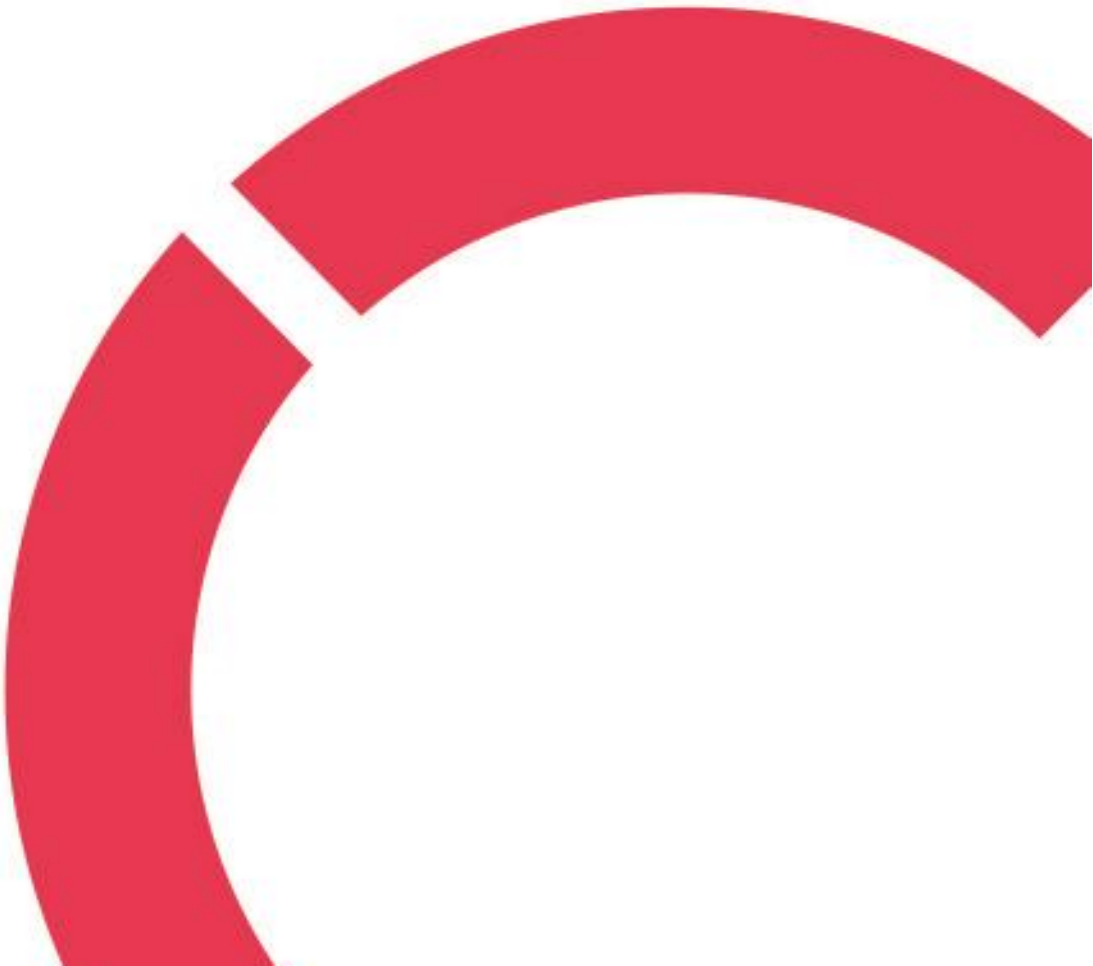
Luan Nguyen and Minh Hoang

BUILDING A FULLSTACK MOBILE APPLICATION WITH FLUTTER AND STABLE DIFFUSION MODEL

Thesis
CENTRIA UNIVERSITY OF APPLIED SCIENCES
Information Technology
May 2024



ABSTRACT

Centria University of Applied Sciences	Date	Author
	May 2024	Luan Nguyen & Minh Hoang

Degree programme
Information Technology

Name of thesis
BUILDING A FULLSTACK MOBILE APPLICATION WITH FLUTTER AND STABLE DIFFUSION MODEL

Centria supervisor	Pages
Henry Paananen	56+3

The thesis' core focus is on the development and implementation of a mobile application using Flutter and the Stable Diffusion model. TikTok has grown as a dominating social network in today's digital world, particularly among the younger generation, overtaking even giants like Facebook and Instagram. Creators often create short, entertaining video content immediately from their mobile devices, which contributes to the platform's appeal. Furthermore, artificial intelligence has grown in popularity, particularly since the release of ChatGPT near the end of 2022. The thesis finds its niche, trying to investigate the concept of directly generating images using artificial intelligence on mobile devices.

The thesis is divided into three parts. The first section presents a thorough technology overview and explains fundamental artificial intelligence models, establishing the groundwork for readers. The second section focuses on the planning and development of the mobile application, which includes fundamental functionality for viewing and generating Japanese drawing-style graphics from real-life photos. It provides useful information about the technical aspects of front-end and back-end implementations. The thesis finishes with a thorough overview that includes recommendations for improving mobile applications.

In short, this thesis is a helpful resource for people interested not just in Flutter and the Dart programming language, but also in artificial intelligence models in the context of image generation, specifically the Stable Diffusion model. By combining these features, the thesis demonstrates how to develop and implement a full-stack solution for a mobile application that is meant to function flawlessly on Android smartphones.

Key words
Android development, Artificial Intelligent, Dart, Flutter, Image Generating, Stable Diffusion

CONCEPT DEFINITIONS

AI	Artificial Intelligence
API	Application Programming Interface
AWS	Amazon Web Services
CI/CD	Continuous Integration and Continuous Deployment
CDN	Content delivery network
CLI	Command line interface
EDA	Event-driven architecture
NLP	Natural language processing
GANs	Generative adversarial networks
GPU	Graphics processing unit
HTTP	Hypertext Transfer Protocol
IDE	Integrated development environment
PNG	Portable Network Graphics
REST	Representational state transfer
S3	Simple Storage Service
SQS	Simple Queue Service

ABSTRACT
CONCEPT DEFINITIONS
CONTENTS

FIGURES

TABLES

PICTURES

1 INTRODUCTION

In today's digital landscape, TikTok has undeniably risen as a dominant social network thus surpassing even the giants of the industry such as Facebook and Instagram (Kaur 2021). This rapid take-off can be attributed to the fact that creators have embraced TikTok as a platform of choice, consistently creating short and viral video content directly from their mobile devices. Their contributions have played a pivotal role in supporting TikTok's popularity with the young generation. Furthermore, the emergence of artificial intelligence got a lot of notice from the public. The launch of ChatGPT in late 2022 has sparked significant interest and curiosity in the possibilities of AI (Yosifova 2023). This breakthrough has created a real sense of excitement and fascination, with the broader tech community keen to investigate the possibilities that AI might provide.

In the field of cross-platform development, Flutter stands out as one of the technologies that is expanding the fastest. It is an open-source framework by Google for building beautiful, natively compiled, multi-platform applications from a single codebase. The Dart programming language, which has many advantages and possibilities, is the main component that fuels the magic of Flutter application creation. Dart is a powerful language with a focus on the best client-side performance, making it a good option for building applications for both web and mobile platforms. Additionally, its versatility covers desktop and embedded devices as well, giving developers a wide range of potential applications (Hajian 2021). Based on Wappalyzer.com, there are more 2900 websites which are developed using Flutter.

This thesis is in its place within this exciting setting. Its primary goal is to investigate a question: how to generate images using artificial intelligence that are all easily available right on mobile devices. This forward-thinking strategy promises creative answers to current problems and is fully in line with the technological trends. The core functionality of the application is enabling users to both view and generate Japanese drawing-style images, anime style, from real-life photos. The front-end of the application is developed using Flutter and Dart, while the back-end is constructed on Amazon Web services using Serverless Framework, API Gateway, Lambda, Simple Queue Service (SQS), Simple Storage Services (S3), DynamoDB, and CloudFront.

2 THE ECONOMIC POTENTIAL OF GENERATIVE AI

ChatGPT, GitHub Copilot, Stable Diffusion, and Google Gemini are examples of generative AI systems that can execute routine tasks like data reorganization and classification. However, it is their capacity to write prose, produce music, and produce digital artwork that has gained attention and encouraged consumers and households to try things on their own that has garnered headlines. As a result, generative AI has the potential to redefine roles and increase performance in areas such as marketing and sales, customer operations, and software development. In the process, it has the potential to release trillions of US dollars in value among industries ranging from finance to life sciences. (Chui, Hazan, Roberts, Singla, Smaje, Sukharevsky, Yee & Zemmel 2023.)

In 2023, McKinsey report believes that generative AI may add an amount of $2.6 trillion to $4.4 trillion yearly to the 63 application cases. Software engineering, sales and marketing, customer operations, and research and development contribute to over 75% of the value that generative AI use cases potentially generate. Generative AI may enhance customer interactions, generate creative material for marketing and sales, as well as produce programming code based on natural-language prompts. Banking, high technology, and life sciences are just some of the areas that could experience the greatest impact from generative AI with regard to earnings. If the use cases were completely executed, information technology might provide an extra $200 billion to $340 billion in every year value. The potential impact on retail and consumer packaged products is similarly enormous, ranging from $400 billion to $660 billion annually. (Chui et al. 2023.)

