CADT IDT

W3 PRACTICE

*PART 1 - GENERIC & SCREEN **WIDGETS***

Learning objectives

- ✓ **Structure Flutter widgets** for extendibility and consistency
- ✓ Comply with **coding conventions**
- ✓ Create a library of widgets aligned with the **product design system**

How to start?

- ✓ Create a **GitHub repository** for this project
- ✓ Get the start code, including the Figma Design System
- ✓ Optionally install the BlaBlaCar **fonts** into your OS for design purpose.
- ✓ Ensure you can run the start project on your computer
- ✓ **Push the start code** into your repository **(**commit: `BLA-000- Start Code`)

How to submit?

- ✓ Each task of this practice shall be related to commit(s) including the **tasks ID.**
  - o As example:

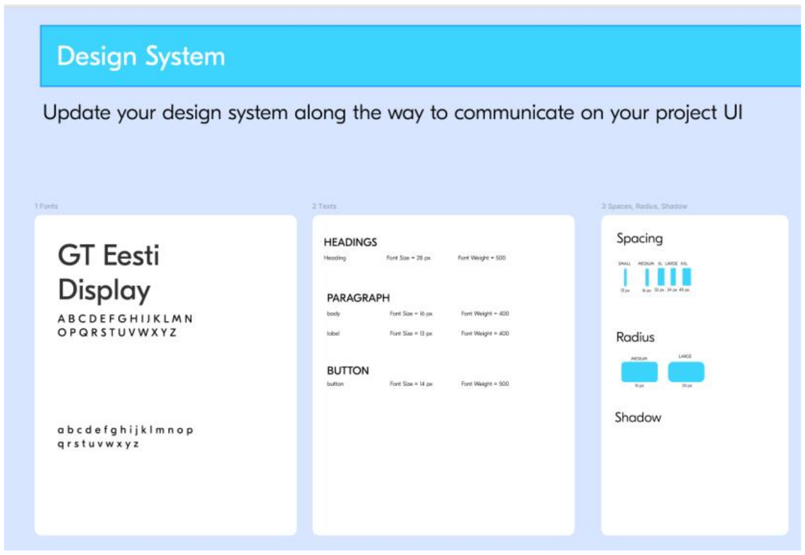
```
BLA-001 - Create the BlaButton
```

- ✓ Once finished, submit on MS Team:
  - o Your GitHub repository URL
  - o This document if needed

BLABLA DESIGN SYSTEM

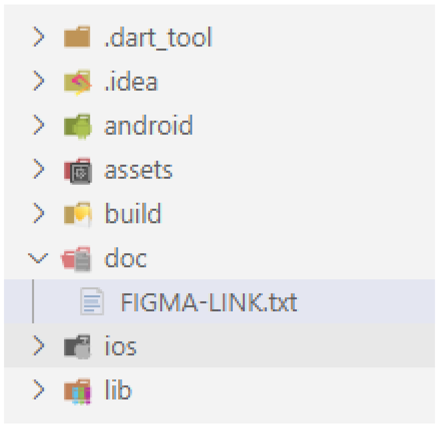
You can find in the /doc folder a link to a start **FIGMA design System** and **user flows**.

- ✓ You can create your **own copy** to it along the development.
- ✓ This file allows you to **identify each widget settings** (color, spaces, icons) and anticipate generic app widget needs.



Design System

Update your design system along the way to communicate on your project UI



```
> .dart_tool
> .idea
> android
> assets
> build
v doc
    FIGMA-LINK.txt
> ios
> lib
```

FIGMA design System and user flows

BLABLA CODING CONVENTIONS

FOLDERS

The project shall be organized around the following folders:

model	Contains data models.
service	Contains service layer code
theme	Contains theming and styling constants
utils	Contains utility functions and helper classes
widgets	Contains reusable app widgets
screens	Contains UI screens and their related components.

