POLITECNICO
MILANO 1863

Design Document

Design and implementation of mobile applications

Professor Luciano Baresi

Academic Year: 2023/2024

Team composition

Sadra Heidary Moghamad

sadra.heidarimoghadam@mail.polimi.it

Computer Science & Engineering Master's degree

Angelo Maximilian Tulbure

angelomaximilian.tulbure@mail.polimi.it

Computer Science & Engineering Master's degree

Table of Contents

1. Introduction

In today's digital era, treasuring the genuine essence of our journeys has become increasingly challenging. Our collective aim with TripTales is to bridge this gap by offering a fresh approach to relieving travel memories.

Travel memories are nowadays often fragmented, leading to the loss of the authentic essence and emotions associated with memorable journeys. The proliferation of random photos, scattered videos, and transient social media content fails to encapsulate the true essence of travel experiences. As time progresses, these memories gradually fade, leaving travellers unable to vividly recall the distinctive moments, emotions, and connections made during their trips.

TripTales allows users to convert their travel memories into engaging narratives. By seamlessly merging images and videos from their journeys into a unified endless canvas, every moment, from tranquil sunsets to vibrant streets, becomes vivid and easily recalled with a quick glance.

At the end of the journey the user can see what he/she has done day-by-day, all in the same canvas and relive those moments.

1.1 Purpose

The purpose of this Design Document for the TripTales mobile application is to offer a detailed description of its main architectural components, showcasing how they function during runtime, interact with one another, and present their interfaces. This document primarily serves developers and testers involved in understanding the application's structure and behaviour.

1.2 Application functionalities

The goal of our application is don't let the memories, emotions and priceless moments experienced during a trip fade away, whether they were experienced with friends, relatives, loved ones or alone.

Starting the application, the users will find a home page. This is the first page of the application. It allows the users to go to the login page or if they don’t have an account to create it.

If the users already have an account, tapping on the login button will bring them to the login page, that will allow them to insert their email address and their password.

If the users need to create a new account, tapping the create account field will bring them to the register page, a much denser page in comparison to the login one, because the application requires more details about the users, such as their date of birth, name, surname, email address and double password confirmation.

After a successful login, the users will be routed to a navigation bar containing the dashboard, the favourite tales page and the user profile page.

This last page is showing all the users' information such us profile picture, name, surname, date of birth, email, password, phone number, gender and bio. The users can modify all these parameters after tapping the "edit profile" button and then, they can save the modifications tapping on "save modifications" button. Automatically the data will be updated on the DB, so the users will see the updated information every time they access the app again.

The Favourite Tales page is a view of the favourite memories of past trips the users have lived in the past. That's an helpful tool to navigate quickly through the most appreciated tales, explore them and relive the joy as it was experienced few seconds before.

The dashboard is the central hub providing a view of the past tales, sorted in chronological order from the most recent one. Here, at the top of the page, the users can create a new tale.

This will redirect them to the Create Tale page where the users can choose the tale name, the tale image that will be shown in the dashboard page, and choose the best canvas that is going to be the background, based on the vibes of the trip and the feeling the city is giving to the users.

After these actions the users are ready to fully create their tales, leaded by their imagination and creativity. They can create a personal journal of the trip by noting down their thoughts and emotions the city they are visiting are giving to them. They can take pictures of the details they like the most or upload a video of a unique experience. They can additionally set the location on a map of their memory and see them in the Tale info section.

The real innovation of TripTales is the possibility to resize and position all these items in the order they fit the best their inventiveness and the no space limit because the app is automatically adding more vertical space when the users are about to finish the space, so they never run out of space.

There are two possible modes: View or Edit. The first one is allowing the user to only scroll down the tale, enjoying what he/she has created so far, while the second mode is allowing the users to modify

the tale, adding new items, resizing them, move them all around the canvas, delete and reorder the layers of the different items.

In the Tale page the user can see the Tale info, a page showing the name of the tale, its image and a map showing all the cards' location present in the current tale. The users can always save a draft of their tale and keep modifying it in any other moments, also when the trip is ended.

1.3 Features

The application offers a wide range of features designed to enhance the user experience and facilitate creating and storing memories while travelling.

- **Register:** If the user has no account he/she can create a new account by writing the name, surname, email, birthdate, and password.

- **Login:** The user can log in with email and password. If it's already registered

- **Login with Google:** The user can log in with a Google account.

- **View tales:** possibility to see a list of the created tales and their name and images.

- **Like Tales:** users can like tales to have easier access to them.

- **View favorite tales:** possibility to see a list of the created tales, that are liked, with their name and images.

- **View Profile:** the user can see all the related information and profile pictures on the profile page.

- **Profile personalization:** a tool that allows the users to modify the profile information.

- **Image profile change:** possibility to add, modify, and delete the profile picture of the users.

- **Dynamic camera image controller:** possibilities to use the camera or browse the gallery to upload images or videos.

- **Password strength:** a visual tool that allows the users to see the strength of their password, checking if it contains capital letters, special characters, and numbers.

- **Logout:** ability to log out of the current account to log in with a new one later.

- **Delete Account:** users can delete their account if they want to remove all of their related data from the TripTales database.

- **Create a new tale:** the ability to create a new tale that can contain multiple cards. The user must choose a name, picture, and a canvas(background) for the tale.

- **Canvas selection:** wide range of tale backgrounds from which the users can choose, depending on which one is fits their ongoing journey best.

- **View mode Tale:** possibility to set the Tale in view mode only. In this mode, no further modifications to the tale are allowed.

- **Edit mode Tale:** possibility to set the Tale in edit mode. In this mode the users can modify the tale, adding new items, resizing them, changing their rotation and position, and modifying the order of the layers.

- **Create image card:** The user can create a card appearing as an image in the tale. To create an image card, the user should write the name, choose an image, and choose the location of the card.

- **Create video card:** The user can create a card appearing as a video in the tale. To create a video card, the user should write the name, choose a video, and choose the location of the card.

- **Create text card:** The user can create a card appearing as a text in the tale. To create a text card, the user should write the name, and the text, modify the text, and choose the location of the card.

- **Pick colors:** For the text card creation users can choose a color to put as the text color and another for the background of the text.

- **Choose a location using Google Maps:** In the process of creating a card, it is possible to choose the location, where the memory card was made. To do this user needs to just click on the map to set a location and submit.

- **See the tale information:** It is possible to see the tale name, picture, and all the places, that were chosen while creating cards. The map can be pressed so the user can have a more clear view of the map and the places he/she went.

- **Edit Tale:** The users can edit the image, name, and canvas as they want.

- **Delete Tale:** the possibility of deleting a tale and all their cards.

- **Search locations on the map:** The ability to search the place on the map. When the user searches the map will move to the searched location.

- **Scaling cards:** possibility to resize the ratio and the dimensions of the cards. To apply the changes, the submit button must be clicked.

- **Rotating cards:** The possibility to rotate the cards in every possible position. To apply the changes, the submit button must be clicked.

- **Moving cards:** possibility to move the cards on all the canvases. To apply the changes, the submit button must be clicked.

- **Reorder cards:** possibility to reorder the cards if they are placed on each other and the user wants to put a card on top of the other. Users can access this feature by clicking on the reorder button in edit mode.

- **Delete cards:** users can delete the cards that they don't want or need anymore. Users can access this feature by clicking on the reorder button in edit mode or simply clicking on the trash bin on the top right of a card.

- **Image customization**: possibility to trim away unwanted parts, focusing on the essential elements. Cropping allow users to fine-tune the visual narrative. Image rotation allows users to change the orientation of an image by turning it clockwise or counterclockwise around its centre. Zooming in and out of images provides several benefits. Zooming in allows users to closely inspect specific areas of an image, enabling a more detailed view, focusing attention on particular parts. On the other way, zooming out provides an overview of the entire image, allowing users to evaluate its composition, structure, and overall appearance. It's also helpful for scaling images to fit within a certain display or for publication. For an user interfaces purpose, implementing zooming functionality ensures images adapt to different screen sizes and devices, maintaining clarity and usability.

- **Text customization:** Text customization in images involves altering text elements to suit specific preferences or requirements. It's like tailoring words to fit the exact style or message desired for a particular image. This can involve changing the font, size, style, colour, alignment, and effects applied to the text. You can experiment with different fonts to evoke different moods, adjust the size to ensure readability, play with colours to match the overall theme, and apply effects like bold, italic, or outlined to make the text stand out.

- **10. Video customization:** possibility to scale the video ratio, based on the size the users want the video to have in the canvas.

1.4 Acronyms definitions and abbreviations

DB: database, the place in which we store the data we want to keep for every session.

API: Application Programming Interface, a way in which is possible to communicate with another systems.

OS: Operating System, the main software that manages the device hardware and software resources, it provides services to other programs and applications.

UX: User eXperience, the way in which is defined how a user interacts with a product, system or application.

UI: User Interface, the components through which the interaction between user and application happens.

App: application.

TripTales: the name of the app, we will use It to refer to the final product.

Tale card: a component that include the background image.

Canvas: background image on where the user insert customized images, videos and text.

Custom Text field: A component that allows the user to write an input as text. This text can be used by the application logic for any operation.

Custom Button: A component that allows the user to tap an input as text. This text can be used by the application logic for any operation.

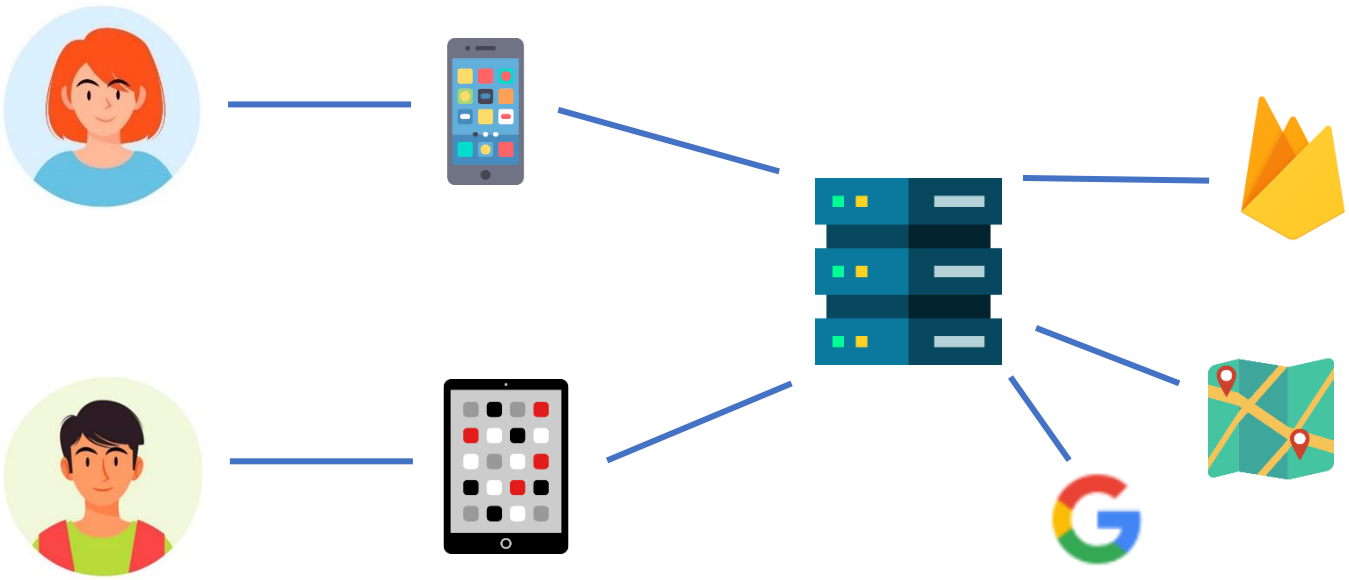
Bottom Navigation Bar: A component positioned on the bottom side of the screen that allows the user to change screen. It is composed of a variable number of tabs where each one corresponds to a specific screen.

Sliver App Bar: A component positioned on the top side of the screen that allows the user to go the previous screen and it automatically hides when the user scroll down the screen.

2. Architectural Design

2.1 Overview

From a high-level point of view, a three-tier architecture appears as a good choice for the system. This is a well-known architecture organized in three logical and physical tiers:



1. **The Presentation Layer (View)** is managed by the Flutter framework, responsible for the user interface and interaction. It furnishes visual components, handles user input, and orchestrates the overall user experience.

2. **The Business Logic Layer (Controller)** houses the application-specific functionalities and business logic. It integrates APIs like Google Maps, Google Sign In, interacting with the presentation layer to manage user requests and data processing.

3. **The Data Access Layer (Model)** is represented by Firebase, serving as the architecture's data infrastructure. It includes services such as authentication, real-time database management, cloud storage, and functions. Firebase governs data storage, retrieval, and synchronization between client applications and server systems.

The advantages of a three-layer architecture encompass:

1. **Modularity**: Changes in one layer typically don't affect the others, enhancing modularity.

2. **Scalability**: Each layer can be scaled independently, allowing for efficient resource allocation.

3. **Separation of Concerns**: It enforces a clear separation of concerns, ensuring that each layer has distinct responsibilities.

4. **Flexibility and Maintainability**: Because of the separation of concerns, it's easier to update or replace specific layers without impacting the entire system. This enhances flexibility and makes maintenance more manageable.

5. **Security**: With a layered architecture, security measures can be implemented at each layer. This multi-layered approach helps in enforcing security measures across the application, reducing vulnerabilities.

6. **Reusability**: Components within each layer can often be reused across different parts of the application or in different applications altogether, promoting reusability and reducing development time.

7. **Testing and Debugging**: The clear separation of layers facilitates easier testing and debugging. Each layer can be tested independently, making it simpler to identify and fix issues.

8. **Collaboration**: Different teams or individuals can work on different layers simultaneously, fostering collaboration and parallel development efforts.

Ultimately, a three-layer architecture forms a robust base for app development, ensuring separation of concerns, maintainability, and scalability.

2.2 Technology Overview

Within our application, we've integrated two pivotal technologies that greatly enhance its functionality and user experience.

2.2.1 Flutter

Flutter stands as the primary framework, prized for its robust support across both Android and iOS platforms. Its utilization ensures a consistent and visually pleasing user interface across various devices. Specifically tailored for larger screens, our UI maximizes available space.

The application necessitates three crucial permissions: access to device location for sharing current or different positions and camera access for uploading pictures. A stable internet connection is fundamental for optimal application performance.

2.2.2 Firebase

Firebase serves as a critical component in our application's architecture, primarily for its real-time data capabilities. Leveraging Firebase ensures a continually updated version of the app's data, providing users with the latest information instantly.

Moreover, Firebase integrates Google and Flutter authentication mechanisms, ensuring secure and streamlined user authentication.

Additionally, Firebase storage efficiently manages user profile pictures. Real-time stream functionality within Firebase manages dynamic information streams like notifications, chats, and lessons, ensuring users consistently access up-to-date and relevant data during their interaction with the app.

The main packages used were ‘firebase_auth’, ‘cloud_firestore’ and ‘firebase_storage’ explained in detail later in the DD.

2.2.3 Supplementary Technologies

Apart from Flutter and Firebase, we've integrated other essential technologies to augment the application's features and capabilities.

- **Google Sign In API:** Google Sign-In API is a secure authentication system that reduces the burden of login for your users, by enabling them to sign in with their Google Account—the same account they already use with Gmail, Play, and other Google services.

- **Google Maps API:** This API enables map retrieval, precise display of user positions, and facilitates location-specific searches. Users benefit from interactive maps, geolocation services, and efficient navigation, significantly enhancing the overall user experience. Granting location permission to the app enables this feature's utilization. When a user is adding a new card to his/her tale, he can select the position. In the Tale info he/she can see all the positions of all the cards added.

2.2.4 External Libraries

External libraries were pivotal in the integration and smooth operation of specific services provided by TripTales app. Here's an overview of some key libraries utilized in the project:

- **Lottie:** a versatile and efficient tool for creating high-quality, interactive animations for a variety of use cases. Some key points about Lottie:
    - *High Quality and Interactive*: Lottie is a JSON-based animation file format that allows for the creation, editing, testing, and collaboration of animations on any platform. It is interactive and responds to user input1.
    - *Small and Efficient*: Lottie files are tiny, can function on any device, and can be scaled up or down without causing any pixelation. This makes them significantly faster than other animation file formats, translating to up to 90 percent faster page loading times.
    - *Versatile Use Cases*: Lottie can be used for a variety of applications, including web interactions, hero animations, emojis and stickers, basic UI elements, and element animations. Whether you want to add motion effects to your social media icons, add hover effects to images, or include scrolling animations, you can do so easily with Lottie.
    - *Easy to Use*: Lottie animations can be easily integrated into a website without concern for performance issues.

- **flutter animate (in combination with Lottie):** A performant library that makes it simple to add almost any kind of animated effect in Flutter.