Generative AI can significantly enhance human capabilities by automating a portion of daily tasks. Current generative AI technologies have the potential to automate up to 70% of the time spent on work activities. This advancement surpasses previous estimates by researchers, which projected that technology could automate half of employee time. The growing ability of generative AI to comprehend natural language, a key skill required for labor tasks that occupy 25% of total work time, is primarily responsible for this increased automation potential. As a result, generative AI is likely to have a more impact on work associated with high-income and highly educated professions than other types of employment. By 2040, generative AI could contribute to annual labor productivity gains of 0.1 to 0.6 percent, depending on the rate of technology adoption and how free time of the workers is reallocated. The combined impact of work automation with other technological improvements could potentially boost annual productivity gains by 0.2 to 3.3 percentage points. However, to fully realize

the benefits of generative AI, workers will need support in acquiring new skills, and some may need to shift to new occupations. If these labor transitions and other potential risks are managed effectively, generative AI has the potential to significantly contribute to economic growth and foster a more sustainable and equitable future. (Chui et al. 2023.)

The thesis provides developers and innovators with reference material on how to create mobile applications that leverage generative models such as Stable Diffusion, particularly in light of the recently opened frontier in generative AI. With the help of this widely utilized technology, unique material that customizes user experiences can be produced. The thesis provides a comprehensive overview of creating mobile applications that use generative models, AWS Web Services, and Flutter to accomplish certain objectives. It draws attention to how these technologies can change businesses by allowing them to find new ways to optimize resources and approach client interaction and product creation. As businesses realize how disruptive generative AI may be, this thesis offers as an example for how to open doors and influence the next phase of innovation in commercial software development.

3 THEORY BACKGROUNDS

In this area, the proposal presents and talks about the key innovations utilized within the creation of the application. The front conclusion of the application is built utilizing the Flutter framework. The application of artificial intelligence models and concepts is additionally secured in this segment, with a center on text-to-image change. Additionally, Automatic Web UI 1111 is a powerful device for text-to-image generation for AI models with Stable Diffusion. The final part of this section discusses the Serverless Framework and other Amazon web services that were used in the back-end development of the extension. Among these administrations are the API Gateway, Lambda functions, S3 storage, CloudFront, DynamoDB, and Single Queue Services.

3.1 Flutter

Flutter is a Google open-source user interface development kit (Flutter Developers 2023). It allows developers to create apps for several platforms from one source code base. The concept of creating user interfaces with widgets is at the heart of the framework's unique approach. It is a core design concept inspired by React's components, a well-known library for web-based applications. This widget-based framework enables developers to construct interactive and dynamic interfaces, fostering an application structure that is highly flexible and customizable. (Sanity Developers 2023.) FIGURE 1 demonstrates the trend of interest in Flutter by the time when compared with its competitor, React Native, while TABLE 1 compares their features. According to FIGURE 1, since 2020, Flutter started surpassing React Native in searching on Google in the worldwide.

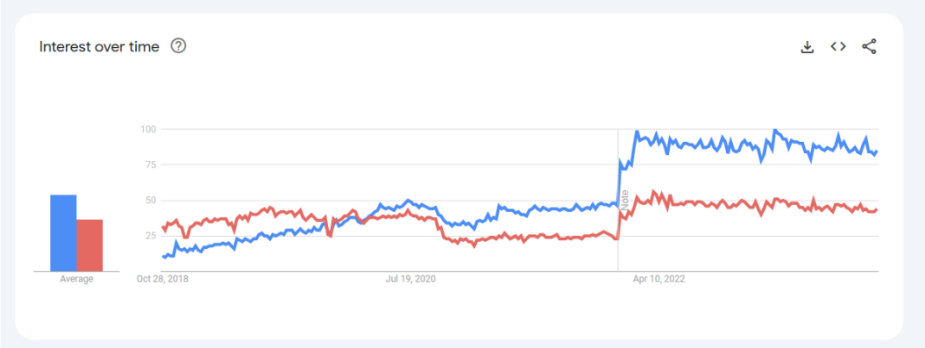


FIGURE 1. The comparison on searching for "flutter" and "react native" on Google trend from October 2018 up to October 2023 in worldwide

TABLE 1. Comparison between Flutter and React Native (Nguyen 2023)

Factor	**Flutter**	**React Native**
Programming language	Dart	JavaScript
Development environment	Android Studio, IntelliJ IDEA, or Visual Studio Code with Flutter plugins	Any IDE or editors
Fundamental building block unit	Widget	Component
Performance	Faster due to compiled code and own rendering engine	Slower due to JavaScript bridge and native UI components
Libraries and packages	Fewer but growing number of packages and plugins available, mostly from Google or Flutter team	More third-party libraries and integrations available thanks to JavaScript ecosystem
Platform Compatibility	iOS, Android, Web, Windows, Mac, and Linux (officially)	iOS, Android, and Web (officially), Windows and Mac (unofficially)
App Maintenance and Updates	Harder to maintain and update due to compiled code and app store policies	Easier to maintain and update due to dynamic code loading and over-the-air updates
Testing and debugging	Provides a rich set of tools for testing and debugging, such as DevTools, DartPad, Dart Dev Compiler, Flutter Inspector	Provides some tools for testing and debugging, such as Flipper, React Native Debugger, Reactotron
Learning Curve	Moderate to learn for web developers who need to learn Dart and Flutter widgets	Easy to learn for web developers who are familiar with JavaScript and React
Use Cases and Applications	Suitable for complex apps that need high performance or custom UI. Examples: Google Ads, Alibaba, Philips Hue, Reflectly	Suitable for simple apps that need to leverage native features or existing JavaScript libraries. Examples: Facebook, Instagram, Tesla, Skype

What genuinely separates Flutter from its competitors are the numerous advantages it provides. One of its key advantages is the ability to create natively compiled applications from one source code. This approach simplifies development while ensuring consistency across many platforms. (Sanity Developers 2023.) However, as compared to native application development, this strategy will meet the application's performance factor. Each platform has its own native development kit and languages, such as Android's Kotlin and iOS's Swift. Creating a solution for each platform will take additional time and resources. (Nguyen 2023.)

Flutter also excels at creating smooth and speedy animations, especially on older devices, boosting the user experience through engaging and visually appealing interactions (Sanity Developers 2023). One of the reasons for this is Dart. Dart is the programming language used by Flutter (Flutter Developers 2023). It is a client-oriented programming language for creating speedy apps on any platform. Its purpose is to provide the most productive programming language for multi-platform development, as well as a versatile execution runtime platform for application frameworks. It converts the code to native machine code, which improves performance and efficiency. (Dart Developer 2023.)

Furthermore, Futter platform channels also provide direct contact with native features and APIs without the need for an intermediary layer. In addition, Flutter includes its own rendering engine, mostly written in C++, which is fast and has an extensive range of functionality and customization capacities. It may also encounter challenges with compatibility or limits with some native features or integrations. Furthermore, because of its own rendering engine and dependencies, Flutter may have a bigger app size. (Nguyen 2023.)

For development, Flutter's "hot reload" feature is a game changer for developers, allowing them to observe the impact of code changes immediately without the need for long recompilation or restarts. This feature shortens the developing cycle, resulting in a more efficient and productive atmosphere. (Sanity Developers 2023.) For release, Flutter applications are compiled directly to machine code, whether Intel x64 or ARM instructions, or to JavaScript if they are intended for the web. It works by injecting updated source code files into the running Dart Virtual Machine. After the virtual machine updates classes with the new versions of fields and functions, the Flutter framework automatically rebuilds the widget tree, allowing developers to instantly view the effects of the changes. (Flutter Developers 2023.)

3.1.1 Widgets

As expressed, Flutter emphasizes widgets as a unit of composition. Widgets are the client interface-building pieces of a Flutter app, and each widget is an unchanging affirmation of a parcel of the client interface. Widgets are organized into a chain of command based on their composition. Each widget comes inside its parent and can secure a setting from it. As this basic case appears, this structure amplifies to the root widget frequently “MaterialApp” for Android or “CupertinoApp” for iOS. (Flutter Developers 2023.)

Widgets in Flutter are ordinarily built from a collection of little and single-purpose components. The framework points to playing down the number of unmistakable plan concepts while guaranteeing a wide building square. This approach is outlined in different layers of Flutter's design. For occasion, within the widgets layer, Flutter utilizes a center concept that covers a wide run of capacities, such as drawing to the screen, format, client interactivity, state administration, theming, liveliness, and route. This bound-together center concept rearranges improvement and guarantees consistency. Essentially, the activity layer rotates around a combination of concepts, movements, and tweens, which collectively cover an endless plan space. The rendering layer depends on “RenderObjects” to portray format, portray, hit testing, and availability. This approach comes in an assorted building piece of widgets, render objects, activity, and tween sorts. (Flutter Developers 2023.)

Flutter's approach is additionally distinctive from other conventional frameworks. It keeps the structure of its building pieces wide and not as well profound. This implies developers can combine these blocks in numerous ways to realize diversity comes. Indeed, fundamental things like adding space around a component or centering something are not built into the center structure. Instep, designers utilize partitioned building blocks for these assignments. For example, to center something, developers can wrap it in a “Center” building piece. There are different building squares for controlling format, like including space or orchestrating things in rows and columns. These format building blocks do not have a visual appearance by themselves, but they exist to control how other widgets or components are organized on the screen. (Flutter Developers 2023.)