MODEL

- Model classes are located in the /**model** folder.
- Models should be **immutable** whenever possible and include:
  - copyWith() method.
  - Proper implementations of == (equals) and hashCode for comparison.
  - A toString() method for debugging purposes
- Models should **only** manage the data structure and its manipulation.
  - No persistence, Flutter, or networking code should be present
- Model classes should be grouped into subfolders based on logical topics:
  - `/model/users`
  - `/model/rides`
  - `/model/ride_preferences`

SERVICE

- Services are located in the /**service** folder.
- For now, services just provide static test data. We will update service later on.

THEME

- Themes are defined in the /**theme** folder.
- theme.dart file define:

Colors	BlaColors
Text styles	BlaTextStyles
Spacing	BlaSpacings
Icons	BlaIcons

- All widgets should reference theme.dart for styling instead of hardcoding styles.

UTILS

- Utility classes are placed in the /**utils** folder.
- These classes contain static methods for common tasks:
  - Formatting dates
  - handling screen animations, etc.

APP WIDGETS

- App (reusable) widgets are placed in the /widgets folder.
- Widgets should be grouped into subfolders based on UI categories:
    - **actions/** (e.g., buttons)
    - **inputs/** (e.g., text fields)
    - **display/** (e.g., cards, lists)
    - **notifications/** (e.g., snackbars, alerts)

- Naming convention: App Widgets should be prefixed with the app's short name.
    - `bla_button.dart`

SCREENS WIDGETS

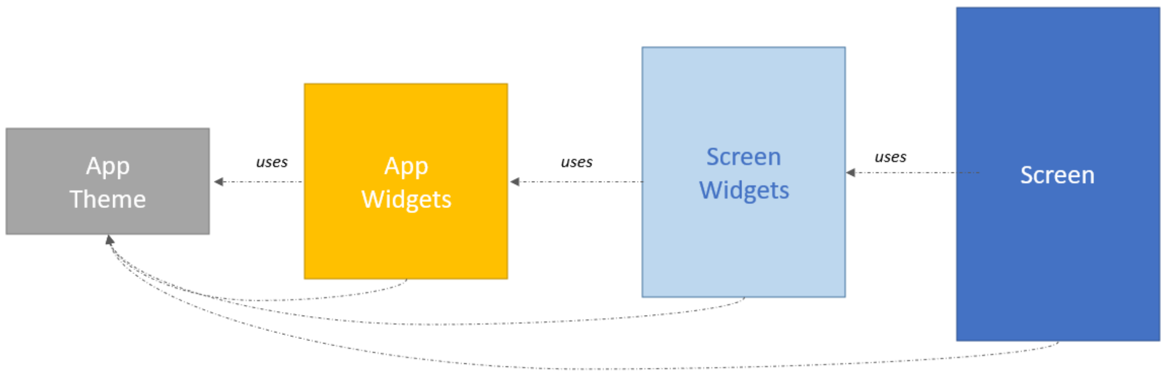
- Screens are located in the /**screens** folder.
- Each screen has its own subfolder: /screens/{screen_name}/
- Screen-specific widgets are placed in /screens/{screen_name}/widgets/

```
/screen/ride_pref/widgets/ride_pref_form.dart
```

- Naming convention:
    - Screen widgets should be prefixed with the screen name.
    - Example:
        - A history tile widget for the ride preference screen: ride_pref_history_tile.dart

APP WIDGETS VS SCREEEN WIDGETS

- The bellow diagram defined the dependencies between widgets and themes.
- All screens and widget shall refer to the App theme constants for color, text styles….