- **image picker:** powerful tool that allows the user accessing and manipulating images from a device's gallery or camera.

- **image cropper**: useful tool for editing images. It allows users to crop, rotate, and scale images. This is particularly useful for adjusting the aspect ratio of images, removing unwanted portions, or focusing on a particular part of an image.

- **matrix gesture detector:** it is a package for Flutter that maps translation, rotation and scale gestures to a Matrix4 object.

- **video player:** necessary tool for handling video playback. It supports both network and local file system videos. It provides a set of controls for play, pause, and seek operations. It also includes volume control.

- **dropdown button 2:** package that extends Flutter's core Dropdown Button widget with a steady dropdown menu and many options you can customize to your needs1.

- **get storage:** A fast, extra light and synchronous key-value in memory, which backs up data to disk at each operation. It allows to read, write and remove data. It also allow to react whenever a change occurs in the storage.

- **get:** extra-light and powerful solution for Flutter. It combines high-performance state management, intelligent dependency injection, and route management quickly and practically.

- **location**: package that provides a convenient way to access device location services, including obtaining the user's current location and monitoring location changes.

- **permission_handler**: used to request and check runtime permissions. It provides a simple and unified API to handle various types of permissions, such as camera, location, contacts and many more.

- **geocoding**: it allows to perform forward and reverse geocoding, converting between geographic coordinates and human-readable addresses, and vice versa. It is commonly used for working with location-based services.

- **flutter_colorpicker**: it provides a color picker dialog and a color picker form field. It allows users to pick colors interactively.

- **flutter_native_splash**: used to generate and configure native splash screens. It simplifies the process of adding and customizing splash screens.

- **firebase_auth**: is part of the Firebase SDK, and it provides a set of Flutter plugins for Firebase Authentication. Firebase Authentication enables your Flutter app to authenticate users using various identity providers, such as email and password, Google, Facebook, Twitter, GitHub, and more.

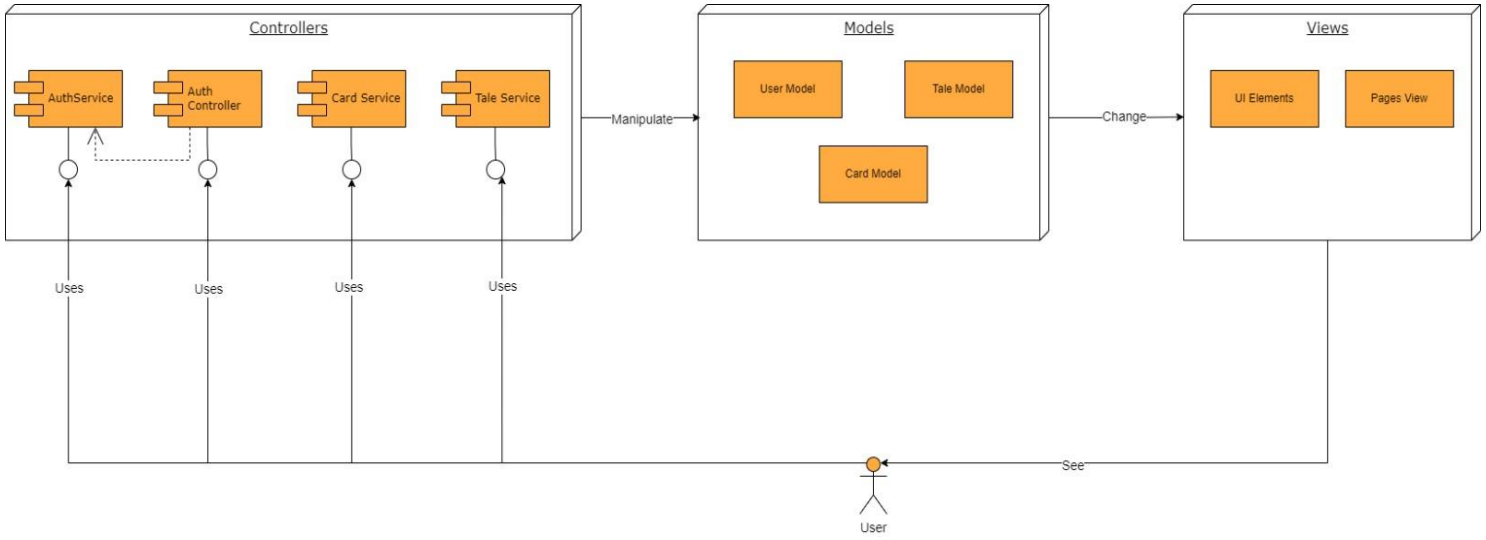
- **cloud_firestore**: it provides a set of Flutter plugins for Cloud Firestore, which is a NoSQL document database offered by Firebase. Cloud Firestore is designed to store and sync app data on a global scale, making it suitable for building real-time collaborative applications.

- **firebase_storage**: provides a set of Flutter plugins for Firebase Cloud Storage. Firebase Cloud Storage is a scalable and secure object storage solution that allows developers to store and serve user-generated content, such as images, videos, and other files, in the cloud.

3. Design Diagrams Overview

3.1 Component View

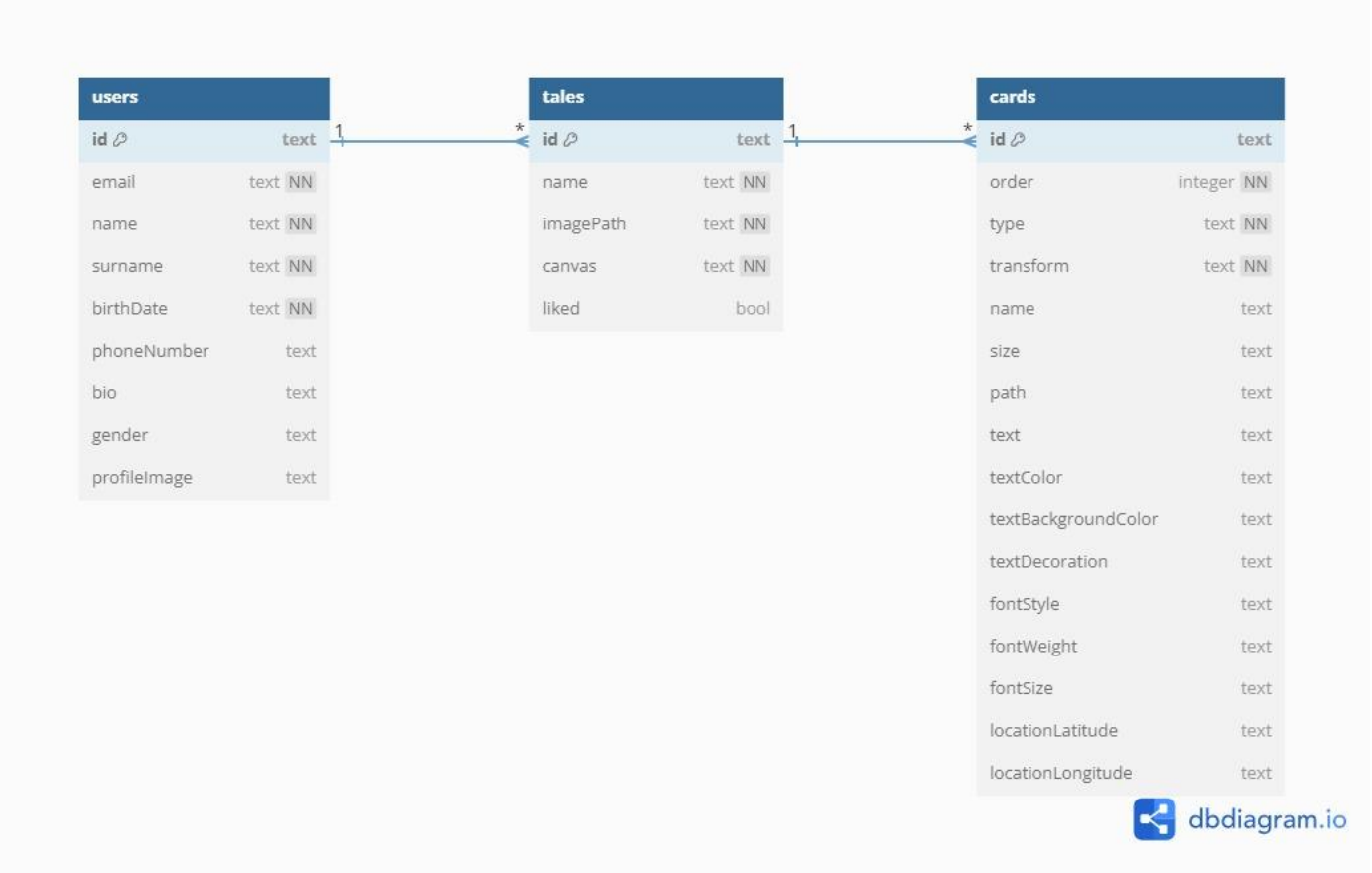
We employed the MVC design pattern as a framework for the architecture, incorporating models for users, Tale and Card, views for each application page displaying pertinent information and interactions, and controllers responsible for authentication, Card service and Tale service.



3.2 Database Entity Relationship diagram

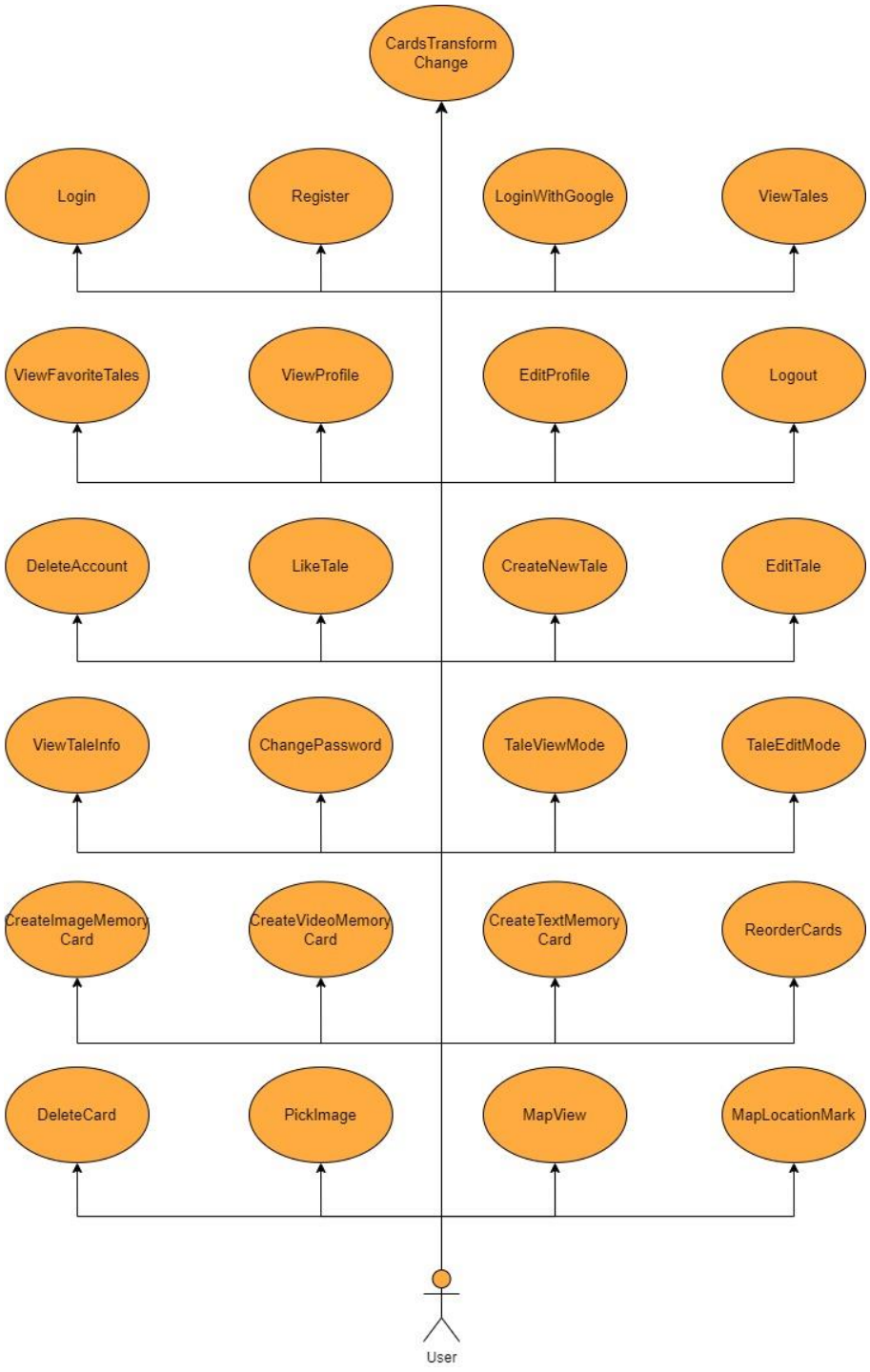
The following database entity-relationship diagram is a visual representation of the entities within the database and the relationships between them.

We can see that every user has one or more Tales, and every tale has one or more cards. With this modelling technique we can observe better the logical structure of TripTales database.



3.3 Use Case Diagram

The following Use case diagram is a visual representation that illustrate the interactions between user and the system, focusing on the functional requirements of the system. It provides a high-level overview of how users interact with the application to achieve specific goals or tasks. We can see that the user can do a variety of different things, that's being key for a high user experience.



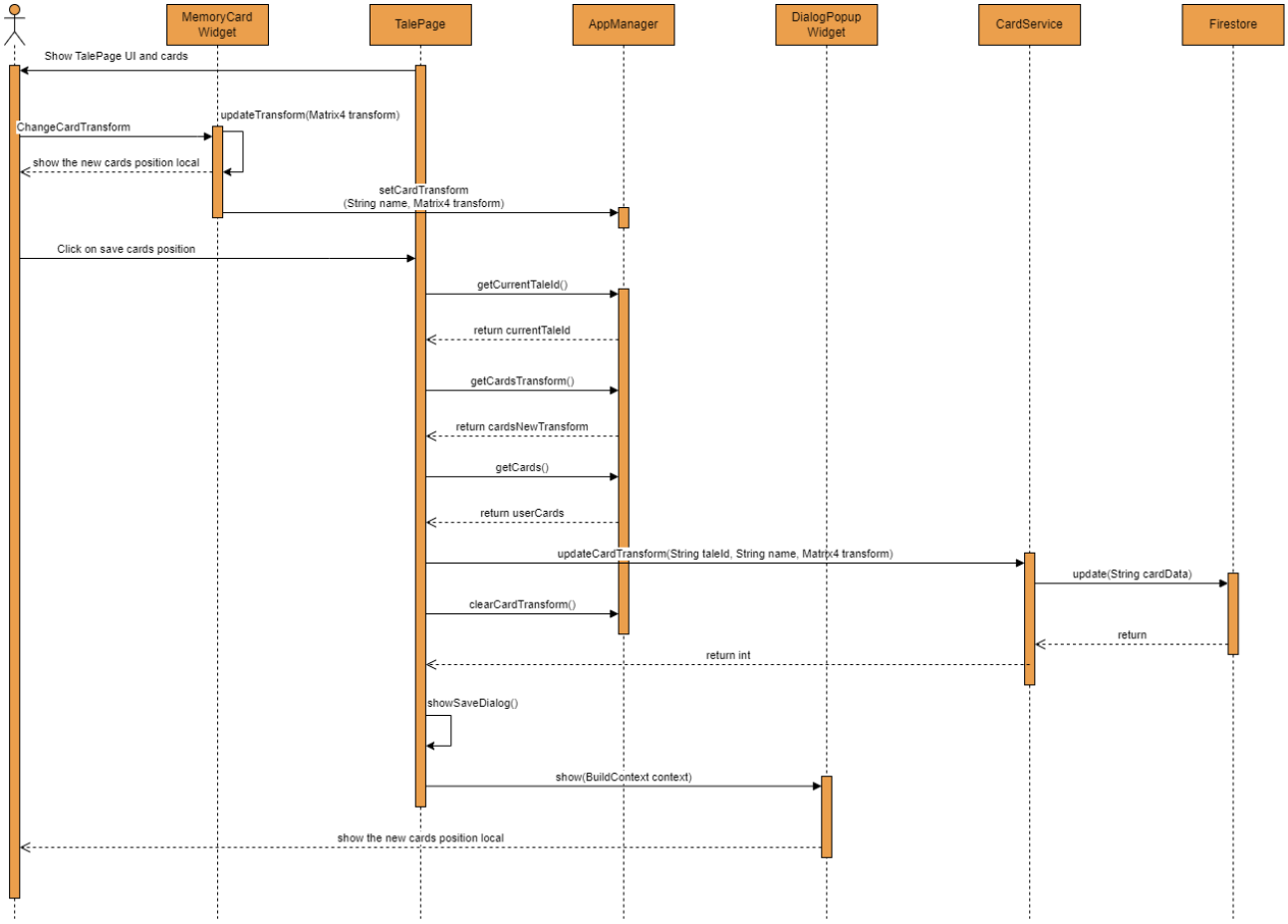
3.4 Sequence Diagram

In the sequence diagrams below, you can see valuable insights into the order of operations and the communication between various components during the execution of a particular use case or scenario.