Flutter's system is built around two fundamental classes of widgets which is stateful and stateless widgets. These two categories serve unmistakable purposes within the advancement of applications. Stateless widgets are widgets that have no changeable state and properties not changing extra time. These widgets are frequently utilized for showing inactive data or components of the client interface

that do not require overhauls or changes amid the application's lifecycle. They are highly efficient since they are lightweight and do not have to track any inside changes. (Flutter Developers 2023.)

CODE 1 illustrates a stateless widget responsible for returning a single “MaterialApp” widget. This widget, a versatile container encapsulating commonly used components in material design applications. It not only provides a convenient means to modify the primary theme color but also offers an array of essential widgets for constructing visually appealing interfaces. The “build” method within the stateless widget is invoked each time a redraw is necessary, ensuring dynamic responsiveness in rendering the user interface. Additionally, this design pattern promotes modular updates, allowing for seamless adjustments to the visual elements and theme preferences as the application grows.

```
class MyApp extends StatelessWidget {
  const MyApp({super.key});

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ), // ThemeData
      home: const MyHomePage(title: 'Flutter Demo Home Page'),
    ); // MaterialApp
  }
}
```

CODE 1. Example code for stateless widget, captured from project development.

On the contrary, stateful widgets store mutable state in a separate class, known as a state object. Stateful widgets do not have a build method, instead, the user interface for stateful widgets is constructed and modified through the state object associated with that widget. The build method resides in the state object and is called whenever the widget needs to be re-rendered due to changes in its mutable state. This separation of concerns between the stateful widget and its associated state object allows for a more organized and efficient approach to managing dynamic user interfaces. (Flutter Developers 2023.)

CODE 2 shows the “MyHomePage” widget, which is stateful. It returns a scaffold widget. The scaffold widget contains an “AppBar”, a “FloatingActionButton”, and a body attribute that may be

assigned to any widget. A body property set to a “Center” widget with a “Column” widget as its child. A “floatingActionButton” property set to a “FloatingActionButton” widget with a “onPressed” property set to “_incrementCounter”, and a child property set to an Icon widget with the “Icons.add” icon. “_incrementCounter” is the method that adds one value to the state “_counter” whenever users click on the floating button.

```
class MyHomePage extends StatefulWidget {
  const MyHomePage({super.key, required this.title});

  final String title;

  @override
  State<MyHomePage> createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Theme.of(context).colorScheme.inversePrimary,
        title: Text(widget.title),
      ), // AppBar
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            const Text(
              'You have pushed the button this many times:',
            ), // Text
            Text(
              '$_counter',
              style: Theme.of(context).textTheme.headlineMedium,
            ), // Text
          ], // <Widget>[]
        ), // Column
      ), // Center
      floatingActionButton: FloatingActionButton(
        onPressed: _incrementCounter,
        tooltip: 'Increment',
        child: const Icon(Icons.add),
      ), // FloatingActionButton
    ); // Scaffold
  }
}
```

CODE 2. Example code for stateful widget, captured from project development.

3.1.2 State management with Riverpod

Riverpod is a receptive caching system for Shudder and Dash. Riverpod handles much of the application's rationale through explanatory and receptive programming. It can handle organized inquiries with built-in blunder taking care of and caching, and it will naturally revive information as required. Advanced applications once in a while incorporate all the data required to show a client interface. Information is as often as possible recovered nonconcurrently from a server. The inconvenience is that working with offbeat code is troublesome. In spite of the fact that Vacillate incorporates a strategy for making state factors and reviving the UI when conditions alter, it is right now very restricted. A parcel of issues stays uncertain, such as the need to cache offbeat questions locally since it would be irrational to re-execute them each time the UI changes. (Riverpod Developer 2023.)

Because the application contains a cache, its substance may go out of date if engineers are not paying consideration. Designers are moreover dependable for dealing with mistakes and stacking states. Idealizing those challenges at scale can be extreme, and they are affected by a wide run of highlights, like drag to revive, unbounded records or recover as clients scroll, look like users' input, and offline mode. These highlights can be troublesome to create, but they are basic for a positive client encounter. Be that as it may, a few bundles endeavor to address those issues straightforwardly, and much of the exertion has to be done physically. This is where Riverpod comes in. Riverpod endeavors to address these issues by giving an unused, unmistakable approach to building trade rationale, propelled by Vacillate widgets. (Riverpod Developer 2023.)

Providers are the most vital portion of a Riverpod application. A supplier is a question that typifies a piece of state and permits tuning in to that state. Wrapping a piece of state in a supplier permits effortlessly getting to that state in numerous areas. Suppliers are a total substitution for designs like Singletons, Benefit Locators, Reliance Infusion, or acquired widgets. It moreover streamlines combining this state with others. It empowers execution optimization. Whether for sifting gadget revamps or for caching costly state computations, suppliers guarantee that as it were what is affected by a state alter is recomputed. It increases the testability of the application. Besides, any supplier can be abrogated to carry on in an unexpected way amid a test, which permits effortlessly assessing a particular behavior. (Riverpod Developer 2023.)

Riverpod applications depend intensely on suppliers. A supplier is a protest that contains a piece of state and permits designers to tune in to it. Covering a piece of state with a supplier lets designers rap-

idly get to it from anyplace in a chain of command of widgets. Suppliers totally supplant designs such as singletons, benefit locators, reliance infusion, and acquired widgets. It makes it less demanding to combine this state with others. It empowers execution upgrades. Suppliers guarantee that as if the components influenced by a state alter are recomputed. It progresses the testability of the application. Moreover, any supplier can be abrogated from acting in an unexpected way amid a test, permitting for the simple testing of greatly specific behaviors. (Riverpod Developer 2023.)

TABLE 2. Different types of Riverpod providers (Riverpod Developers 2023)

Provider Type	**Provider Create Function**	**Example Use Case**
Provider	Returns any type	A service class / computed property (filtered list)
StateProvider	Returns any type	A filter condition / simple state object
FutureProvider	Returns a Future of any type	A result from an API call
StreamProvider	Returns a Stream of any type	A stream of results from an API
NotifierProvider	Returns a subclass of (Async)Notifier	A complex state object that is immutable except through an interface
StateNotifierProvider	Returns a subclass of StateNotifier	A complex state object that is immutable except through an interface. Prefer using a notifierProvider
ChangeNotifierProvider	Returns a subclass of ChangeNotifier	A complex state object that requires mutability

A basic provider stated in the Dart language can be seen in CODE 3. The annotated function “my,” the reference “ref,” and the annotation “@riverpod” are the three key components. Annotating each provider with "@riverpod" or "@Riverpod()" is mandatory. Developers can annotate classes or global functions with this. The interaction with the provider is determined by the name of the annotated function. A generated variable called “myProvider” will be created for a provided function called “my”. A “ref” has to be included as the first parameter in annotated functions. The function can have any number of parameters, including generics, aside from that. This function will be executed when the provid-

er is initially read. Subsequent readings will return the cached value rather than calling the function again. The “ref” object is used to communicate with other providers. Every provider has one, either as a parameter to the provider function or as a property of a Notifier. The function or class name determines the type of this object. (Riverpod Developer 2023.)

```
@riverpod
MyValue my(MyRef ref) {
  return MyValue();
}
```

CODE 3. Declare a provider with Riverpod annotation (Riverpod Developer 2023)

3.2 Stable Diffusion

Stable Diffusion is a counterfeit insight show that creates design based on content inputs or prompts. When a client gives a printed depiction, such as “a plane”, Stable Diffusion produces pictures that fit the portrayal of having a plane interior, based on its prepared information as outlined in FIGURE 2. The complexity of the given prompts has a coordinated effect on the made image’s quality, fashion, and level of detail. Stable Diffusion exceeds expectations in different spaces in expansion to text-to-image generation. It is able of consistently including and supplanting components inside an existing picture, a handle known as “inpainting”. It moreover has the capacity to expand a picture by filling in purge pixels, a strategy known as "outpainting”. These highlights make Stable Diffusion a valuable instrument for an assortment of graphic-related applications. Besides, it is vital to note that Stable Diffusion is as of now accessible as an open-source show, which implies it can be get to and utilized by anybody free of charge. (Laukkonen 2023.)

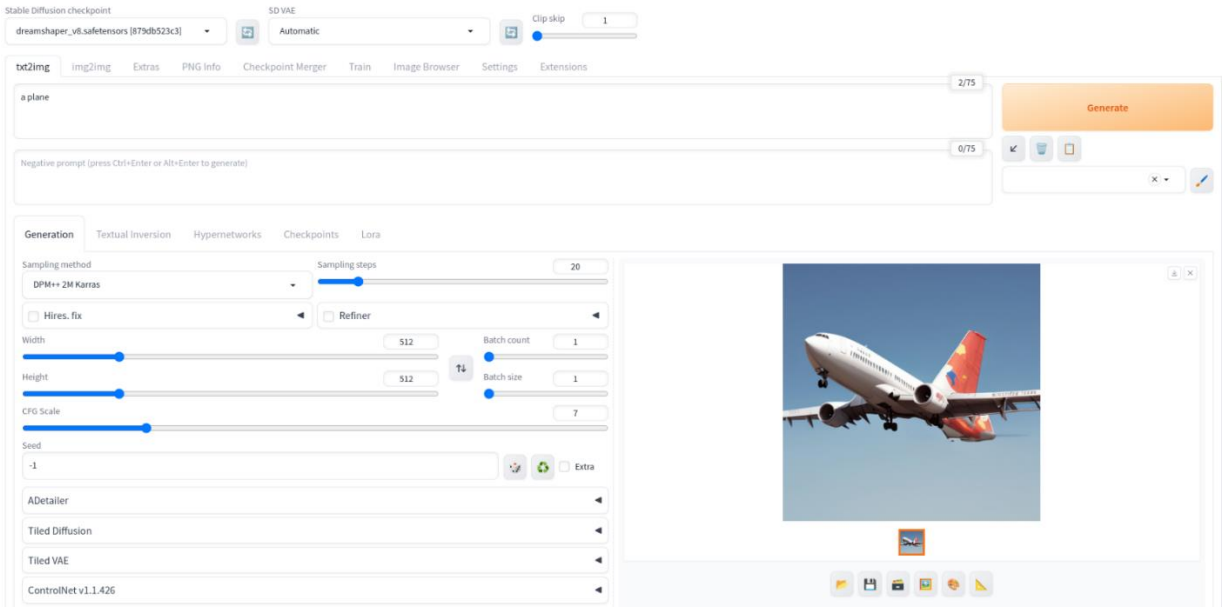


FIGURE 2. Generating image of a plane from Stable Diffusion that using AUTOMATIC1111 WebUI, captured from project development