COMMENTS

Three types of comments are required:

Explaining a class…

```
/// This screen allows users to:
/// - Enter ride preferences and launch a search.
/// - Select previous ride preferences and reuse them.
```

Explaining Statements…

```
departure = null;                    // User must select the departure
departureDate = DateTime.now();      // Defaults to now
arrival = null;                      // User must select the arrival
requestedSeats = 1;                  // Default: 1 seat
```

Clarifying Steps…

```
// 1 - Notify the listener
widget.onSearchChanged(newText);

// 2 - Update the cross icon
setState(() {});
```

NAMING CONVENTIONS

Identifiers

Class	UpperCamelCase
Methods	lowerCamelCase
Variables	lowerCamelCase
File Names	lowercase_with_underscores.dart

Class names, Enums, typedefs, and type parameters **should capitalize the first letter of each word.**

Getters

- Use getters to expose computed values from the model.

```
bool get showArrivalPlaceholder => arrival == null;
String get dateLabel => DateTimeUtils.formatDateTime(departureDate);
```

Type explicitly

- Always specify types explicitly.

  Bad:

```
  final dynamic initRidePreferences;
```

  Good:

```
  final RidePref initRidePreferences;
```

Naming Best Practices

- Use terms consistently.

  Good:

```
pageCount // A field.
updatePageCount() // Consistent with pageCount.
toSomething() // Consistent with Iterable's toList().
```

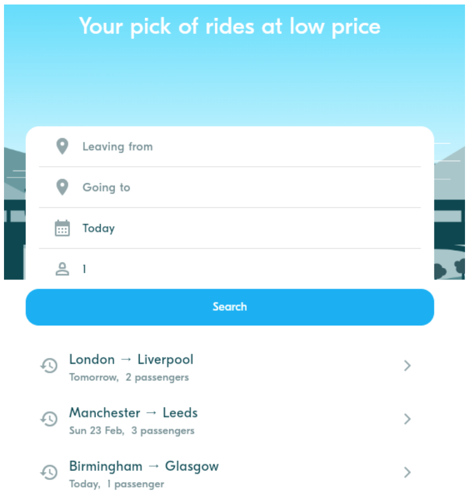
  Bad:

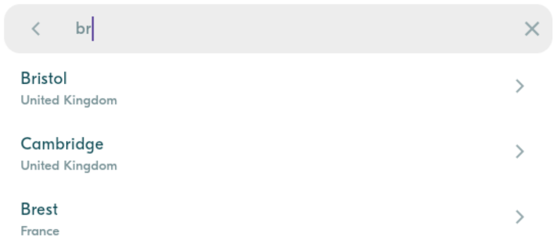
```
renumberPages() // Different from pageCount.
convertToSomething() // Inconsistent with toX() precedent.
wrappedAsSomething() // Inconsistent with asX() precedent.
```

OBJECTIVES FOR THIS PRACTICE

For this first iteration, we want to handle **the Ride Preferences screen**:

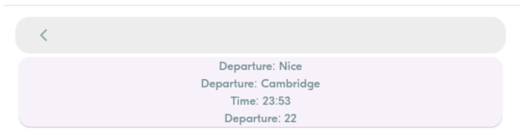
- To input the locations, date and seats and press on Search
- Or to click on a past entered **Ride Preference**





You need to implement the Rides Preferences screen but also the related input modals

- Once selected, we should navigate to the **Rides screen**, displaying all rides matching with preferences (fake view for now):



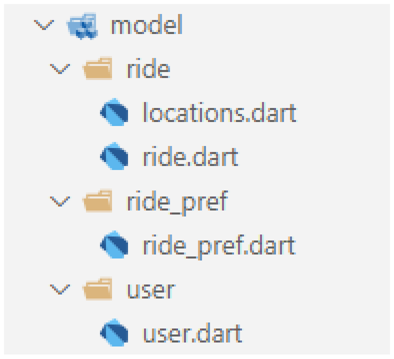
The Rides screen is just a fake screen for now

This practice focusses more **on clean code structure** rather than Flutter technical skills.
We will evaluate how well **you follow the coding conventions,** how your name your class, variables etc.

BLA-001 – Analyze the **models** and the **services**.

Our model so far is composed of **Users**, **Rides**, **Locations** and **Ride Preferences**.