The below sequence diagrams are made for the three most important features of the application. These features are card transform change (move, resize, and rotate), card creation (in this case image card creation), and show tale info.

3.4.1 Card Transform Change (Move, Resize, Rotate)

This is the most important feature, which helps the user to move, resize, and rotate the created cards and design their tales in the way they want. You can see the sequence diagram for this feature. In this sequence diagram scenario, the user changes the transform (position, sized, and/or rotation) and clicks on save button to save the transform.

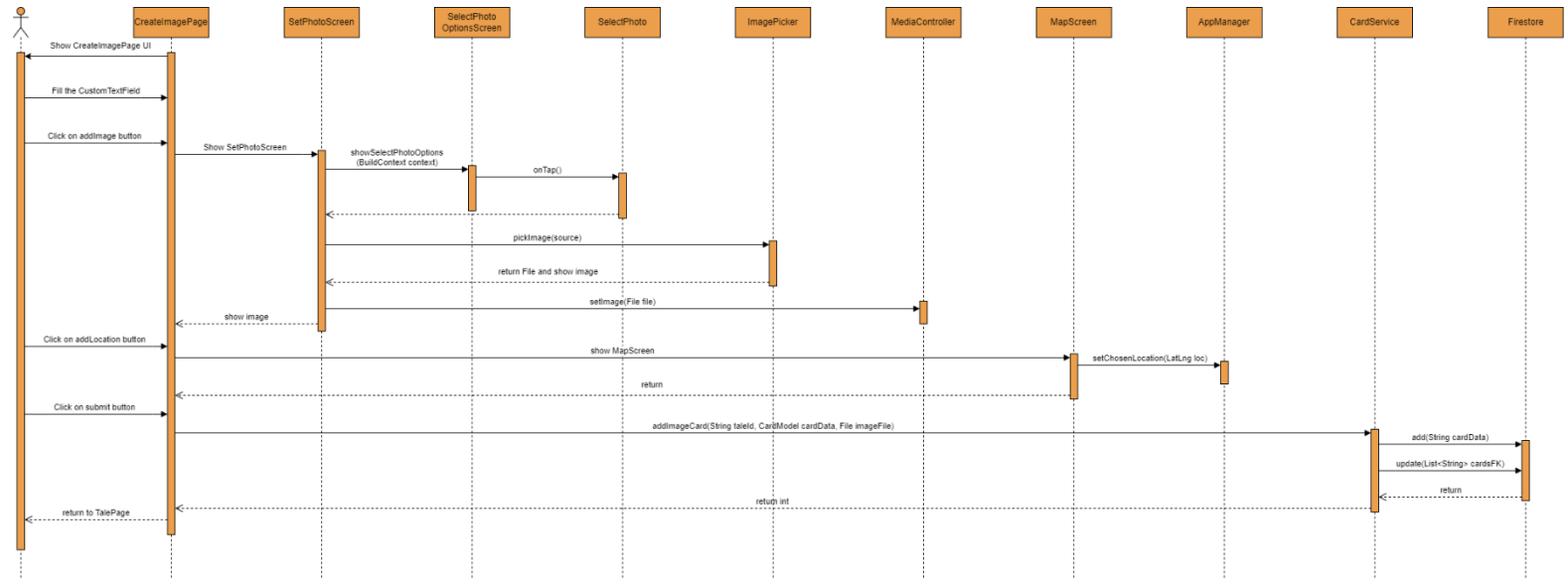


3.4.2 Create Card (Create Image card)

This is the second most important feature, which helps the users to create their cards so they can design and edit it in the tale page as they want.

In this case you can see the sequence diagram relate to image creation, which is pretty similar to the other types of card creation (video, and text card creation).

This diagram shows a scenario in which user fills the name of image, adds an image file, adds a location, and submits.

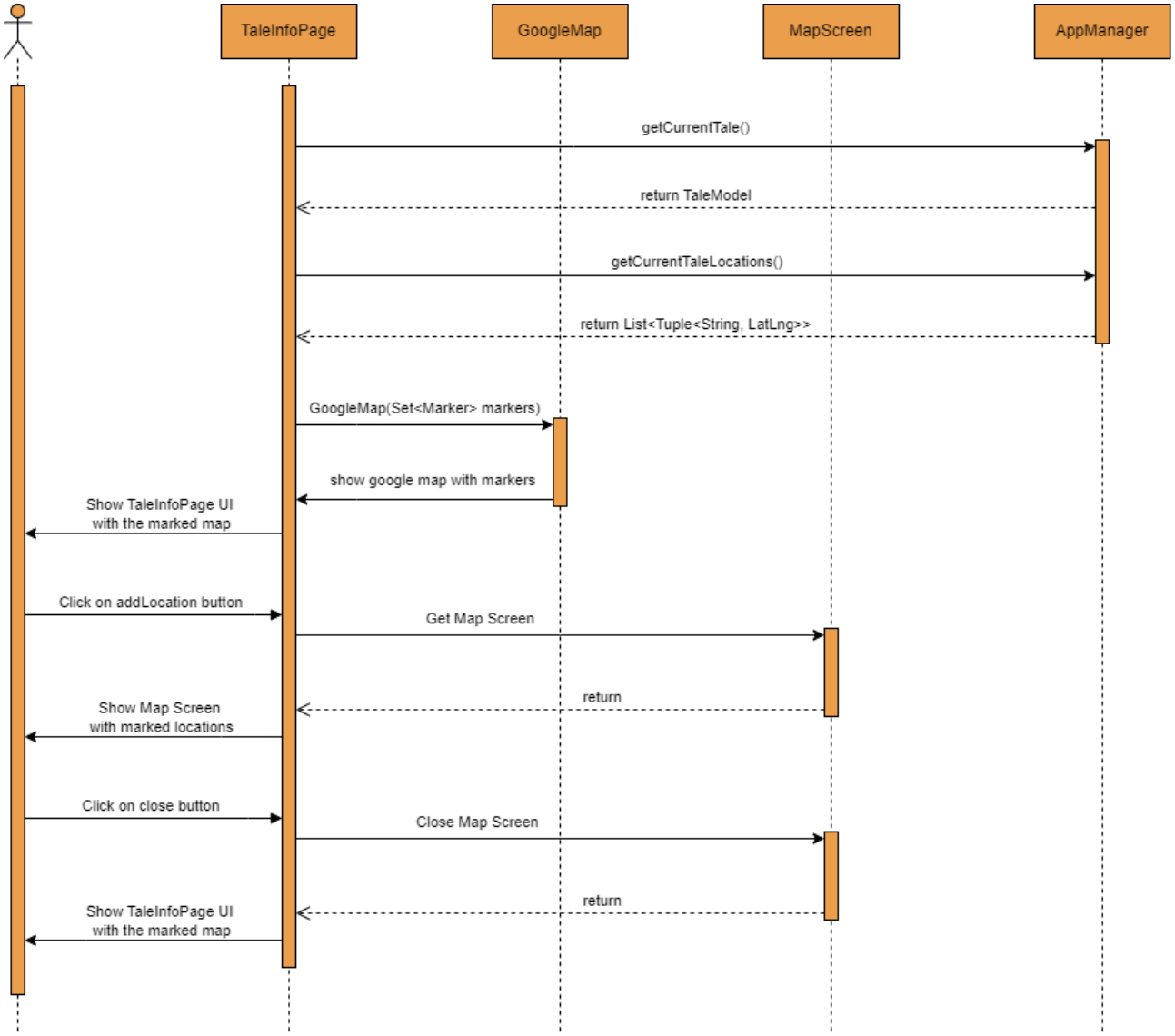


3.4.3 Show Tale Information

This is also one of the most important features, which helps the users to see their tale information such as image, name, and locations that it is visited in that tale. The reason why this feature is important is that users can visualize all the places they have been and have created a card for. So, they can see which places they have been visited in each tale.

It is important to notice that this diagram is a scenario in which user goes to TaleInfo page and then clicks on map and closes the map.

Also, as it can be seen, there is no controller, service, or database in this diagram, while the tale data including the name, image, and locations of the tale have been already saved in AppManager. When users enter a tale, the info of that tale is saved, and when they add or delete a card the locations will be updated.



4. Design

We aimed for an attractive design in TripTales, evoking the colours of out logo, deep purple as main colour light orange for the buttons and some details, and red for error messages. All these three colours where were surrounded by white, which gave a sense of elegance to our app.

We crafted two distinct layouts, one for smartphones and another for tablets, which we'll showcase along with app screenshots. For the best user experience the app could be used only in vertical on smartphones because we want to focus on the endless canvas and allow the users to not put a frame on their memories. The only limit to their creativity is their imagination.

4.1 UX Diagrams (Flow of interactions between the elements)

In this section we will show the various interactions that the user can do while in the application.

The first interaction scenario represents how to set up at first login. The user can create a new account by filling the form, sign up with a google account (in that case the app will take all his/her details from the google account) or, if the user has already an account, to log in just by inserting the email and the password. By using the navigation bar, the user can navigate through the different pages.

If the user is new, in the dashboard he/she will not see any tale, same thing for the Favourite Tales page, as it’s the first time the user is using the app, and he/she didn’t create anything yet. The only thing that the user can see is the personal details of the profile account. (4.1.1)

The second interaction scenario is showing how the user can change the personal details in the profile page. The user can edit the profile by tapping the ‘Edit Profile’ button. Now the fields are modifiable (if the user is not tapping that button the fields are view only). The user can additionally change the password of his/her account by tapping the profile text field. After that the user can insert a new password, confirm it and see the strength of the password he/her just insert. After tapping on change the user’s password is updated. Moreover, the user can change the profile picture by tapping the camera icon. (4.1.2)

The third interaction scenario is showing how to create a new Tale. The user after tapping on the plus button is redirected in the create tale page. Here he can select the Tale image that will be shown in the Home, the tale’s name, and the canvas he wants to be the background of his tale. The user

can select the image from the gallery or use the camera. After selecting the image, he will be able to crop it in order to select only the parts he wants. (4.1.3)

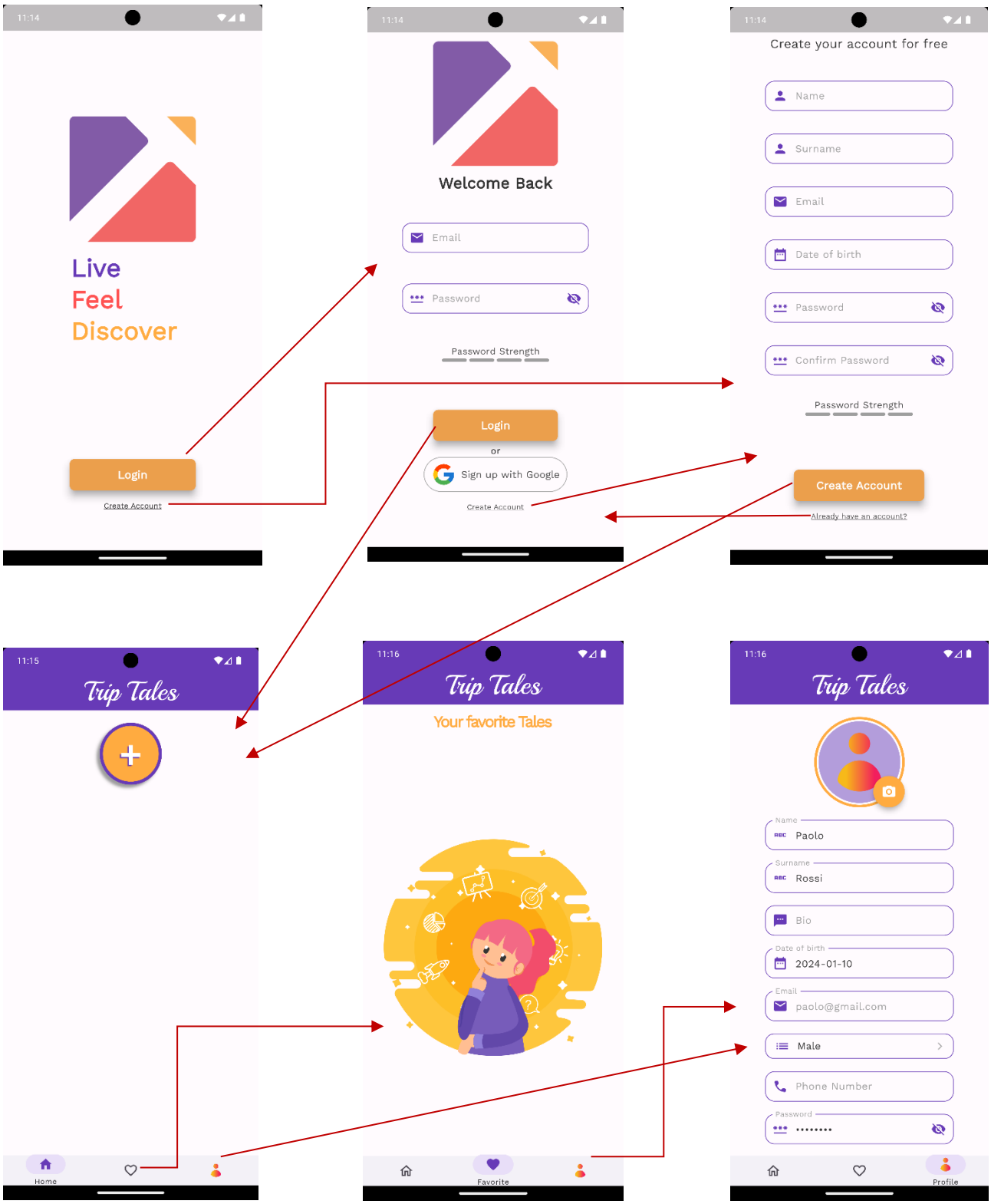
The fourth scenario is showing how to modify a new tale. First the user must select the Edit mode in order to be able to add new images, videos or text. After selecting the favourite media, he can tap on the “Location” button and select on the map the place that media was made. In the case of text, the user has e wide variety of features he can add to his text. (4.1.4)

The fifth scenario is showing how to reorder a Tale. By selecting the edit mode, the user can then select the “Reorder” button. After that he can see the actual order of the cards in the current tale. By changing the order, he can put one card or parts of it over the other. After tapping on close the tale will show the current order of the cards. (4.1.5)

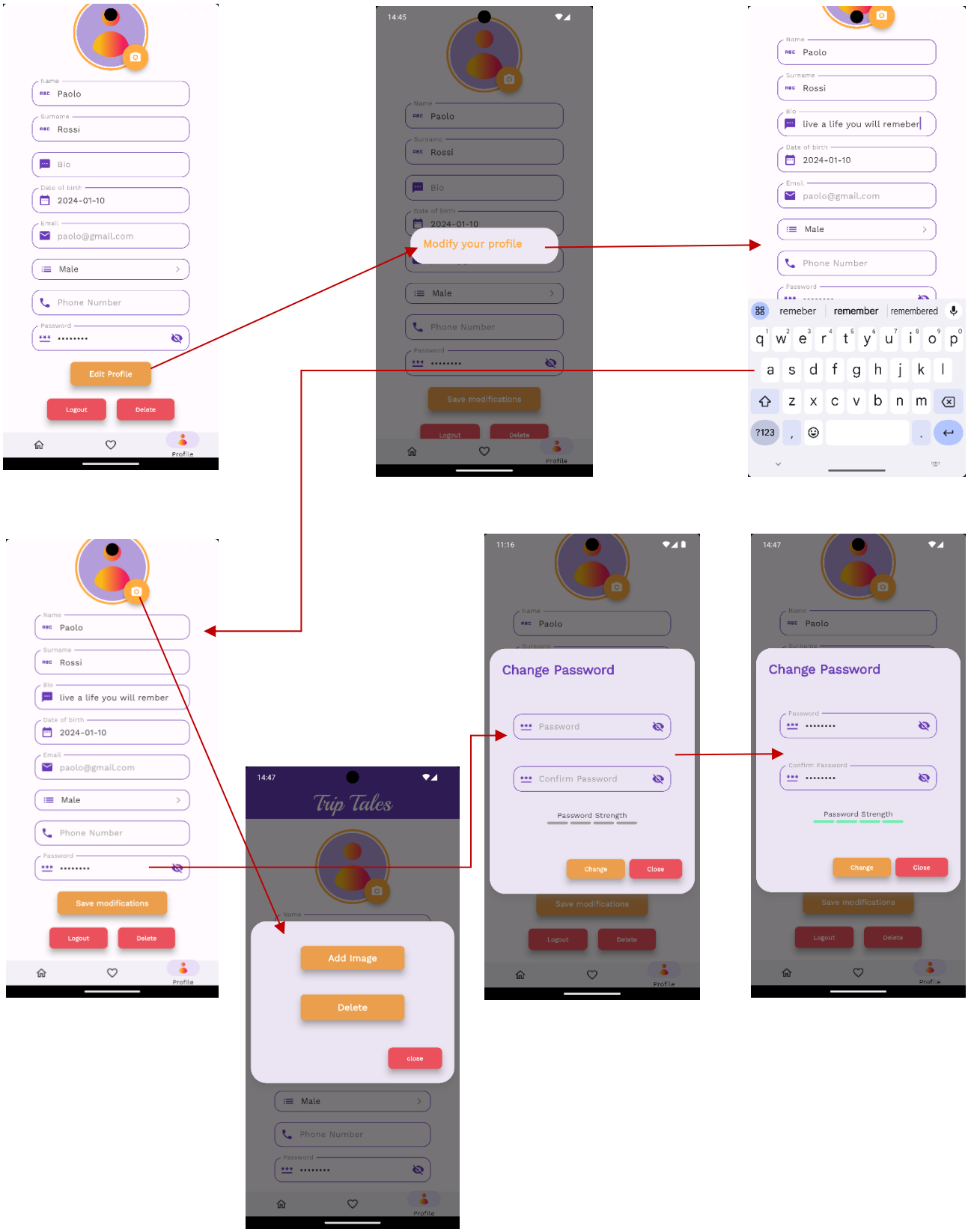
The sixth scenario is showing the tale info’s. By tapping on the “Tale info” button the user will see a new page, showing the current tale image, its name and a map showing all the points he put the cards. By tapping on the “Edit” button he can change the name of the Tale and its image. (4.1.6)