3.2.1 How AI generate images from prompt

The innovation that enables AI models to comprehend and create pictures lies on two fundamental components which are natural language processing (NLP) and general adversarial networks (GANs). NLP, a fundamental aspect of AI, is the field devoted to educating machines on how to get and translate human dialects. It prepares AI models with the capability to handle and make sense of printed portrayals, such as “a beautiful scene with a beautiful sunset.” NLP enables these models to decipher human dialect into a shape that can direct the generation of images. Essentially, natural language processing (NLP) bridges the gap in human-machine communication by allowing AI systems to comprehend and react to natural language inputs. Brownlee (2019b) On the other hand, GANs represent a neural network that surpasses expectations in terms of generating up-to-date information and image counts by utilizing patterns and data extracted from previously published datasets. A discriminator and a generator are GAN’s two primary parts. The discriminator checks the data that the generator, in this case, provided in the form of images. (Brownlee 2019a.)

Training these models is a process that involves exposing them to a broad and varied dataset making up hundreds of millions of labeled images. During this training, the model analyzes the images and constructs an internal framework of probabilities that encapsulates the relationships within each image. For instance, it learns that a human face typically consists of two eyes, and a cat is characterized by its

four legs. To enhance the model's abilities, scientists insert visual noise, similar to the static on old televisions, challenging the AI to eliminate this noise and produce a clean image. This repetitive process is repeated multiple times, gradually increasing the complexity of the noise, and tasking the AI model to purify the image accordingly as shown in FIGURE 3. The outcome of this training is an AI model capable of interpreting textual descriptions and beginning with a noisy image, which then refines to align with the given description. (Bushwick 2023.)

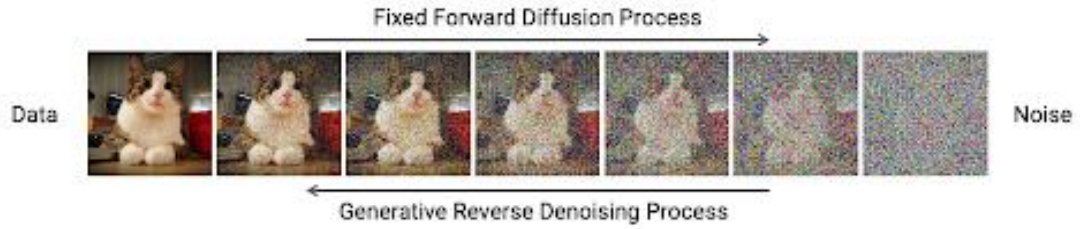


FIGURE 3. Diffusion model processes moving to and from data and noise (Vahdat & Kreis 2022)

However, major drawbacks to this method have to be considered. The AI model's inability to generate completely new. It is reliant on the training dataset, which limits its originality. Furthermore, complicated instructions can be challenging for these models, resulting in misunderstanding or mistakes in the output visuals. Because of the biases inside the training dataset, the AI model will have the same biases about classes, age, race, or gender. (Bushwick 2023.) For instance, when using the prompt "judge," only about 3% of the generated images by Stable Diffusion are women, while in reality, 34% of United States judges are women as show in FIGURE 4. This statistic is collected by United States Bureau of Labor Statistics tracks the race and gender of workers in every occupation through a monthly household survey and the dominated training dataset is from United States. Additionally, the model not only underrepresents women in high-paying jobs but also overrepresents them in low-paying positions. (Nicoletti & Bass 2023.)

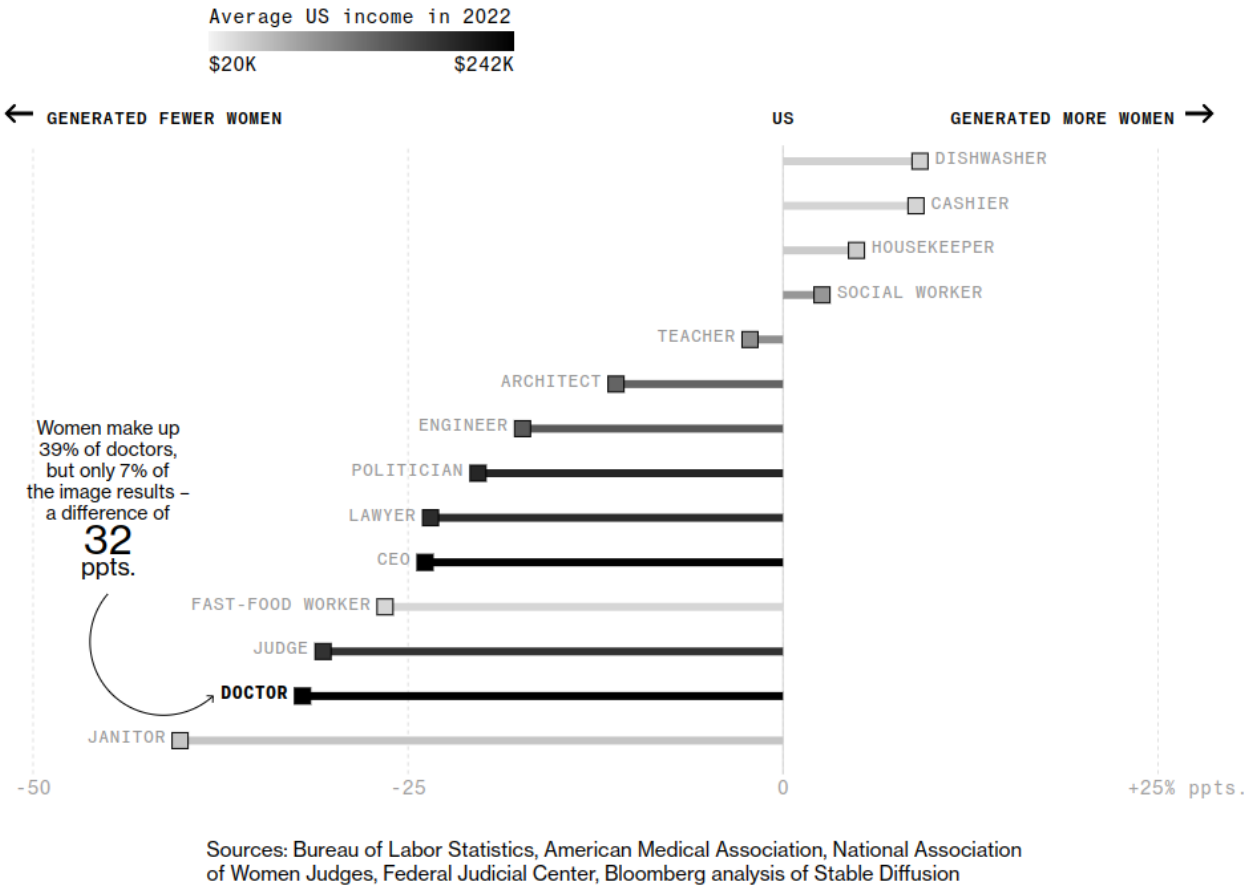


FIGURE 4. The difference in percentage between the reality statistics and generated image result by Stable diffusion in occupation for women (Nicoletti & Bass 2023)

Stable Diffusion's training data primarily comprises a substantial number of images sourced from the internet. These images are drawn from a variety of online platforms, with notable contributions from websites like Pinterest, DeviantArt, Flickr, and ArtStation. (Laukkonen 2023.) However, there may be instances of copyrighted material. This situation has sparked discussions and concerns related to issues of plagiarism and intellectual property rights (Bushwick 2023). PICTURE 1 shows the protest of artists on ArtStation against the decision of the platform which does not refuse to ban AI generated graphics. Protesters worry that AI-generated art is a copy of human artists' labor and frequently uses their creations without acknowledgment or a fee (Weatherbed 2022).

PICTURE 1. “No AI Art” images posted by artists started to dominate the trending section of ArtStation following the platform’s refusal to ban AI-generated artwork (Weatherbed 2022)

3.2.2 Introduction to AUTOMATIC1111 Stable Diffusion Web UI

AUTOMATIC111 Stable Diffusion Web UI is a robust and effective application for generating pictures from text prompts. Its user-friendly interface, extensive parameter controls, and advanced capabilities make it a fantastic choice for both newbies and expert users who want to explore the creative potential of text-to-image AI. The application’s first noticeable signature is its simple UI. The graphical user interface (GUI) has a simple layout and intuitive controls, making it accessible to users of diverse skills in technology. Users can generate images by entering text prompts, allowing for creative expression and investigation of various concepts and ideas. (Tu, A.H. 2023.) The UI’s main features are grouped into tabs: “txt2img”, “img2img”, “Extras”, “PNG info”, “Checkpoint Merger”, “Train”, “Settings”, and “Extensions”. The “txt2img” tab is seen in FIGURE 5. It enables users to enter prompts, which aids the Stable Diffusion model in generating images.