- We have also created a **fake data folder** to store some fake instance for test purpose.



```
model
  ride
    locations.dart
    ride.dart
  ride_pref
    ride_pref.dart
  user
    user.dart
```

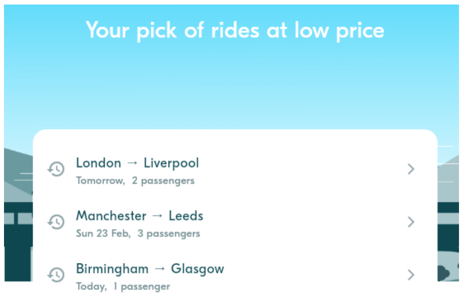
Q1 – Draw the UML diagram of the **current model**

Q2 – Draw the UML diagram of the **3 services** (*rides, locations service and rides preferences)*

Q3 – Test the services: write a main () that display the list of rides **available today**.

BLA-002 – Analyze **Ride Preferences** screen

We have created a first version of the Ride Preferences screen, showing only the past entered ride preferences. Let's analyze the start code.


Your pick of rides at low price

London → Liverpool
Tomorrow, 2 passengers

Manchester → Leeds
Sun 23 Feb, 3 passengers

Birmingham → Glasgow
Today, 1 passenger

Q1 – App font

Explain how the font (*Eesti*) is loaded from the assets and is applied to all widgets.

Q2 – Widgets

Analyze the Ride Preferences screen and complete the table.

Widget	Screen / Screen Widget /App Widget	Parameters	Callbacks
RidePrefScreen			
RidePrefForm			
BlaBackground			
RidePrefHistoryTile			

Q3 – App Theme

Analyze how the **history tile** interacts with **the App theme**.

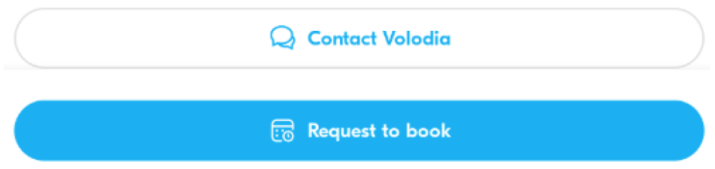
History Tile	BlaTextStyle	BlaColor
title		
subtile		

Q4 – Date Formatting

Explain how the **date** is converted into a readable label

BLA-003 – Implement BlaButton

The BlaButton is used in many places in the application.



The BlaButton widget shall handle primary and secondary, without or with icons…

Q1 – First identify the **possible variations** of this button and complete the table

Widget	Screen / Screen Widget /App Widget	Parameters	Callbacks
BlaButton			

Q2 – Then implement the button and **test** all its variations using a **test screen**.

Q3 – Once validated, commit the code with the proper commit message.

```
BLA-003 – Implement BlaButton
```

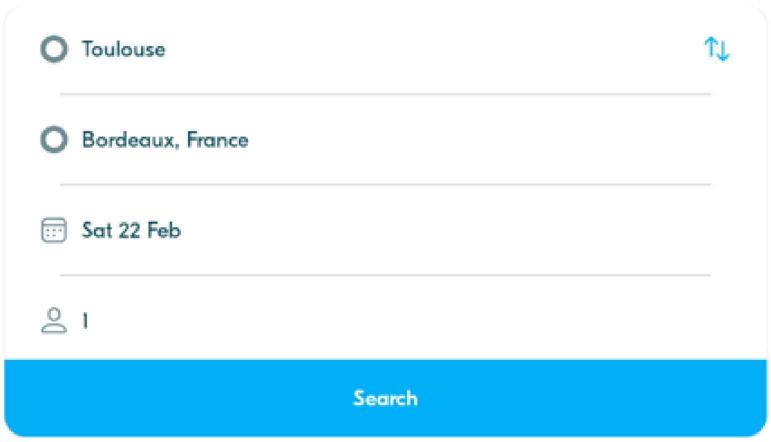
BLA-004 – Implement Ride Preferences Form

The ride preferences from can be used either in the **Ride Preferences** screen (*as a screen component)* or in the **Ride screen** (*as a top modal dialog*)

⚠️ Its s important to ensure this component **can be used on both screens**

⚠️ You need to code the **component exactly as in the real app**.