The seventh scenario is showing all the tales in the home page. By tapping on the “favourite” icon in one or more tales the user will see them (and only them) in the favourite tales’ page. (4.1.7)

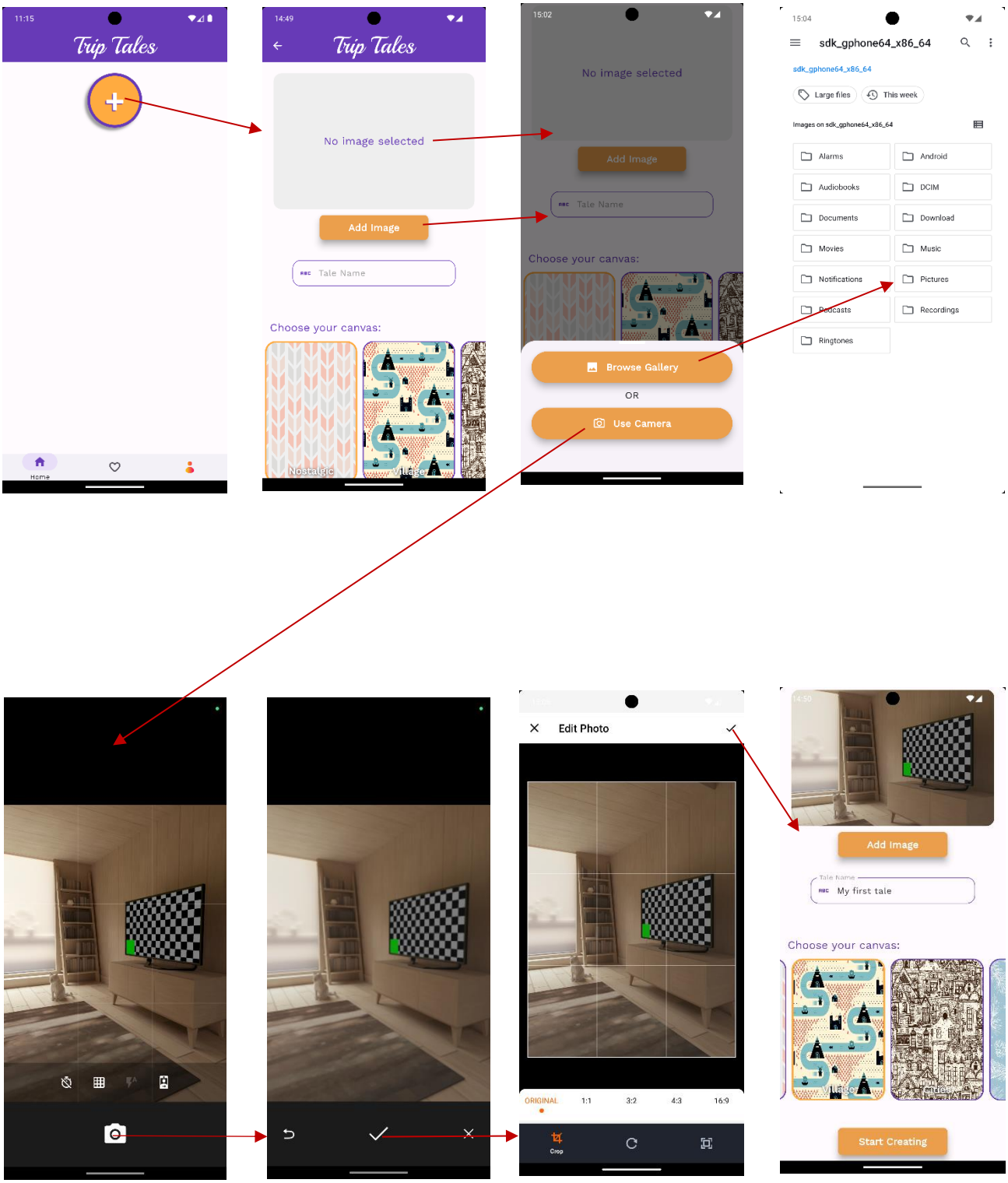
4.1.1 Setup at First Login



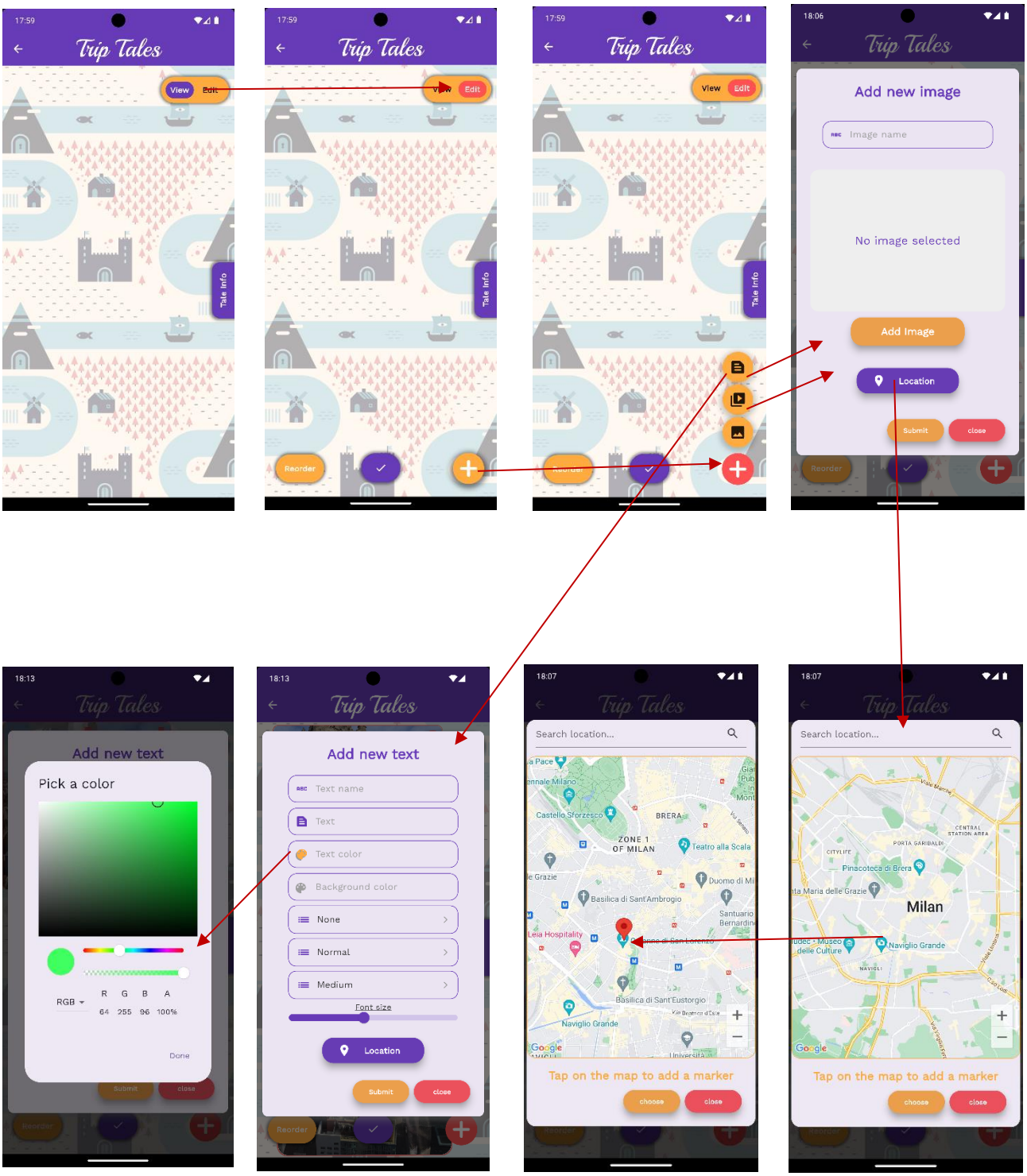
4.1.2 Modify User details and Change password