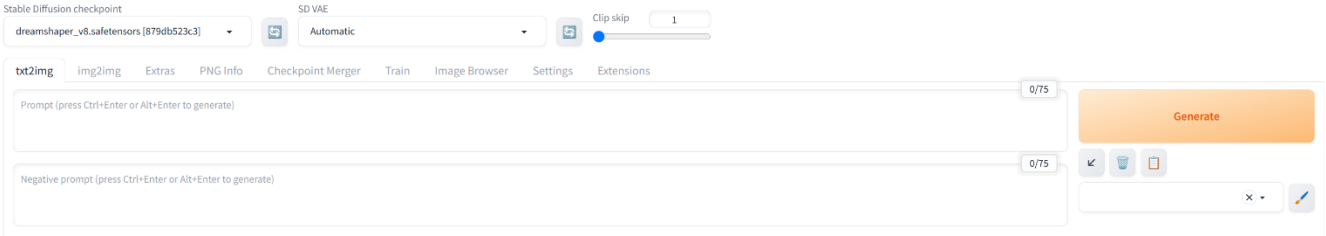


FIGURE 5. “txt2img” tab where users input prompt to generate image, captured from project development

Its arrangement of inspiration forms the basis of AUTOMATIC111 Stable Diffusion Web UI's creative potential. Users can fully unleash the creative potential of AI by modifying limitations and rules with the aid of this effective tool. Two hidden input boxes on the "txt2img" tab, "Prompt" and "Negative Prompt," are specifically designed for text-to-image conversion. The canvas where clients can pass on their thoughts for the wrapped-up picture is the "Prompt" box. It enables them to depict the precise points of interest, appearances, and common tones they select to depict. The more particular and clear the jolt, the more obvious it is that the AI gets it the specified result. By giving a select few fundamental topics of interest that absolutely facilitate the image's substance, color plot, lighting, and composition, clients can reasonably supervise the AI's creative process. The "Negative Prompt" expansion has been included within the "Prompt" box so that clients can demonstrate what they do not need to see inside the final thing and work toward finishing their pined for the result. This negative input is utilized by the AI to channel out undesirable components, ensuring that the wrapped-up thing totally fulfills the user's desires. (Tu, A.H. 2023).

PICTURE 2. Generated image with prompt “a cat”, captured from project development.

The establishment for the coherent advancement of pictures is the association between the "Prompt" and the "Negative Prompt." The pictures displayed in PICTURE 2 have the positive instigator "a cat" and no negative effect. It can be a common trigger, which might result in unpredictable visits. Clients can help scholarly people communicate their vision to the AI by giving it unambiguous data within the positive prompt and allowing undesirable suppositions within the negative prompt. As a result, AI is more pivotal to creating exact and solid yields. Particularly, as the signals ended up more correct, the client's influence on the wrapped picture developed. The created pictures with the positive input "a cat with white fur" and the negative incite "sitting at home" are shown in PICTURE 3. The incite is more change and can summon the full circle picture of a kitty with a white stowaway who isn't contributing all day interior of the house.

PICTURE 3. Generated image with prompt “a cat with white fur” and negative prompt “sitting at home”, captured from project development.

In this tab, users can play with classifier-free guidance (CFG) scale to create more variances of the generated image. It is an important choice that determines how strictly Stable Diffusion follows the content cue in both text-to-image and image-to-image jobs. The result will be more in alignment with what is entered prompt as well as the original picture if CFG Scale is increased, but it will be distorted. The smaller the CFG Scale value, on the other hand, the greater probable it will differ from the command or the uploaded image, but the higher the quality. The level of detail between the command and result is directly compared to the value of the CFG scale. The CFG Scale value and output quality are inversely related to each other. (Raj, G. 2023.) PICTURE 4 shows an example of what has been changed when the users keep the same prompt “a cat with white fur” but change the value of CFG scale between 5, 10, 15, 25.

PICTURE 4. Generated image with prompt “a cat with white fur” but in different CFG scale values, captured from project development

The “img2img” tab allows users to convert an uploaded image into a fresh style graphic. It provides different image style control parameters, such as resolution, color palette, and diffusion quality. This allows users to fine-tune the generated photos to their taste. Generated images can be saved in PNG format for easy sharing and preservation. This format can be utilized to extract significant insights from created photographs, thus improving the user experience. Users may easily access the original prompt, check specific parameter settings, and even extract image dimensions in the “PNG Info”. (Raj, G. 2023.) FIGURE 6 depicts the “PNG Info” tab’s user interface, which pulls information from the uploaded PNG file. Users can use this prompt to create a similar artwork or experiment with various seeds to discover new styles.

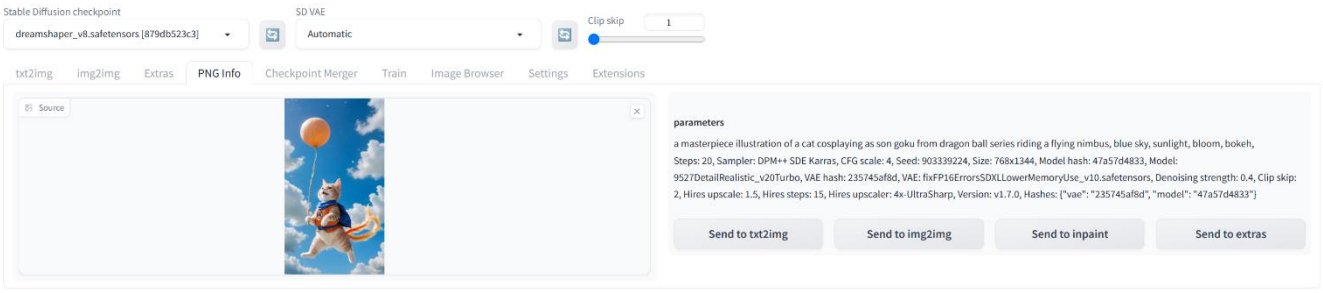


FIGURE 6. “PNG Info” tab where to view the information of AI generated image, captured from project development

3.3 Amazon Web Services

Amazon Web Services (AWS) began delivering IT infrastructure services to businesses as web services in 2006, giving rise to the term “cloud computing.” One of the most significant advantages of cloud computing is the potential to replace upfront capital infrastructure expenses with cheap variable costs on that scale. Because of the cloud, businesses no longer need to plan for and purchase servers and other IT equipment weeks or months ahead. Instead, they may quickly build up hundreds or thousands of servers and give results. Today, AWS offers a reliable, scalable, low-cost cloud infrastructure platform that supports hundreds of thousands of enterprises in 190 countries worldwide. (AWS Developers 2023g.)

3.3.1 AWS API Gateway

Amazon API Gateway is an Amazon Web Services (AWS) benefit that empowers engineers to construct, publish, maintain, monitor, and secure REST, HTTP, and WebSocket APIs at any scale. API designers can make APIs that provide customers get to AWS or other web administrations, as well as information put away in the AWS Cloud. APIs can be made by engineers for utilize in their claim client applications. Developers may, moreover, make APIs accessible to third-party app engineers. API Gateway, in pith, makes HTTP-based Restful APIs. It licenses client-server communication that is stateless. It makes utilize of standard HTTP methods including “GET”, “POST”, “PUT”, “PATCH”, and “DELETE.” (AWS Developers 2023a.)

3.3.2 AWS Lambda

AWS Lambda is a flexible computing benefit planned to disentangle the handle of running code without the requirement for server provisioning or administration. This benefit works by executing code on a high-availability compute framework while taking care of all related authoritative assignments. These errands incorporate server and working framework upkeep, the provisioning of computing assets, programmed scaling, and comprehensive logging. Developers utilizing Lambda have the comfort of giving their code in one of the backed dialect runtimes. This code is organized into discrete units known as Lambda functions, permitting secluded and proficient code administration.

Lambda functions are executed by the benefit as they were when required, guaranteeing asset productivity, and they consequently scale to suit changing workloads. Moreover, AWS Lambda engages engineers to center on their code and application rationale calming them of the burdens of server administration and asset assignment, coming about in a streamlined and hassle-free improvement prepare. This serverless computing demonstrates adaptability and effectiveness, guaranteeing that computing assets are designated as required without manual intercession. (AWS Developers 2023b.)

3.3.3 AWS SQS

Amazon Simple Queue Service (Amazon SQS) gives a dependable and accessible facilitated line that serves as a basic apparatus for coordination and decoupling dispersed program frameworks and components. This benefit offers a secure and strong arrangement for overseeing message queues in a dispersed design. Amazon SQS joins highlights like dead-letter lines, which are vital for taking care of messages that cannot be handled effectively, and fetched assignment labels to assist designers oversee costs successfully. These highlights upgrade the unwavering quality and taken a toll effectiveness of the informing framework. With a nonexclusive web administration API, Amazon SQS guarantees adaptability in its availability. Designers can be connected with this benefit by utilizing any programming dialect upheld by the AWS software development kit, making it available and flexible for designers working with assorted dialects and stages. This wide compatibility streamlines the integration of Amazon SQS into a wide run of program frameworks, making it a principal component for building versatile and resilient disseminated applications within the AWS ecosystem. (AWS Developers 2023f.)

