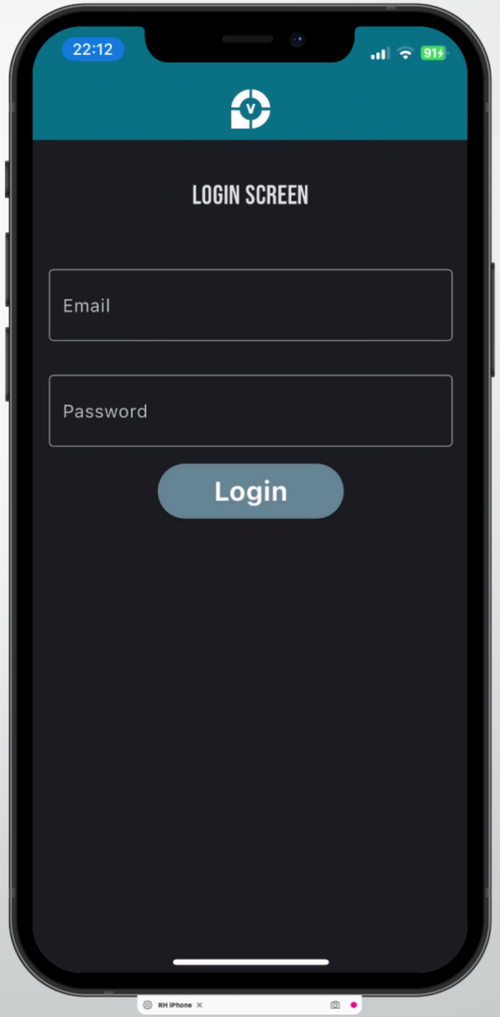
ID
8357AF5F-68D7-4736-B657-2CA8E6561690



22:52 LTE 99

WELCOME TO VALET BUDDY!

App Demo



22:12 91

LOGIN SCREEN

Questions?