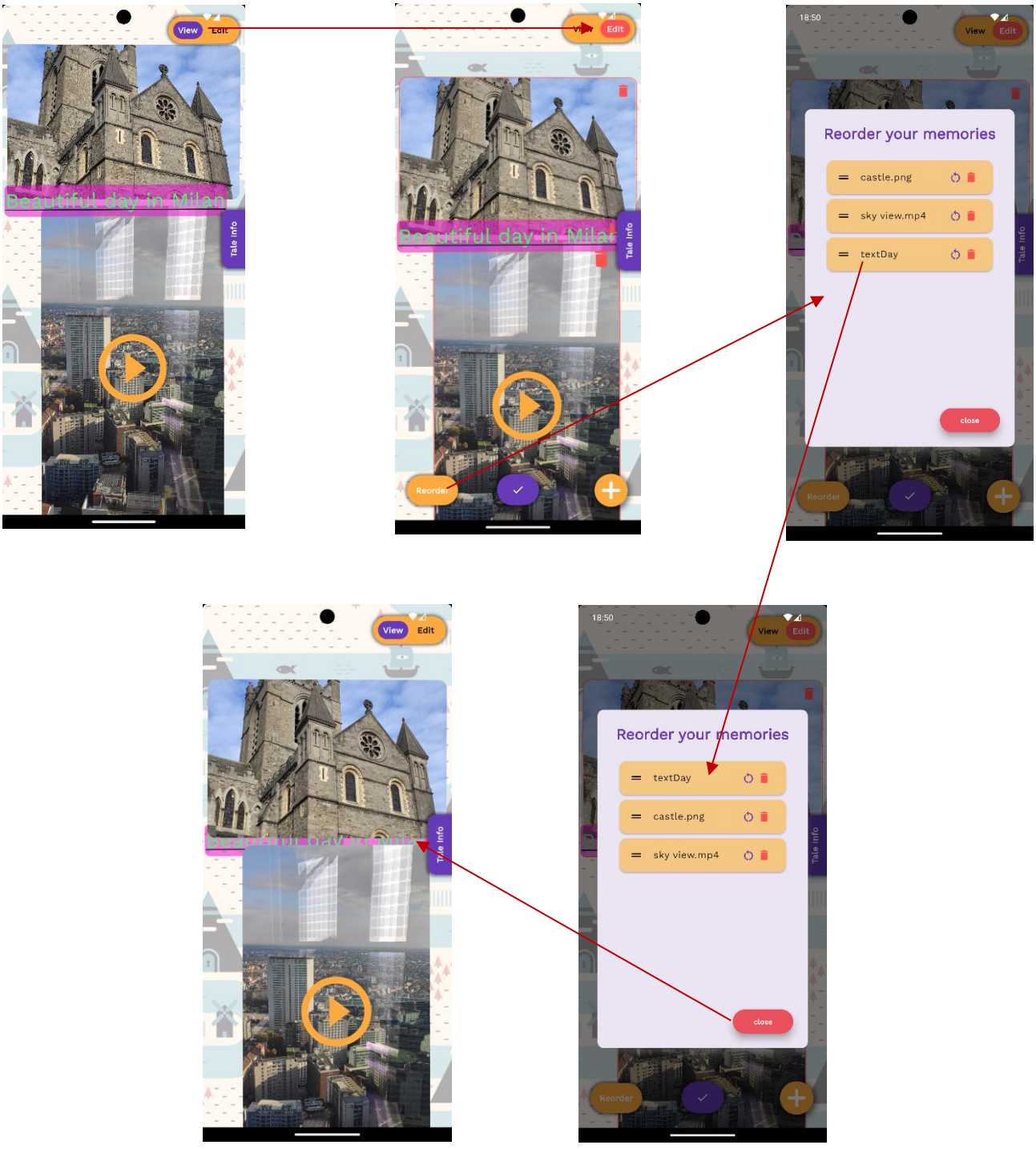
4.1.3 Create new Tale



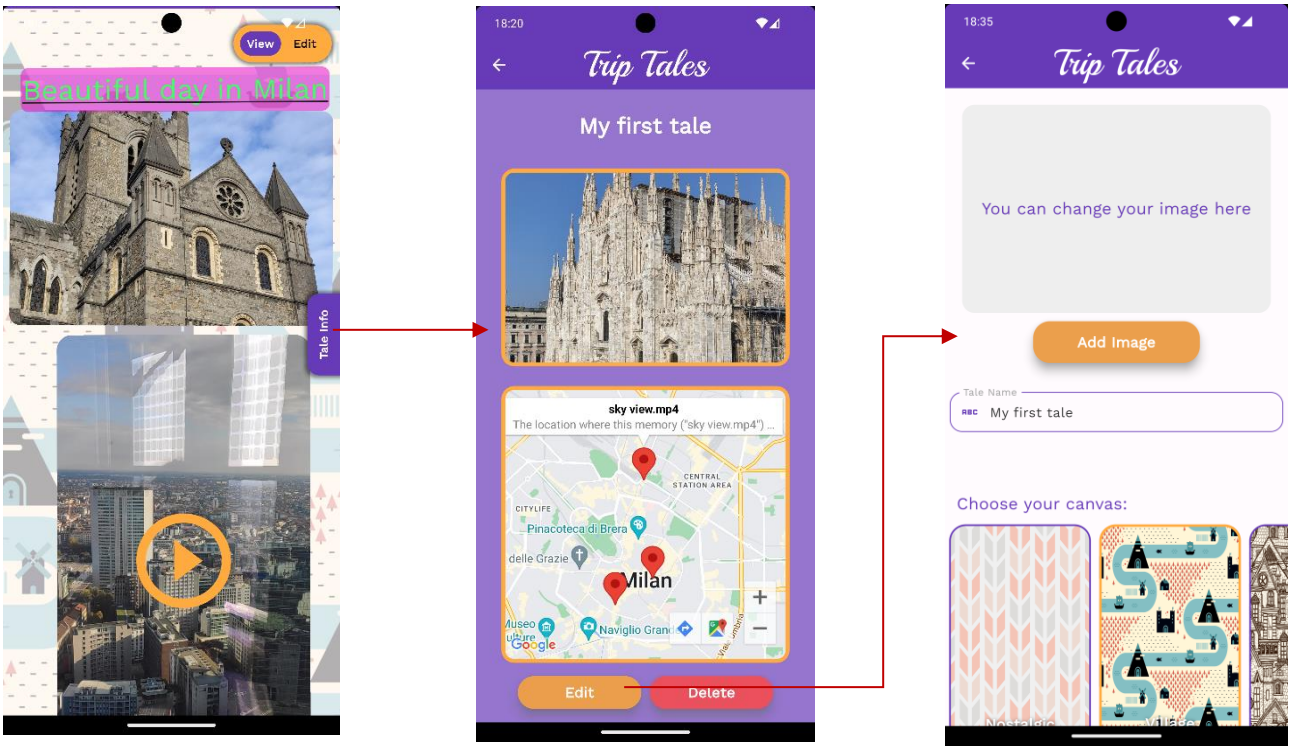
4.1.4 Modify new Tale



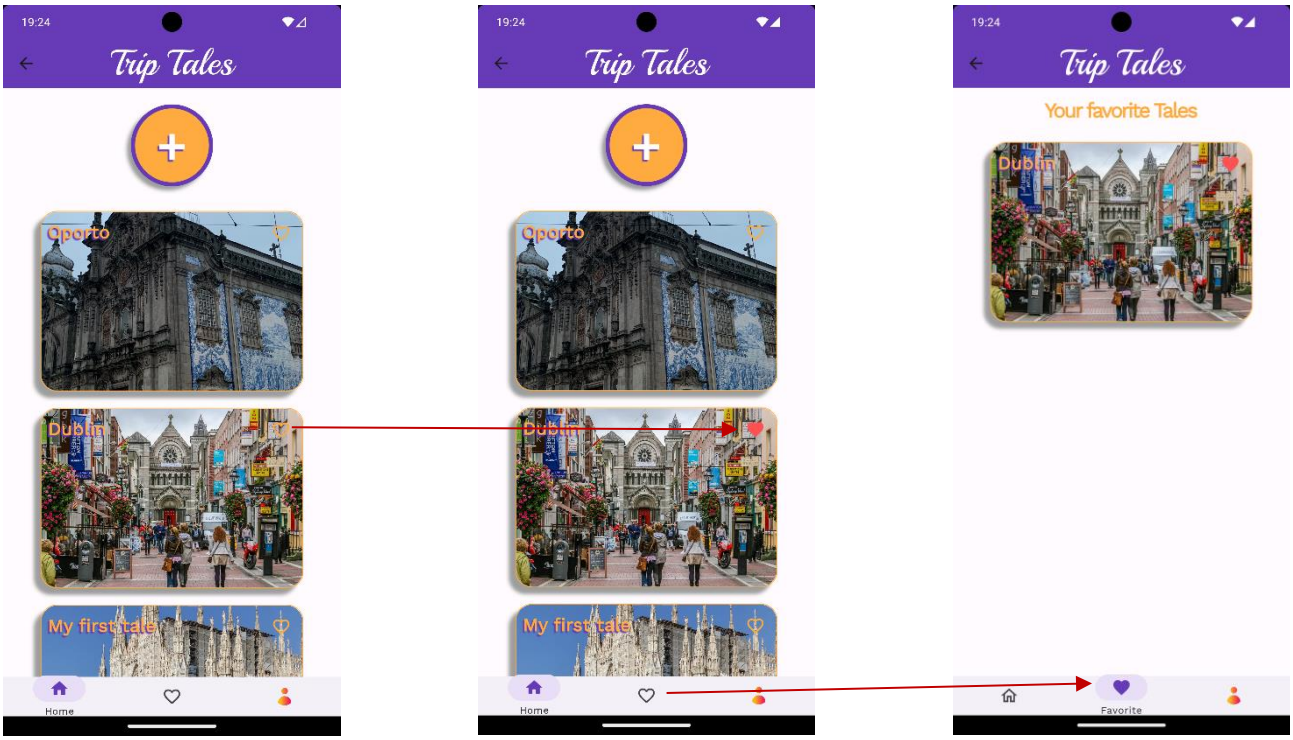
4.1.5 Reorder cards



4.1.6 Tale info



4.1.7 Tales created and Favourite Tales

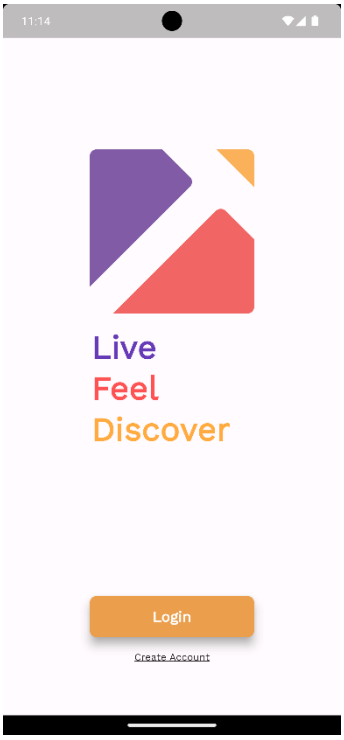


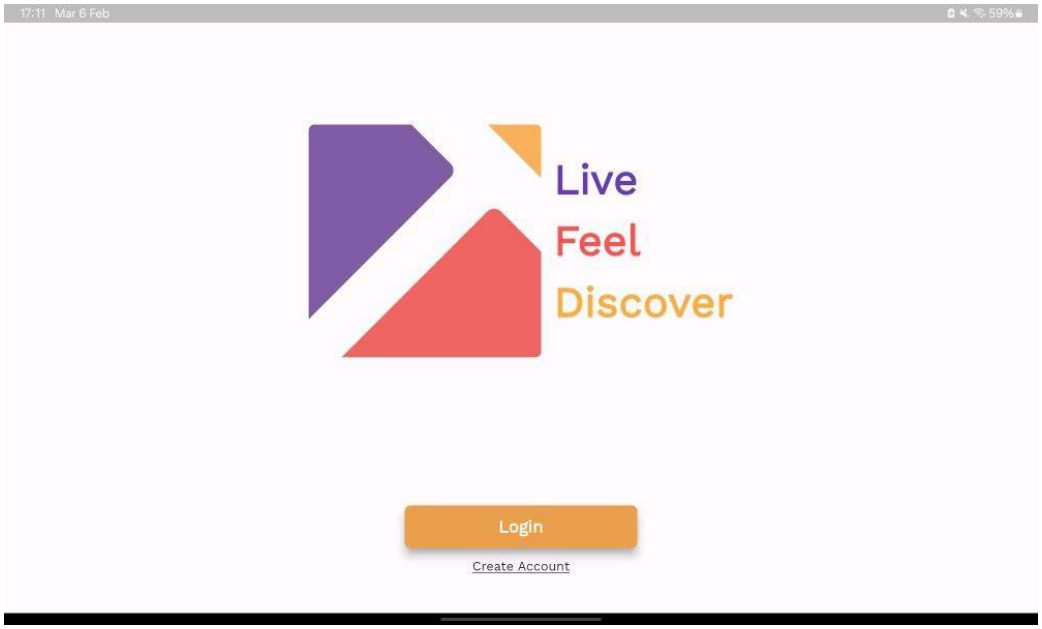
4.2 User Interfaces Design

In this section we will show the various screens of the application. Since we significantly changed the layout for the tablet version, we will put both the smartphone and tablet layouts.

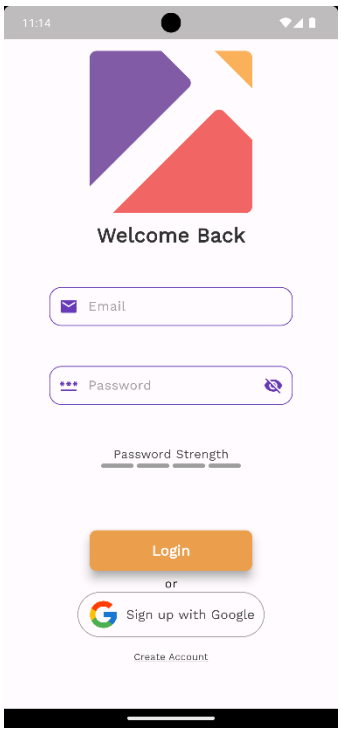
We decided to use a portrait screen orientation only for the smartphone devices since we think it's the most suitable way in order to fully enjoy the app, while for the tablet mode we opted for both landscape and portrait device orientation.