3.3.4 AWS S3 Storage

Amazon Simple Storage Service (Amazon S3) may be a vital capacity advantage that stands out for its remarkable security, extraordinary execution, tremendous versatility, and steady information quality. It may be a flexible arrangement that is useful for putting away and shielding enormous sums of information for an assortment of applications, catering to desires of clients of all sizes and businesses. Amazon S3 demonstrates the importance of a wide cluster of utilize cases, counting but not being constrained to information lakes, facilitating websites, supporting portable applications, encouraging information reinforcement and reclamation, chronicling profitable data, serving undertaking applications, interfacing with IoT devices, and empowering broad enormous information analytics.

One of Amazon S3's key qualities lies in its comprehensive administration highlights. These instruments empower clients to fine-tune, oversee, and arrange to get to their put-away information to guarantee that it meets their trade destinations, organizational necessities, and compliance guidelines. Because of its flexibility, Amazon S3 is a fundamental component for any endeavor attempting to maximize the esteem of their information while keeping up the most elevated levels of information judgment and security. (AWS Developers 2023c.)

3.3.5 AWS DynamoDB

Amazon DynamoDB comes up as a completely overseen NoSQL database benefit that is at the cutting edge of innovation, empowering speedy and unsurprising execution while supporting adaptability effortlessly. DynamoDB shows up as an arrangement in database administration, permitting developers to center on advancement by facilitating the weight of commonplace authoritative exercises. This includes taking care of complex methods like equipment supply, setup, and setup, replication, computer program fixing, and cluster scaling while giving developers a disentangled and productive involvement. (AWS Developers 2023e.)

DynamoDB allows developers to design database tables with flexibility. These tables can store and retrieve massive amounts of data with ease, allowing them to handle a wide range of request volume. The platform’s dedication to delivering a dynamic and responsive environment for developers is demonstrated by the ability to grow tables’ throughput capacity without suffering downtime or performance loss. The AWS Management Console further empowers developers by providing a user-friendly interface for monitoring resource use and performance indicators, allowing for informed real-time decision-making. (AWS Developers 2023e.)

In addition to its performance and scalability capabilities, DynamoDB includes an on-demand backup option, giving developers the tools they need for long-term data retention and archive, which is required for regulatory compliance. A critical feature of data protection is the ability to produce on-demand backups and enable point-in-time recovery. Point-in-time recovery protects tables from unintentional data loss by allowing developers to restore a table to any point in the last 35 days, ensuring data resilience and business continuity. (AWS Developers 2023e.)

3.3.6 AWS CloudFront

Amazon CloudFront rises as a web benefit pointed to optimize the dispersion of both inactive and energetic web substance to conclusion clients, counting records such as “.html”, “.css”, “.js”, and pictures. CloudFront's unmistakable is the utilization of a worldwide arrange of information centers known as edge areas. These edge locales are basic within the substance conveyance handle since they guarantee that clients get asked for substance from the area with the least inactivity, hence maximizing execution. (AWS Developers 2023d.)

When a client starts a substance ask over CloudFront, the benefit scholarly people course the ask to the edge area with the least idleness, permitting the data to be conveyed rapidly. In case the asked data is as of now existent in this chosen edge point, CloudFront serves it immediately. If the substance is not accessible at that exact area, CloudFront brings it powerfully from a foreordained root, which may be an Amazon S3 bucket, a “MediaPackage” channel, or an HTTP server. (AWS Developers 2023d.) CloudFront optimizes the website's substance transmission by naturally directing each visitor's ask over the AWS backbone organized to the edge area that can serve the substance the most viably. This regularly includes directing the ask to a CloudFront edge server to guarantee the speediest conveyance to the watchers. Utilizing the AWS network diminishes the number of systems that consumers' demands have to pass through, hence progressing performance. Reduced inactivity, eminently the time it takes to stack the primary byte of a record, as well as higher information exchange speeds, advantage watchers. (AWS Developers 2023d.)

CloudFront moves forward the constancy and accessibility of a website's substance by purposely putting away duplicates of information, moreover, known as objects, over various edge areas all through the world. This worldwide caching approach progresses accessibility and constancy by permitting guests to retrieve content rapidly and with negligible delay over changed topographical locales. In substance, CloudFront could be an all-encompassing arrangement that not as it were quickens substance conveyance but moreover makes strides web application execution, solidness, and accessibility. (AWS Developers 2023d.)

3.4 Serverless Framework

The Serverless Framework is a powerful tool for developing and deploying AWS Lambda functions, along with the required AWS infrastructure resources. This framework uses a command line interface to provide an organized, systematic, and best-practice-driven approach from the start. Its major goal is to simplify the development process, allowing developers to focus on constructing complex, event-driven, serverless computing systems comprised of Functions and Events. (Serverless Framework Developers 2023.)

What separates the Serverless Framework from other application frameworks is its ability to manage not just the source code but also the related infrastructure. It streamlines the development lifecycle by providing a full solution that includes both application logic and the required cloud resources. This integrated method reduces manual configuration and management overhead, resulting in enhanced efficiency and faster development cycles. (Serverless Framework Developers 2023.)

The Serverless Framework's language flexibility is another differentiating feature. It supports a variety of programming languages, including Node.js, Python, and Java. This adaptability allows developers to choose their preferred language, resulting in a more inclusive and adaptable working environment. Whether a project necessitates the efficiency of Node.js or the robustness of Java, the Serverless Framework accommodates a wide range of language preferences, meeting the needs of a wide variety of developers. (Serverless Framework Developers 2023.)

3.4.1 Functions

Functions are the foundation of the Serverless Framework. A function within the application is the code deployed and executed within AWS Lambda functions. Each function, like a microservice, is an autonomous unit of execution and deployment. A function is a specific set of instructions deployed in the cloud environment and often designed to do a single task. This task can include anything from saving user data to a database to processing a file within a database to doing scheduled actions at predetermined times. (Serverless Framework Developers 2023.)

CODE 4 describes instances of declaring functions within "serverless.yaml." It is the file that holds the configuration for Lambda functions, S3 buckets, SQS queues, and DynamoDB. When declaring a

Lambda function, it is necessary to know what the handler function is and what event will be used to run the handler. The login endpoint is “login,” and it is taken care of by a file named “login.js” inside the “auth” directory, its endpoint is “/api/auth/login,” and its valid method is “POST.” Lambda functions can use global configuration environment variables or restrict themselves to a subset of them. The registration route can use an environment variable named “QUEUE_URL” that cannot be taken by other endpoints such as login and logout.

```yaml
functions:
  # Authentication routes
  login:
    handler: auth/login.handler
    events:
      - httpApi:
          path: /api/auth/login
          method: post

  logout:
    handler: auth/logout.handler
    events:
      - httpApi:
          path: /api/auth/logout
          method: post

  register:
    handler: auth/register.handler
    events:
      - httpApi:
          path: /api/auth/register
          method: post
    environment:
      QUEUE_URL: ${construct:users-created-queue.queueUrl}
```

CODE 4. Declare functions inside “serverless.yaml” file, captured from project development

3.4.2 Events

Events serve as the trigger for the orchestration of function execution. These events come from a range of AWS resources, demonstrating how versatile and scenario-adaptable the framework is. An HTTP request on an API Gateway URL, a new file uploaded to an S3 bucket, the start of a CloudWatch routine that runs every five minutes, or a message posted in an SNS topic are a few examples of event sources. The Serverless Framework takes care of the underlying infrastructure requirements smoothly when developers construct an event for a Lambda function. For instance, if an HTTP request event is configured, Serverless Framework automatically generates the necessary infrastructure components,

such as an API Gateway endpoint, and configures the associated functions to respond to this event. This automated infrastructure provisioning eases developers from the complexities of manual setup and configuration, streamlining the development process. (Serverless Framework Developers 2023.)

CODE 5 shows an example of declaration of a function with event in serverless framework. All the events and functions are declared inside “severless.yaml” file. The foundation of a function is included handler name, events of handling, and optional environment variables. In the picture, the register endpoint is managed by the file named “register.js” which lies inside the “auth” directory. The client an access to the endpoint with a “POST” method HTTP request which is sent to “/api/auth/register”. Function can access global or private environment variables by giving declaration inside environment property. The register function can use an environment variable named “QUEUE_URL” that contains the URL string for new user created queue.

```
register:
  handler: auth/register.handler
  events:
    - httpApi:
        path: /api/auth/register
        method: post
  environment:
    QUEUE_URL: ${construct:users-created-queue.queueUrl}
```

CODE 2. Declare functions with event inside “serverless.yaml” file, captured from project development

3.5 Event-driven architecture

The event-driven architecture (EDA) is a form of software architecture and application design approach. An event-driven system is intended to capture, communicate, and process events between services that are disconnected. This means that systems can stay asynchronous while still exchanging data and performing tasks. Many modern application designs, such as customer engagement frameworks that have to use client data in real time, are event-driven. Because event-driven is a programming approach rather than a language, event-driven apps can be written in any programming language. Because event-driven architecture allows for minimal coupling, it is a viable choice for modern, distributed application architectures. An EDA is loosely connected because event producers

are unaware of which event consumers are listening for an event, and the event is unaware of the consequences of its occurrence. (RedHat.com 2019.)