What are the possible colors in the input fields ? The mandatory and optional icons? Actions?



Q1 – What are the widget parameters? What are the default values?

Q2 – What is the condition to validate the Search button? What type of data is popped if valid?

Q3 – What are the **form sub widgets** and **app widget** you plan to use for this form?

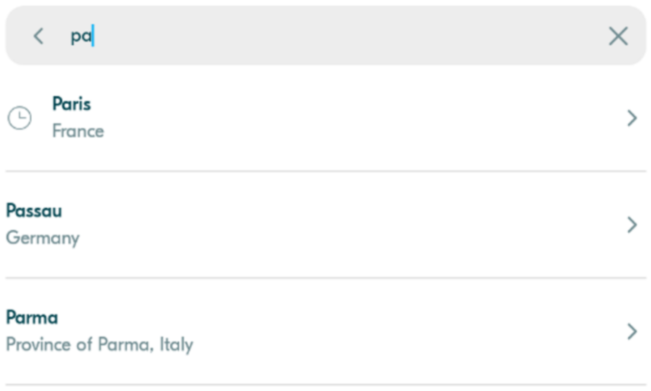
Widget	Screen / Screen Widget /App Widget	Parameters	Callbacks
BlaButton	App Widget		

Q5 – How to implement the **switch location action** () in a clean way?

Q6 – Implement the widget and **test** all use cases.

BLA-005 – Implement the Location Picker

The location picker will be used in many views (as a passenger as well as a driver).


pa

Paris
France

Passau
Germany

Parma
Province of Parma, Italy

Q1 – Analyze the real app picker behavior

How many actions can be done? When the list of locations is displayed? Etc.

Q2 – What are the **picker sub widgets** and **app widget** you plan to use for this form?

Widget	Screen / Screen Widget /App Widget	Parameters	Callbacks

Q3 – Implement the widget and **test** all use cases.

BLA-006 – Add a Tween animation to show the Location Picker

The Location Picker shall be displayed with a bottom to top transition (slide).

Q1 – Learn how to use Tween and offset for animated transitions.

Q2 – Complete the code on animations_util.dart, to create a bottom to top transition.

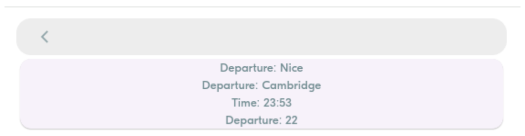
```
static Route<T> createBottomToTopRoute<T>(Widget screen) {
  const begin = Offset(0.0, 0.0);         //  TODO Change this line
  const end = Offset(0.0, 0.0);           //  TODO Change this line
  return _createAnimatedRoute(screen, begin, end);
}
```

Q3 – Use this route to **navigate** to the **Location Picker** form the **RidePreferenceForm**.

BLA-007 – Implement a (*first draft of*) **Rides Screen**

Once ride preferences have been selected, the Rides Screen shall be displayed with the matching rides.

For now, we just want to know if it works…



Departure: Nice
Departure: Cambridge
Time: 23:53
Departure: 22

Not the final look, but just to test the preferences

Q1 – Widgets

Analyze the Ride screen and complete the table with your widget strategy

Widget	Screen / Screen Widget /App Widget	Parameters	Callbacks

Q2 – Implement the widget and **test** all use cases.

BLA-008 – Implement the Seat number spinner

Following the same methodology, you can as a bonus implement the seat number spinner.
It shall render and behave exactly as in the real app.

BLA-009 – BONUS - Implement the Date picker

Following the same methodology, you can as a bonus implement the date picker.
It shall render and behave exactly as in the real app.