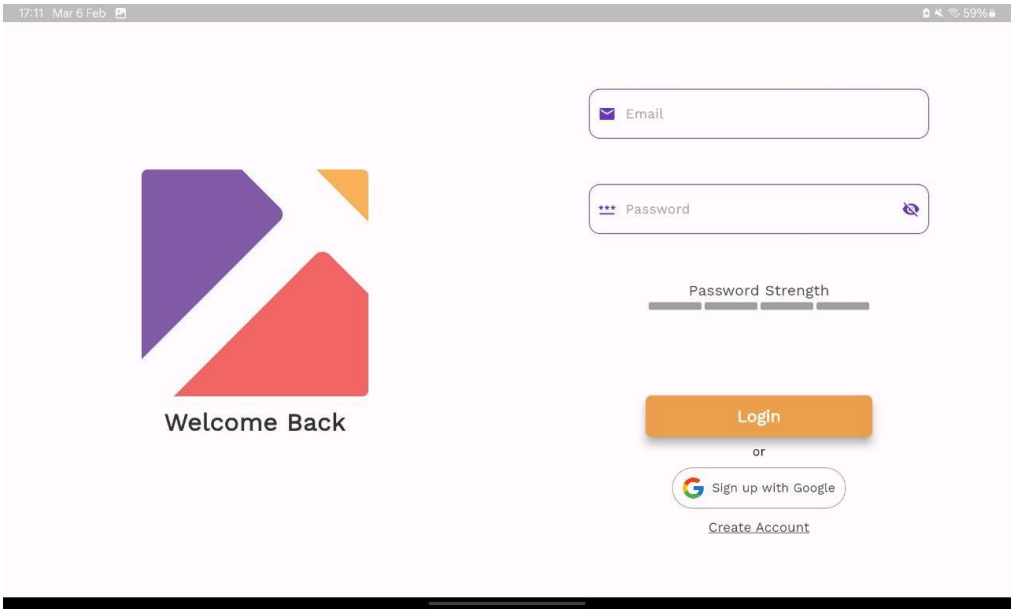
4.2.1 Home Page



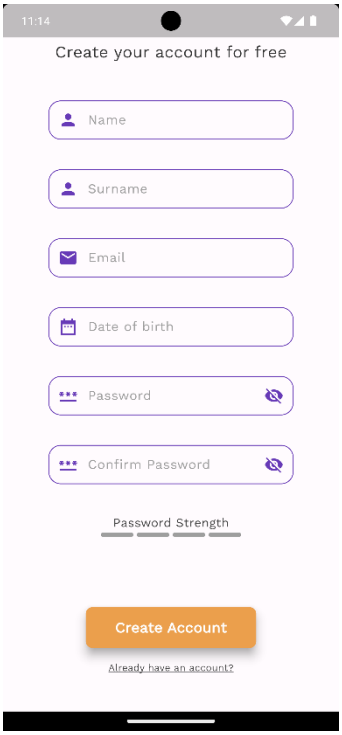


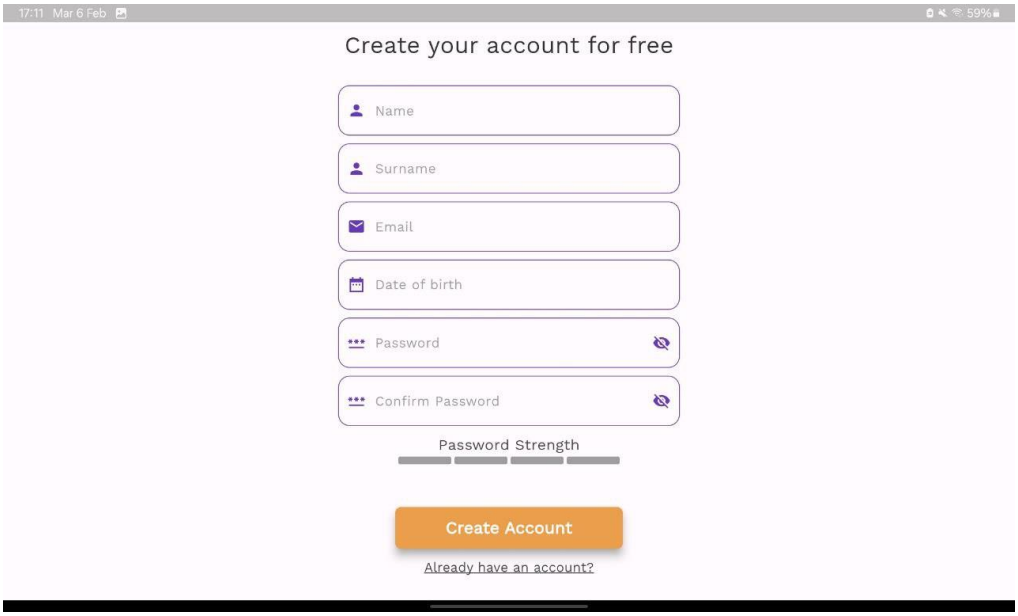
4.2.2 Login



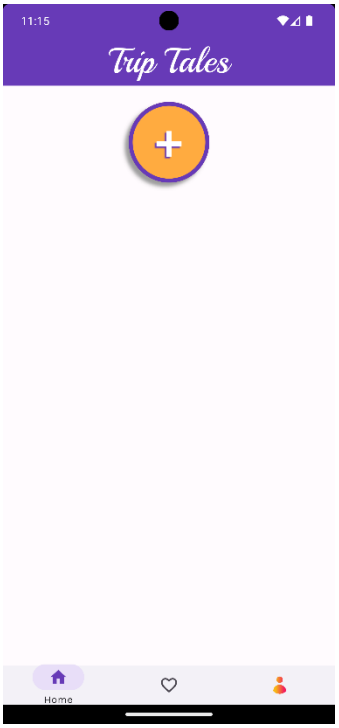


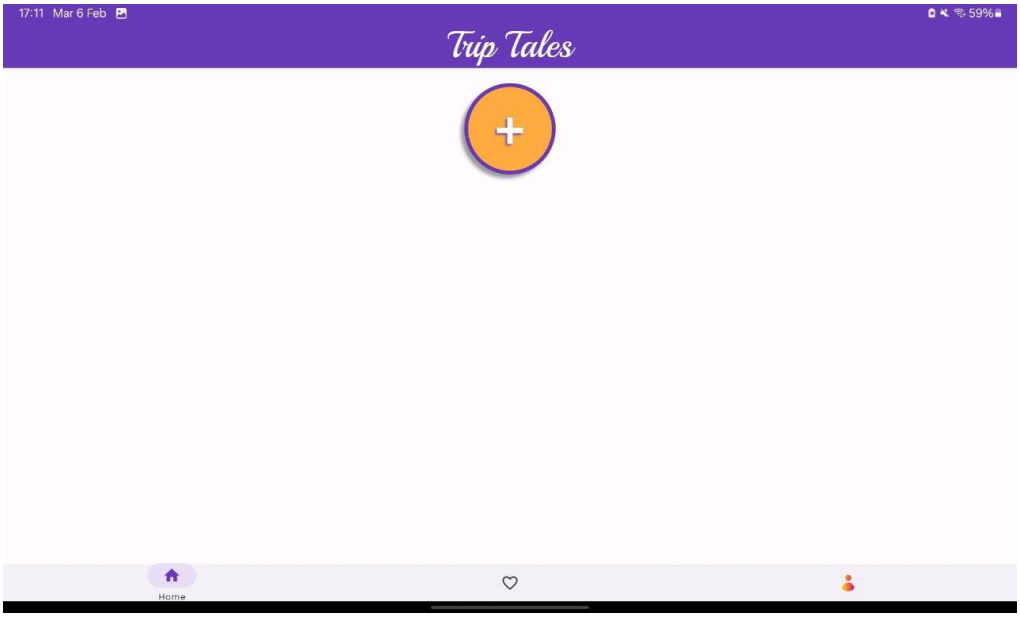
4.2.3 Registration phase



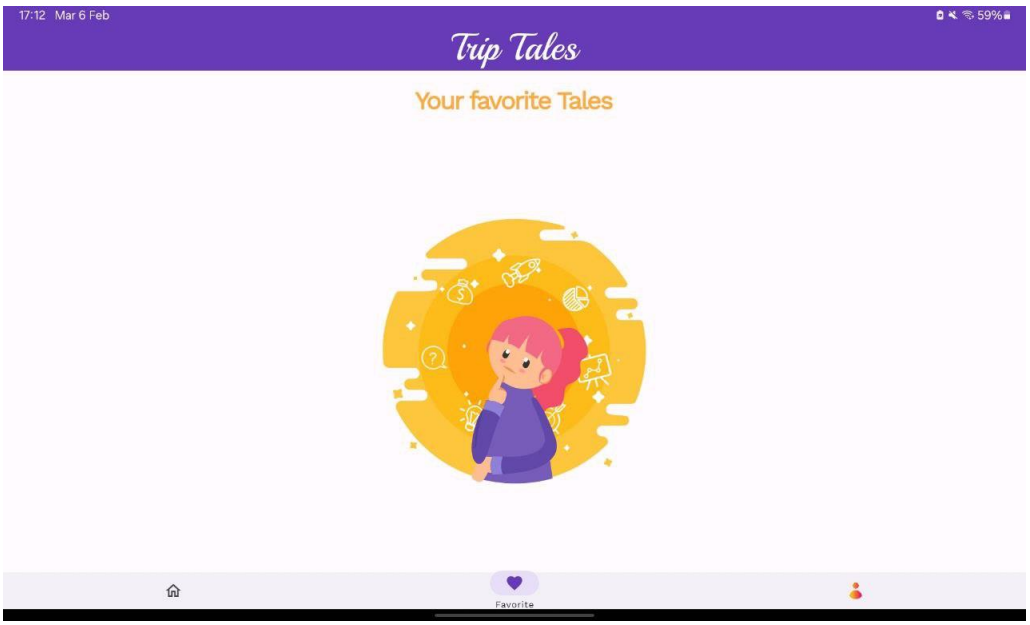


4.2.4 Dashboard for new user

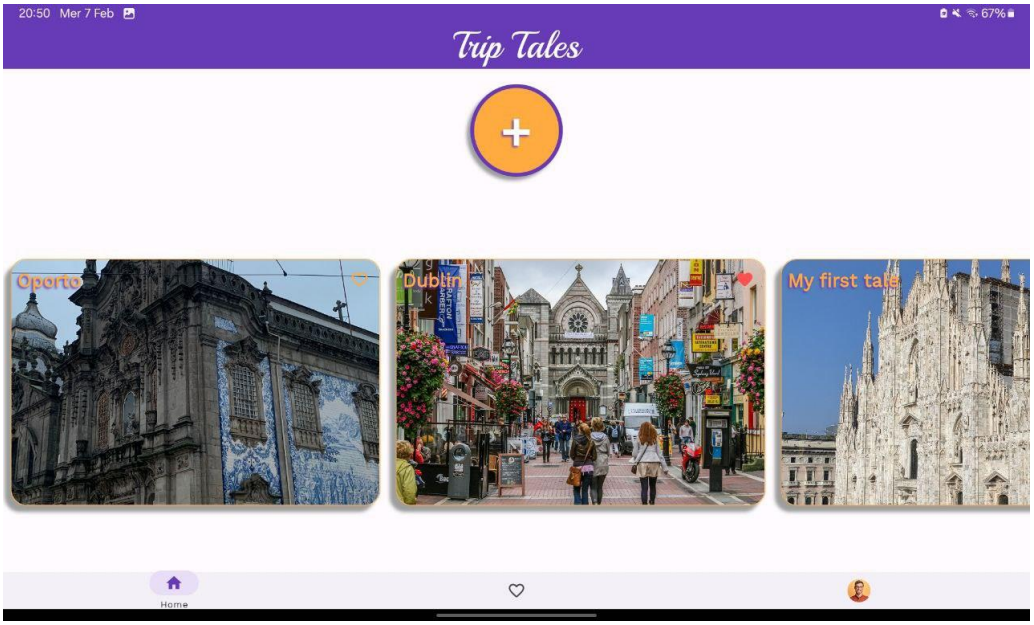




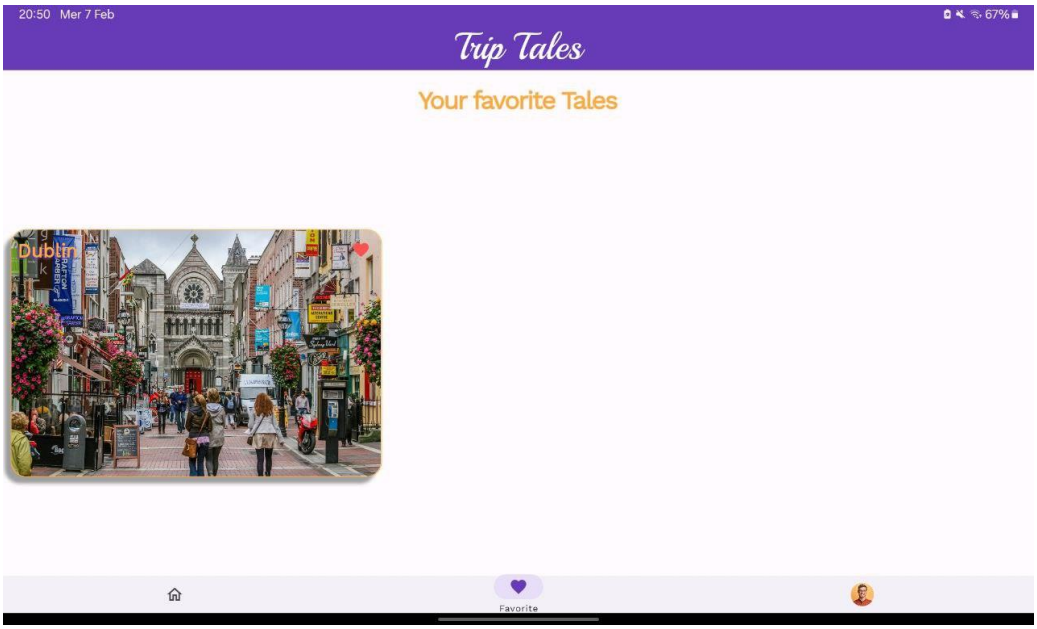
4.2.5 Favourite Tale for new user



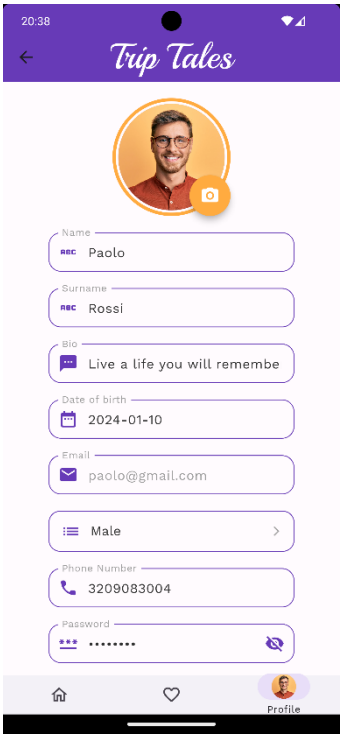
4.2.6 Dashboard for user already registered