In the design of event-driven systems, decoupling and loose coupling are independent but interconnected ideas. Decoupling means decreasing direct relationships between the parts of a system and ensuring that no single component is heavily reliant on another. Decoupling is archived in event-driven architecture (EDA) by allowing components to generate events without regard for specific consumers, promoting independence and flexibility. Loose coupling, a type of decoupling, tries to lessen interdependence between components without separating them completely. Components interact in a loosely connected system without establishing reliance, promoting flexibility and independence. Decoupling and loose coupling both improve system agility and scalability by encouraging component independence. (RedHat.com 2019.)

Due to decoupling, EDA requires event producers (publishers) detecting events and portraying them as messages without knowledge of the event consumers or outcomes. When an event is identified, it is broadcast to event consumers via an asynchronous event processing platform via event channels. Consumers who are aware of the event may process it or be affected by it. The event processing platform responds appropriately, and subsequent action exposes the event's consequence. (RedHat.com 2019.)

A pub/sub model or an event stream model can be used by an EDA. The pub/sub model operates on a messaging infrastructure where subscribers are connected to an event stream. In this setup, an event is sent to subscribers after its occurrence or publication to keep them informed. In contrast, the event stream model involves writing events to a log. In this paradigm, consumers do not subscribe to an event stream; rather, they can read from any part of the stream and join at any moment. Diverse types of event streaming exist, such as event stream processing, which leverages a data streaming platform to ingest and process or transform events. Simple event processing triggers an immediate action in the consumer upon event occurrence, while complex event processing involves the consumer processing a series of events to identify patterns. Event stream processing is employed to detect meaningful patterns within event streams. (RedHat.com 2019.)

Event-driven architecture enables organizations in the development of adaptive systems that improve workflows by adapting to changes and enabling decision-making in real time. Real-time awareness of circumstances ensures that both manual and automatic business choices can use up-to-date data that

reflects the present status of systems. Events gathered as they happen from sources such as IoT devices, apps, and networks allow for real-time sharing of status and response data between event producers and consumers. (RedHat.com 2019.)

The use of event-driven architecture in systems and applications improves scalability, reactivity, and access to critical data for better corporate decision-making. The benefit of decoupling is introduced by EDA, which eliminates the requirement for direct connection between data or service suppliers and users. This decoupling supports a more flexible and scalable system, facilitates the integration of new components, encourages fault tolerance, and improves overall system efficiency. (RedHat.com 2019.)

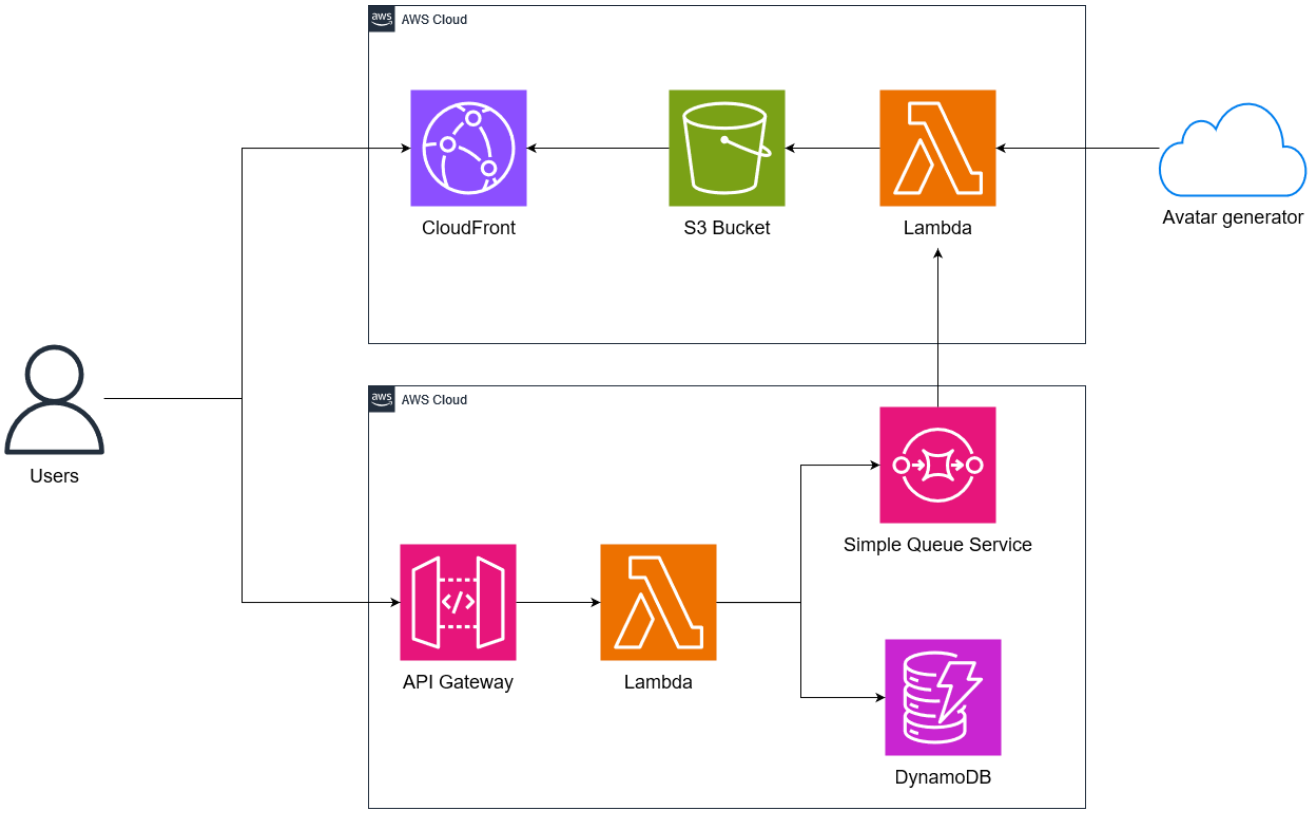


FIGURE 7. Workflow for register new account using API Gateway, Lambda, Simple Queue Service, DynamoDB, S3 Storage, and CloudFront

FIGURE 7 shows the workflow of user registration process in the Flutter application which launches a sequence of events orchestrated through AWS services. Upon a new user signing up, the user information seamlessly transits the API Gateway and is processed by a Lambda function. This Lambda function efficiently carries out two essential tasks. Firstly, the function saves the user information into DynamoDB, ensuring persistent storage of crucial user data. Simultaneously, as the Lambda function executes, it generates a message containing a unique user identification code, which is then dispatched to Amazon Simple Queue Service (SQS). The message landing in SQS serves as a trigger, initiating

another Lambda function designed to download a new image from an external API. This image is subsequently stored in an Amazon S3 Bucket for future retrieval. The stored image in the S3 Bucket is intelligently managed through Amazon CloudFront, offering users efficient and secure access to requested images. CloudFront not only optimizes image delivery but also serves as a protective layer, restricting access to the images solely to authenticated users.

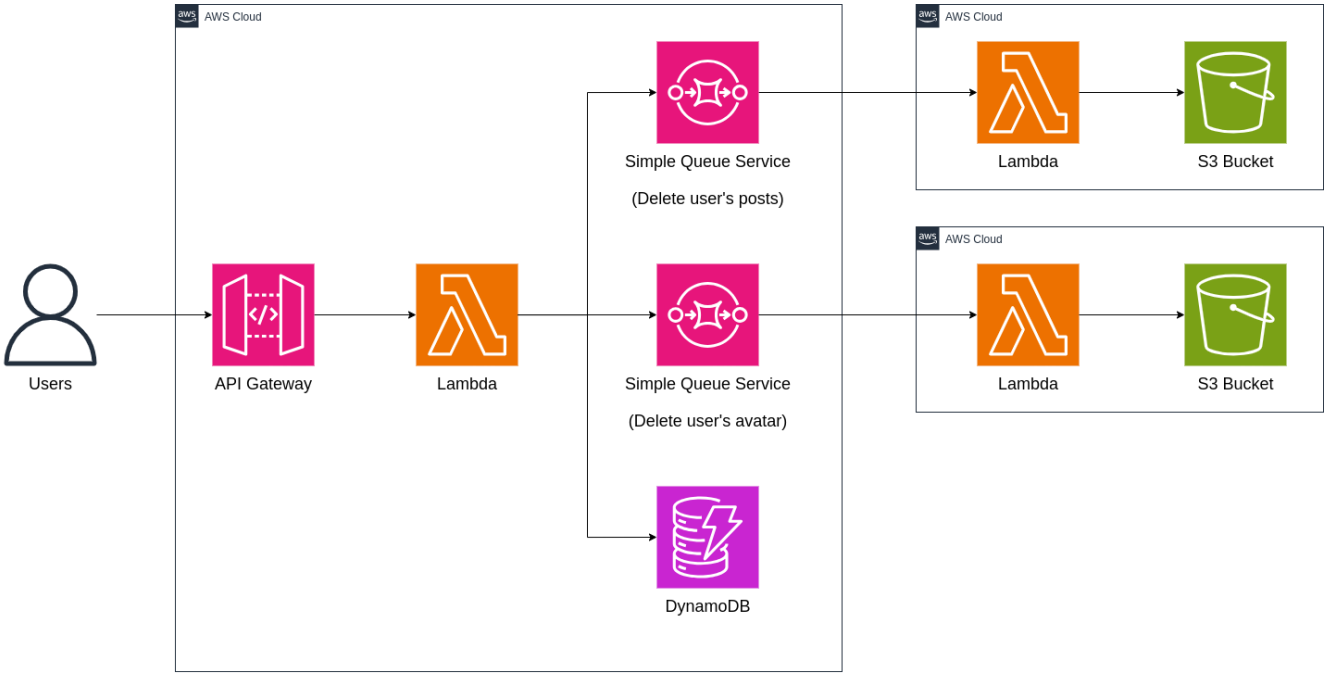


FIGURE 8. Workflow for delete user's account using API Gateway, Lambda, Simple Queue Service, DynamoDB, and S3 Storage

The event-driven workflow involved in the back-end preparation for removing a user's account is visually represented in FIGURE 8. When a client uses the AWS API Gateway to send an inquiry to the back-end, the method is initiated. This request is always handled by a Lambda function, which is given two primary tasks to complete: first, deleting the user's data stored in DynamoDB, ensuring that all relevant client information is removed; and second, sending a message to AWS Simple Queue Service (SQS) that includes the client's unique proof code. After receiving the message, SQS initiates a Lambda task that is assigned to it, which initiates the removal of the user's avatar and postings from two separate S3 buckets.