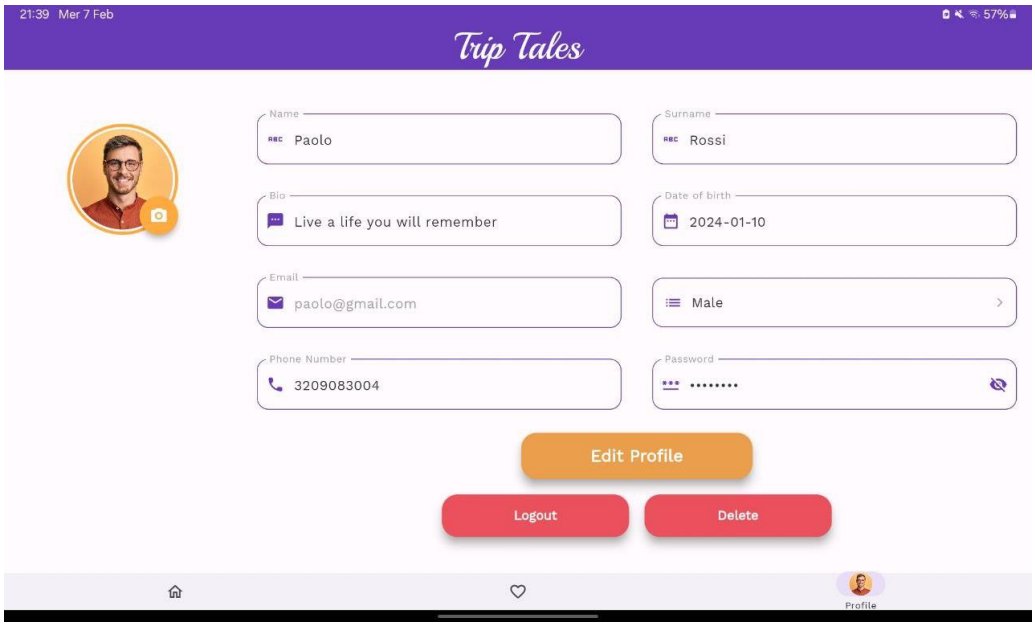


4.2.7 Favourite Tales for user already registered

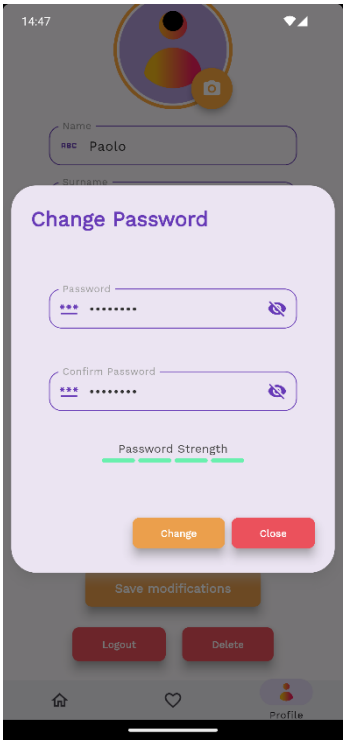


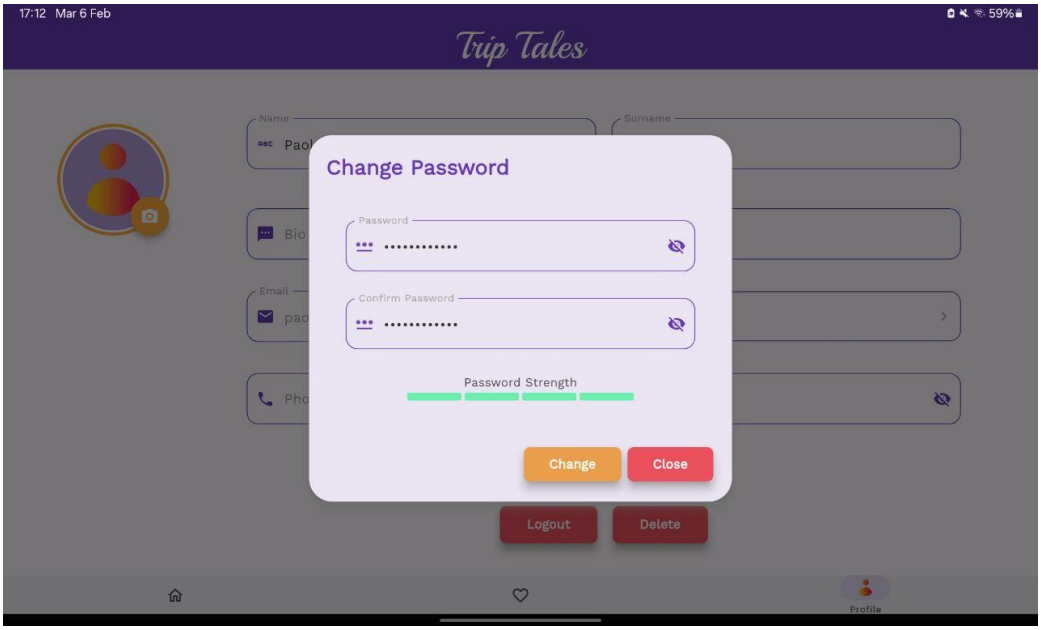
4.2.8 Profile Page



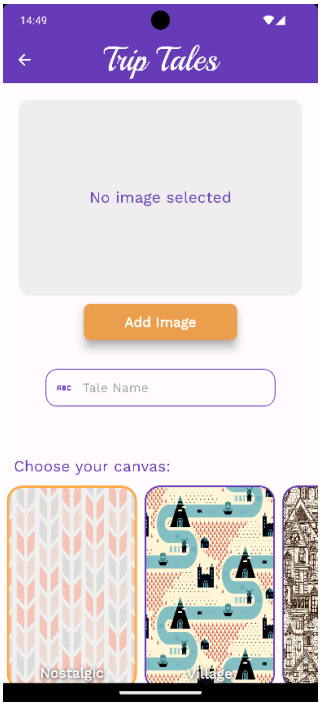


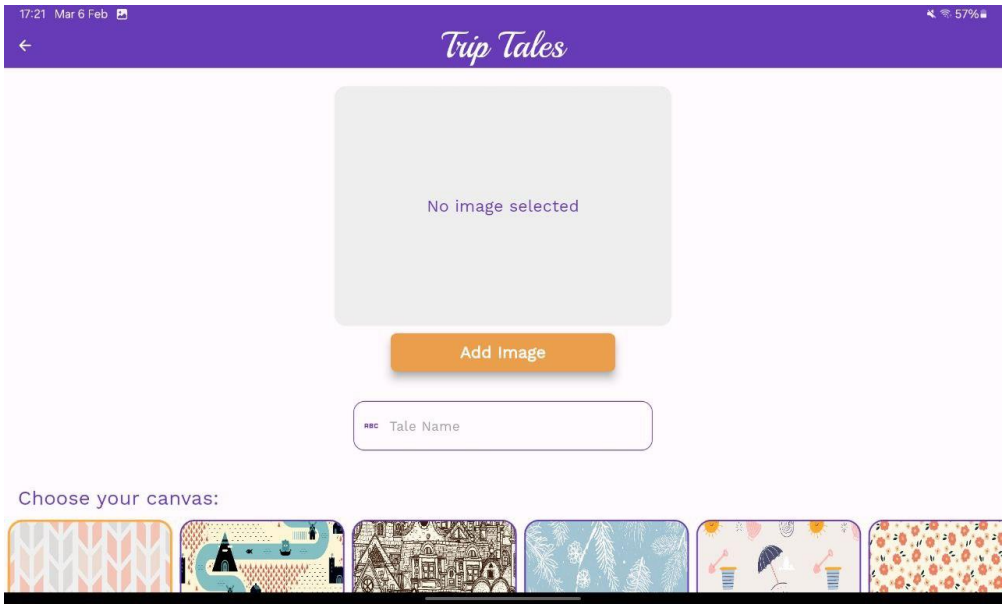
4.2.9 Change password

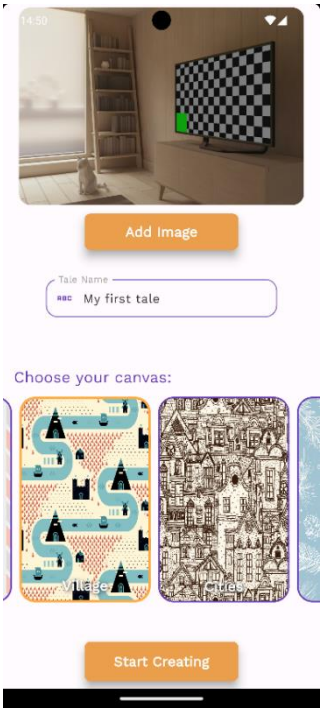


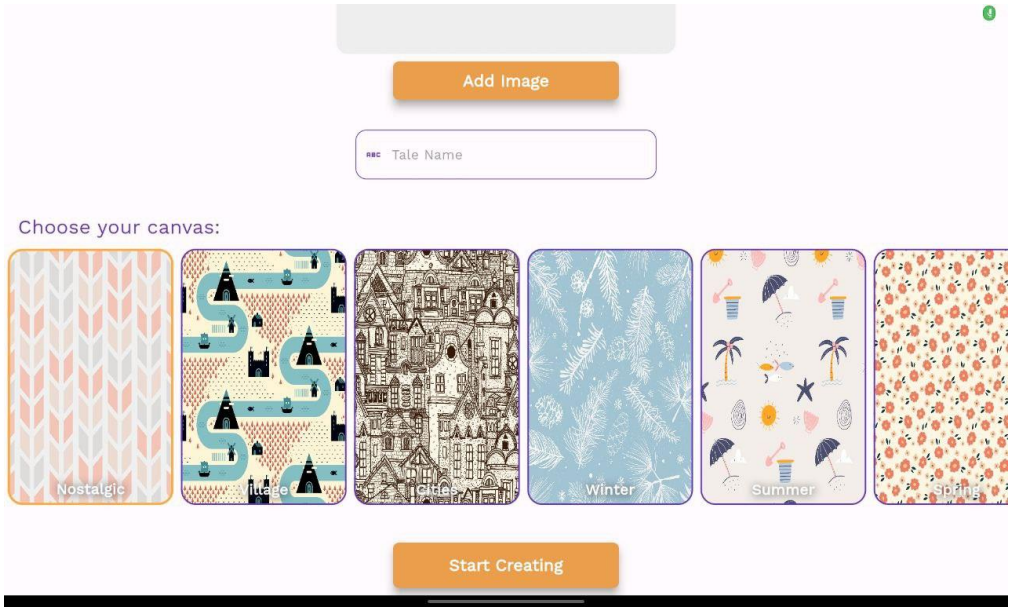


4.2.10 Create Tale Page





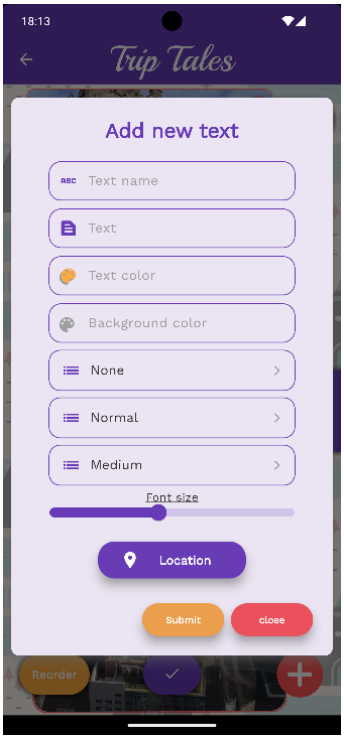


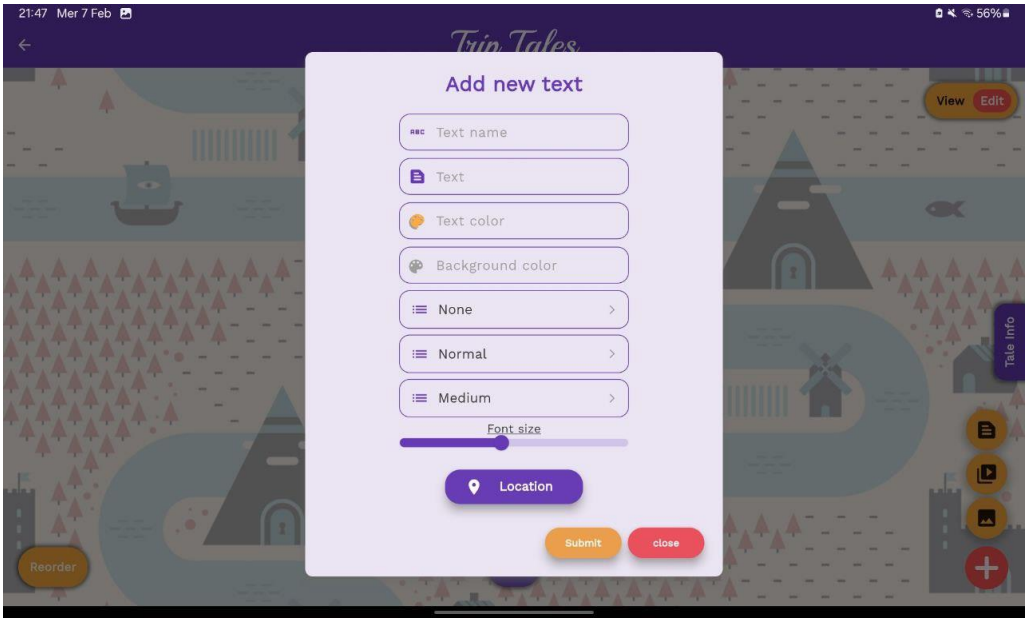


4.2.11 New Tale Page

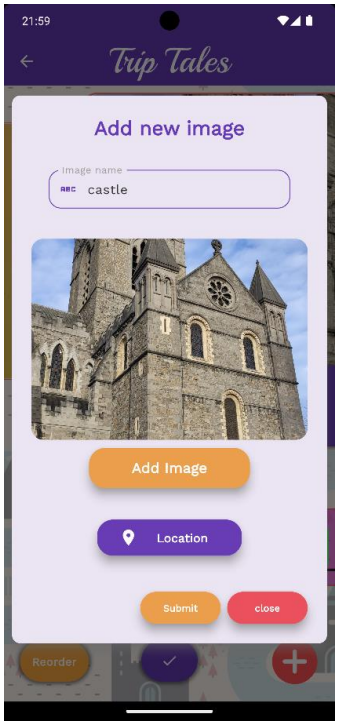
4.2.12 Edit Tale Page

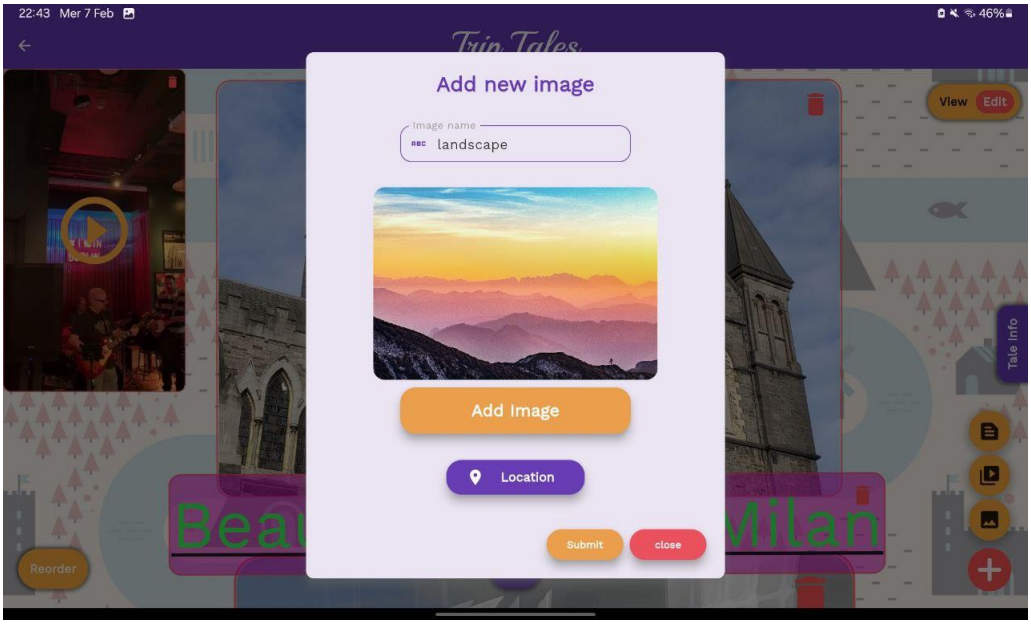
4.2.13 Add Text in Tale Page



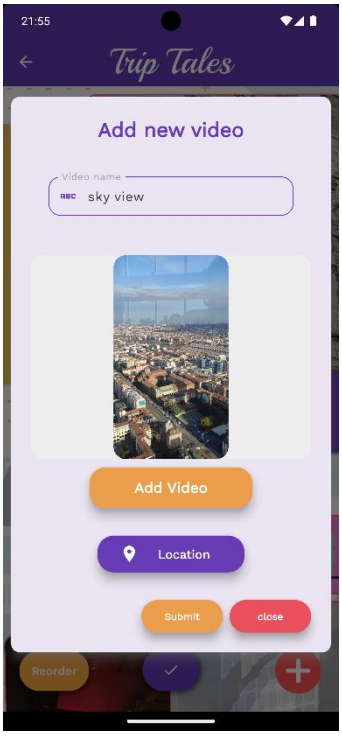


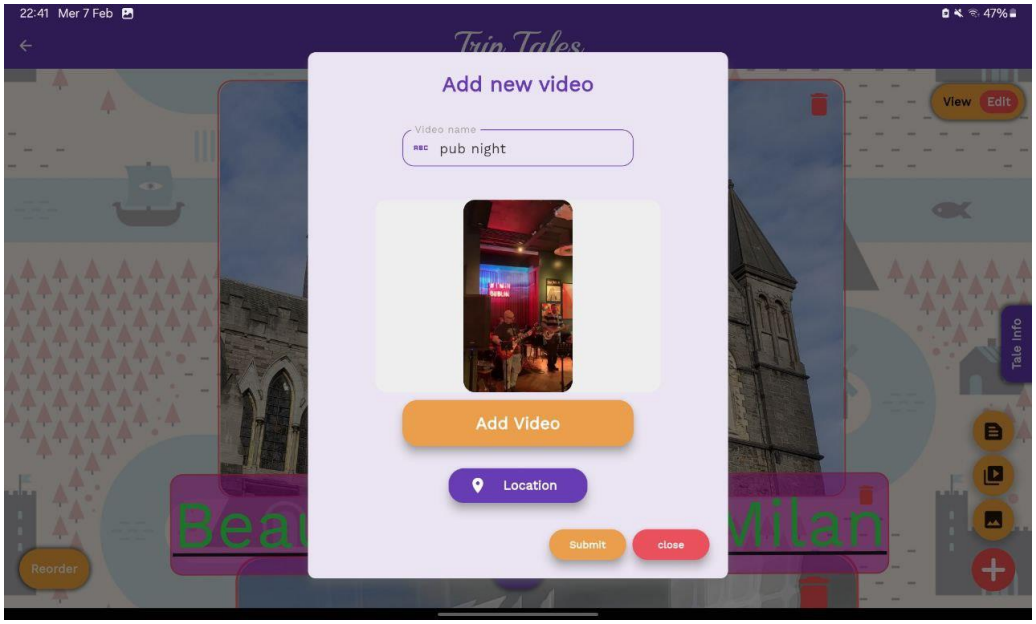
4.2.14 Add Image in Tale Page



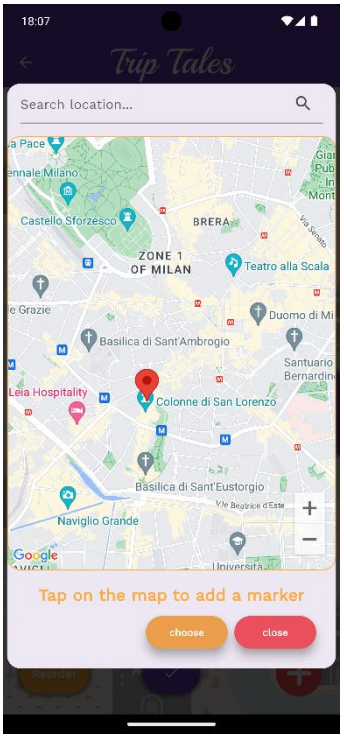


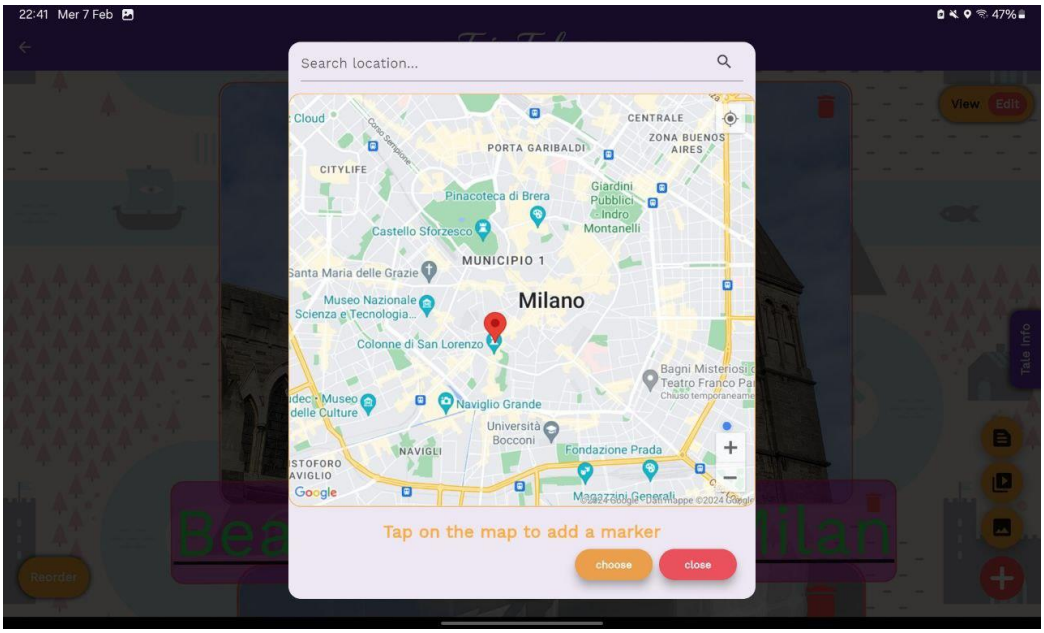
4.2.15 Add Video in Tale Page



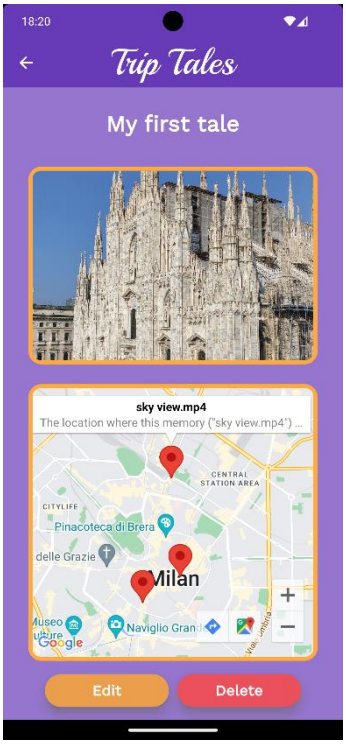


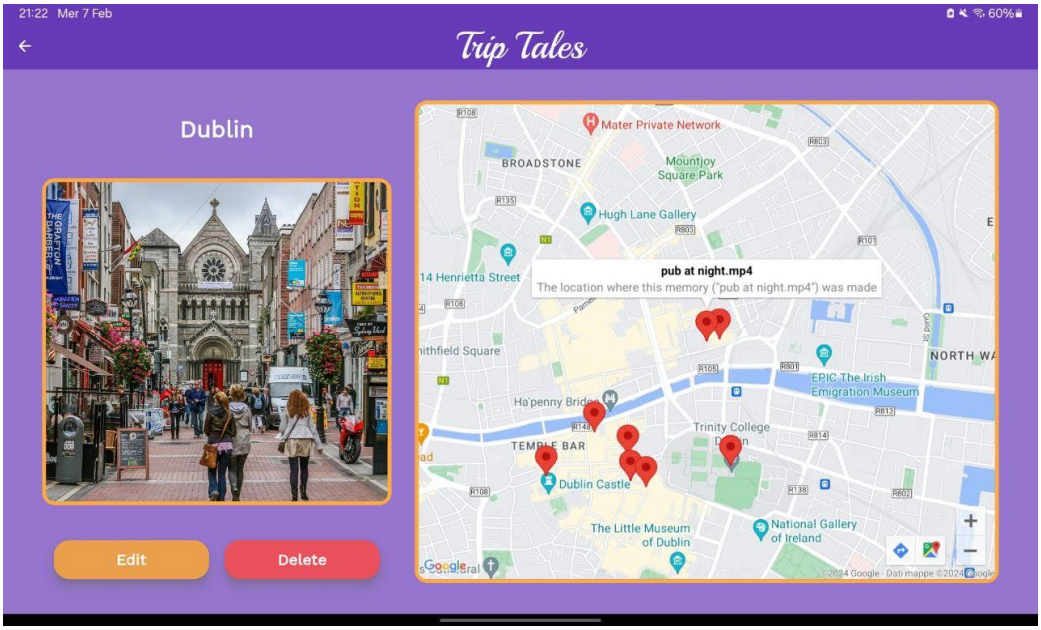
4.2.16 Add location to card



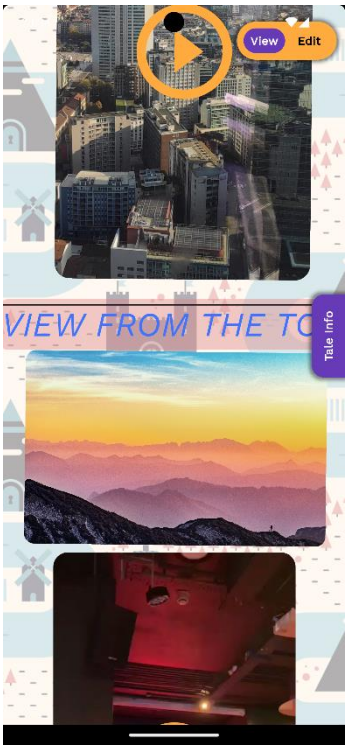


4.2.17 Tale info





4.2.18 Complete Tale View mode



4.2.19 Complete Tale edit mode

5.Testing

During the testing phase, our primary emphasis was on widget and unit testing to ensure TripTales operates seamlessly. Our goal was to validate the intended flow of interactions within the widgets and confirm proper functionality in every view.

Starting with the Home Page, we performed extensive tests on both tablet and smartphone versions.

In the upcoming section, we'll showcase completed test images and briefly evaluate their objectives. Furthermore, we'll present a code snippet illustrating a test, shedding light on our testing approach.

Our strategy involved scrutinizing each widget within the views and assessing all possible interactions, guaranteeing that any test failures stemmed from specific and identifiable causes.

Ultimately, we executed a total of 261 tests. Below is shown an overview of all the test made:

5.1 Widget Testing

Throughout our testing phase, we meticulously verified the application pages across smartphone and tablet versions. Our core aim was to confirm the existence and correct operation of each component according to its intended function.

Our scrutiny extended beyond mere visual confirmation of components; we engaged in comprehensive testing of every possible interaction within each page. Our goal was to ensure users encounter a smooth, error-free experience by thoroughly examining and testing these interactions. Below, you'll find an example illustrating a widget test:

```
Run | Debug
testWidgets('LoginPage email', (WidgetTester tester) async {
  await tester.pumpWidget(MaterialApp(home: LoginPage()));

  final emailTextFieldFinder = find.byKey(const Key('emailCustomTextField'));
  expect(emailTextFieldFinder, findsOneWidget);

  // Enter text into the email form field
  await tester.enterText(emailTextFieldFinder, 'example@example.com');
  await tester.pump(); // Trigger a rebuild

  // Verify the text in the field
  expect(find.text('example@example.com'), findsOneWidget);

  // Clear the text field
  await tester.enterText(emailTextFieldFinder, '');
  await tester.pump(); // Trigger a rebuild

  // Trigger validation
  await tester.pumpAndSettle();

  // Verify "Enter your email" error is displayed
  expect(find.text('Enter your email'), findsOneWidget);

  // Enter text into the email form field again
  await tester.enterText(emailTextFieldFinder, 'anotherexample@example.com');
  await tester.pump(); // Trigger a rebuild

  // Verify the new text in the field
  expect(find.text('anotherexample@example.com'), findsOneWidget);

  // Trigger validation
  await tester.pumpAndSettle();
});      You, 2 months ago • Tests and page layouts
```

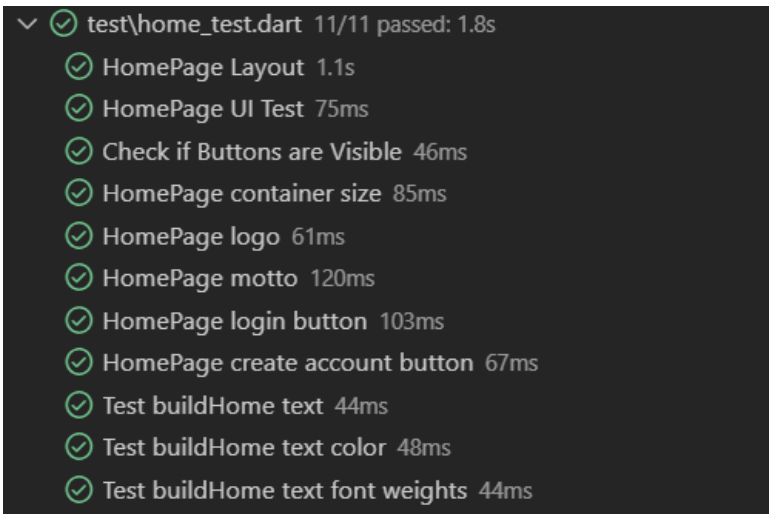
This widget test validates the functionality of the email field within the login page under two distinct scenarios: when it remains blank and when a correct email is provided. In the initial scenario, the test evaluates how the application manages an empty email field. It commences by configuring the widget tester and rendering the login interface. Subsequently, it identifies the email field using a unique key “emailCustomTextField” and confirms its presence on the interface. The tester inputs the text 'example@example.com' into the email field and promptly clears it. Following an interface

update trigger, the test anticipates the appearance of the error message 'Enter your email' on the interface, thus confirming the application's accurate detection and handling of an empty email input.

In the subsequent scenario, the test explores the application's behavior upon entry of a correct email. It reutilizes the previously established widget tester setup and re-renders the login screen. It locates the email field, inputs 'anotherexample@example.com' into it, and initiates an interface update. In this case, the expectation is that the error message 'Enter your email" will not be present on the interface anymore, signifying the successful recognition of a valid email input by the application.

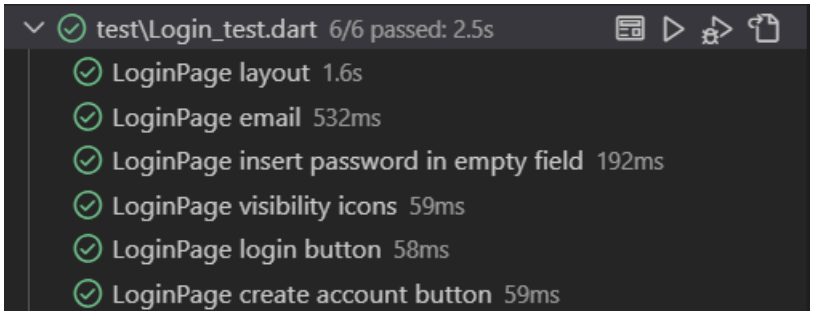
In the following pages there will be showed some examples of widget test (not all of them).

5.1.1 Home Page



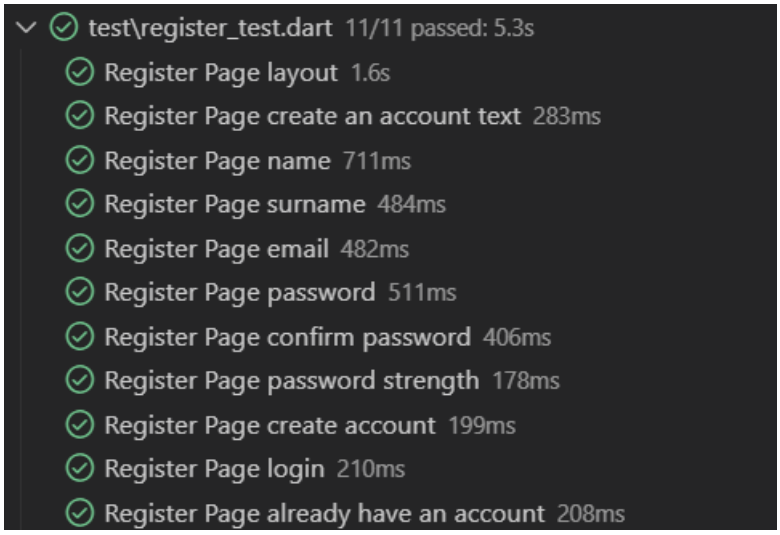
Test to check the overall layout of the page, all the buttons' visibility and functionalities and all the text colours and weights.

5.1.2 Login Page



Test that check the layout, the email, the password, the visibility of the password, the button function and the create account button function.

5.1.3 Register Page

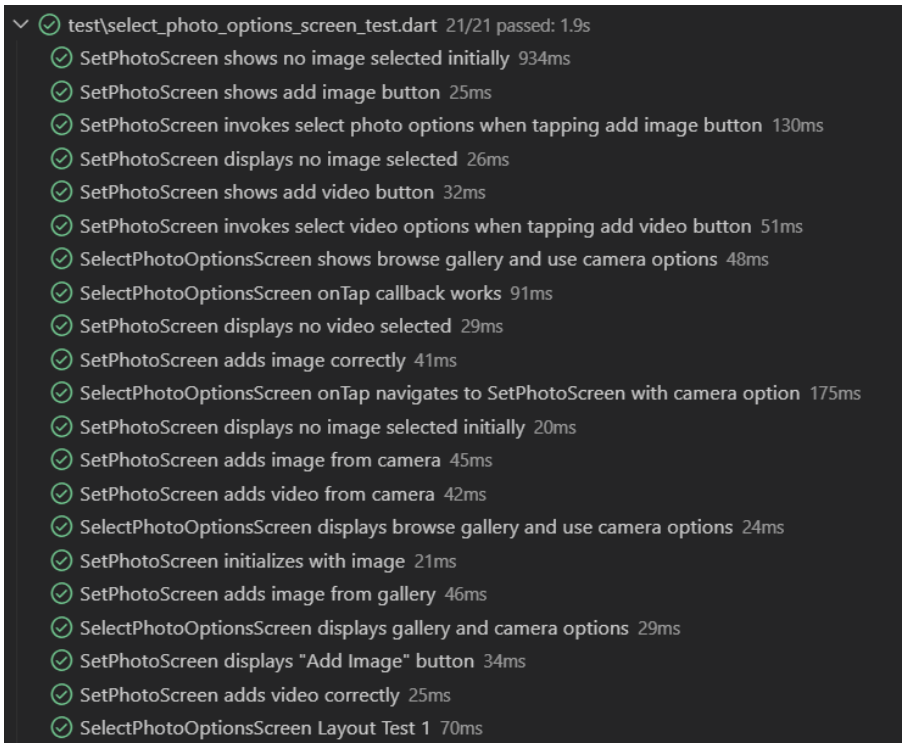


Tests all the layout of the register page, checking all the text fields and if the user has already an account.

5.1.4 Profile Page

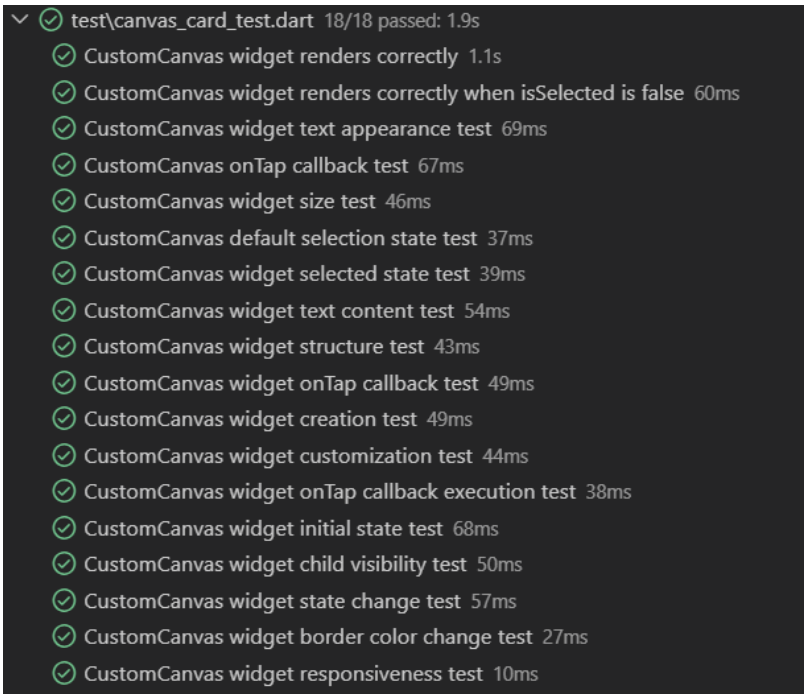
Tests for checking the layout of the profile page and all the text fields of the user's personal details.

5.1.5 Select Photo Options



Test for checking all the possible cases in using the set photo screen, it's rendering and the overall correctness of the widget.

5.1.6 Canvas Card



Tests for checking the rendering of the widget, the size, all its states and all the possible interaction with it.

5.2 Unit tests

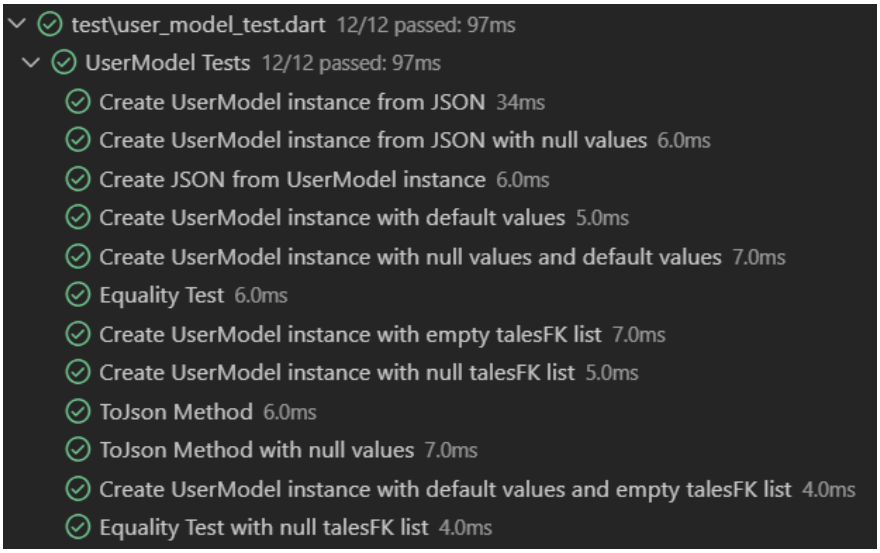
Alongside widget tests, we've incorporated unit tests to assess particular functionalities within the application. Unit tests enable us to concentrate on isolated units or components of the application, ensuring their accuracy and dependability. Below is an illustration of a unit test:

```dart
test('Create UserModel instance from JSON', () {
  final json = {
    'id': '123',
    'email': 'test@example.com',
    'name': 'John',
    'surname': 'Doe',
    'birthDate': '1990-01-01',
    'phoneNumber': '123456789',
    'bio': 'Lorem ipsum',
    'gender': 'Male',
    'profileImage': 'profile.jpg',
    'talesFK': ['1', '2'],
  };

  final imageURL = 'https://example.com/image.jpg';
  final user = UserModel.fromJson(json, imageURL);

  expect(user.id, '123');
  expect(user.email, 'test@example.com');
  expect(user.name, 'John');
  expect(user.surname, 'Doe');
  expect(user.birthDate, '1990-01-01');
  expect(user.phoneNumber, '123456789');
  expect(user.bio, 'Lorem ipsum');
  expect(user.gender, 'Male');
  expect(user.profileImage, 'https://example.com/image.jpg');
  expect(user.talesFK, ['1', '2']);
});
```

The intent behind these unit tests was to pinpoint particular functions or modules within the app, analyzing their behaviors and anticipated results. By isolating these units and conducting independent tests, we can promptly detect and address any potential issues or bugs in the code.

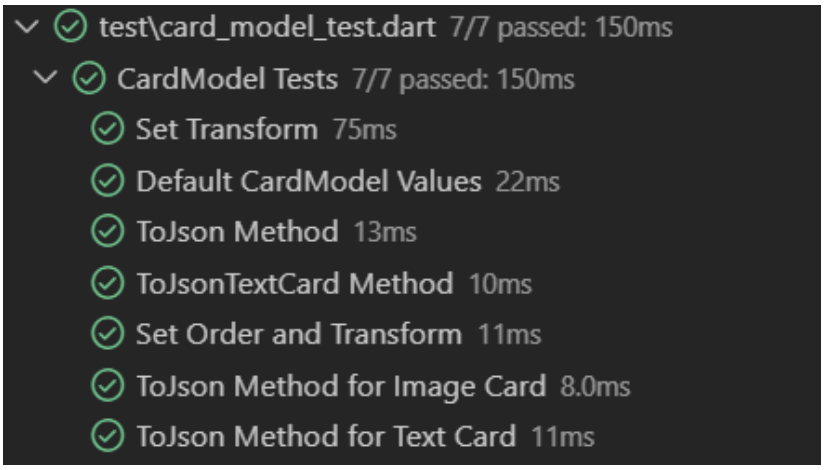
In the following pages it will be shown some of the unit tests we have conducted (not all of them).

5.2.1 User Model Unit test



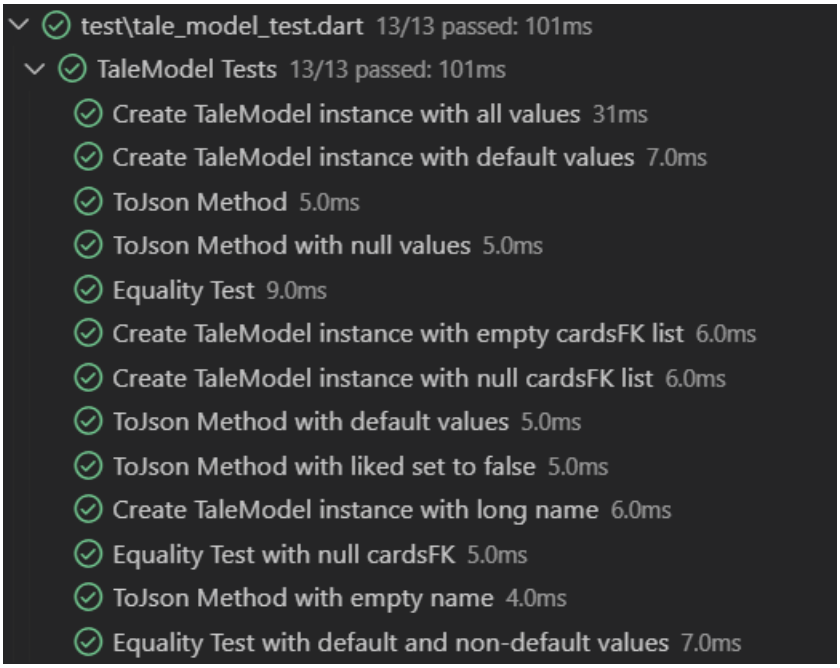
The purpose of this unit test is to validate the correctness and robustness of the User model by assessing its individual components and functions in isolation.

5.2.2 Card model unit test



The purpose of this unit test is to validate the correctness and robustness of the Card model by assessing its individual components and functions in isolation.

5.2.3 Tale model unit test



The purpose of this unit test is to validate the correctness and robustness of the Tale model by assessing its individual components and functions in isolation.