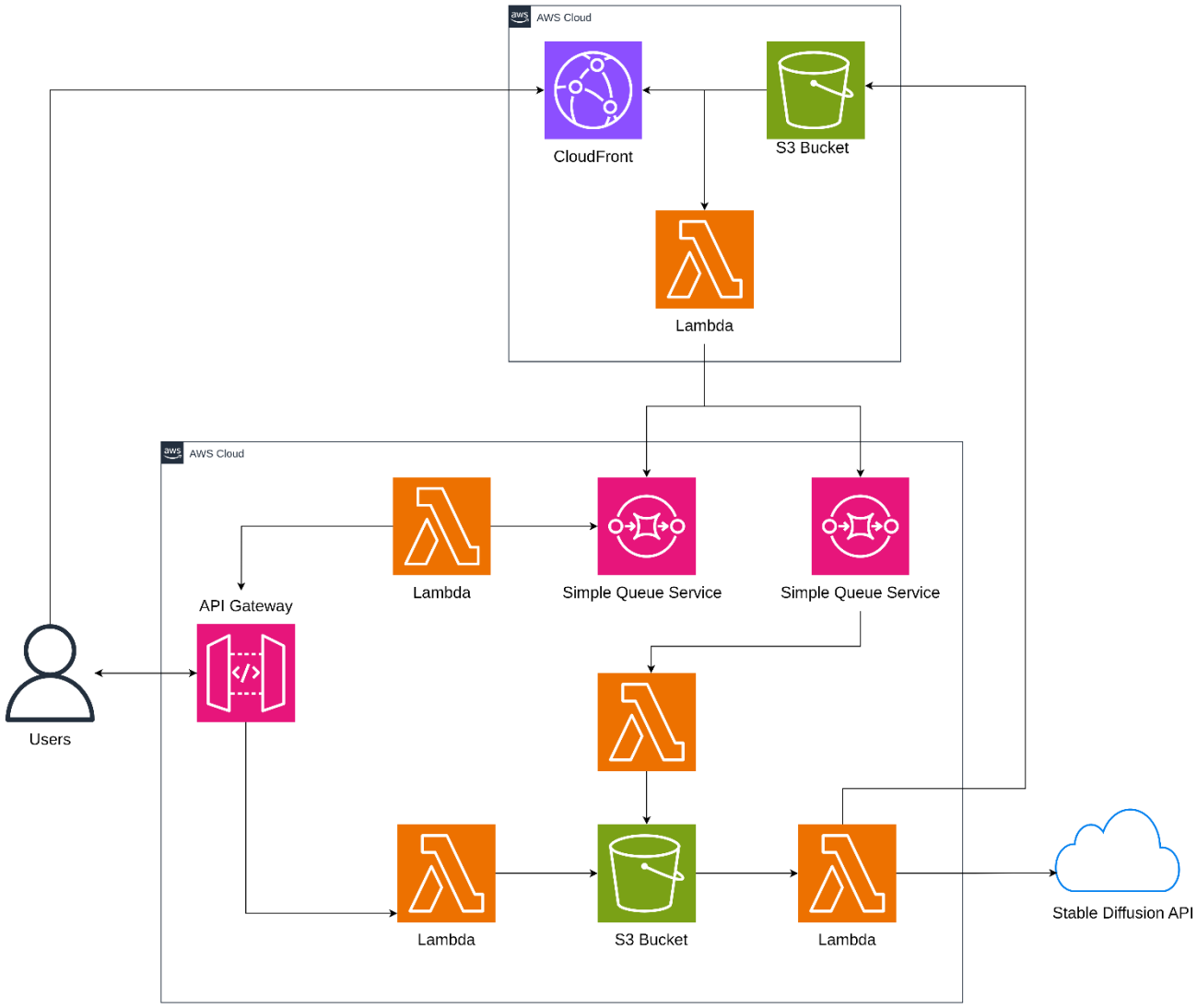


FIGURE 9. Workflow for creating from photo to “Anime” picture utilizing API Gateway, Lambda, Simple Queue Service, S3 Bucket, and CloudFront.

FIGURE 9 gives a comprehensive diagram of the event-driven workflow included in changing a photo into an “anime” picture. The method dispatches with the client uploading a photo to an S3 bucket for the conservation of crude pictures, encouraged through the API Gateway and a Lambda function. In this way, the client ceaselessly sends demands, anticipating the prepared picture. Upon a transfer occasion activated by S3 storage, a Lambda function comes into play, sending the picture to the pre-configured Stable Diffusion API. This API forms the picture and returns the comes about, which are at that point put away in another S3 bucket. The fruitful sparing occasion in this unused bucket acts as a trigger for a Lambda function, starting the celebrity of messages to two partitioned SQS queues.

One line informs the client that the picture has been effectively handled, whereas the other lines and ask for the erasure of the crude photo. Clients get the prepared pictures once the initial photo recog-

nizable proof is included in the line for handled comes about. Openness to these pictures is secured through CloudFront, including an extra layer of assurance. This perplexing event-driven engineering consistently arranges different components, optimizing the change handle and guaranteeing clients get handled pictures safely and proficiently.

4 FRONT-END IMPLEMENTATION

This segment portrays the application's front-end executions, which draw on the innovations secured in prior areas, such as Flutter. The primary organization is to create the model in Figma, a web-based tool for making energetic and responsive client interfacing. It encourages the arranging of client interfacing and client encounters in the development of coding, as well as testing and feedback from potential customers. The primary area of this chapter talks about the venture structure, which incorporates the front end's components, pages, and courses. At that point, it examines how the front-end interfaces with the application's back-end, which oversees client information and intelligence through API. This area will show the front-end usage of each page and component in detail, counting the challenges and arrangements confronted amid the improvement handle.

4.1 Project structure

The front-end codebase is organized according to feature-first architecture, thoroughly arranging the Flutter project into four well-defined layers: data, domain, application, and presentation such as in FIGURE 10. The data layer serves as the foundation, interacting with third-party APIs to fetch and process external data. It employs Data Transfer Objects (DTOs) to represent this data in a structured format, often in JSON format for efficient network transmission. Repositories, acting as middleman between the data layer and the rest of the application, encapsulate data access logic and provide type-safe model classes for seamless data retrieval and manipulation within the application’s domain layer. (Bizzotto 2022.)

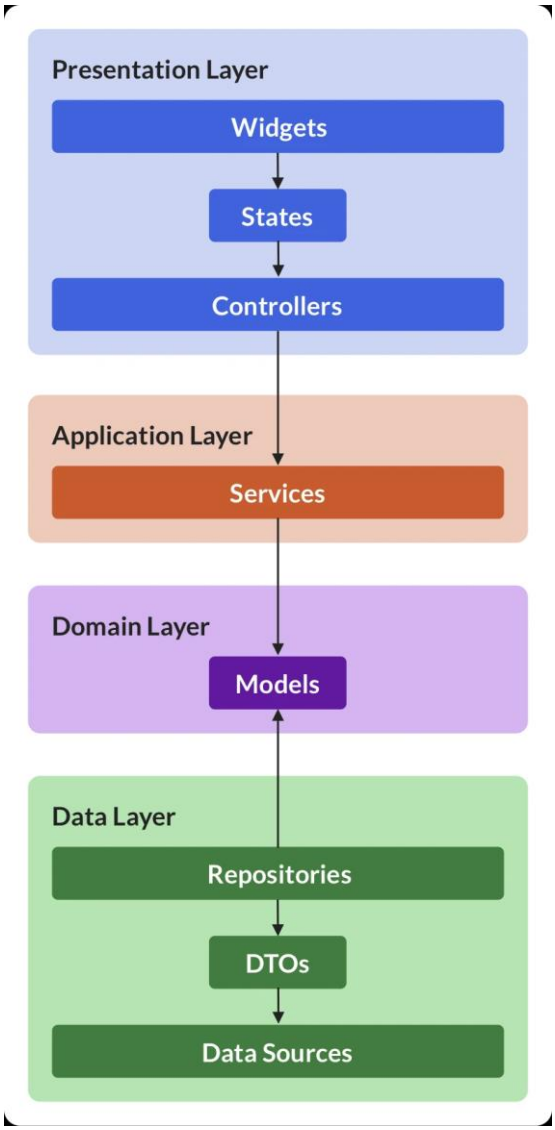
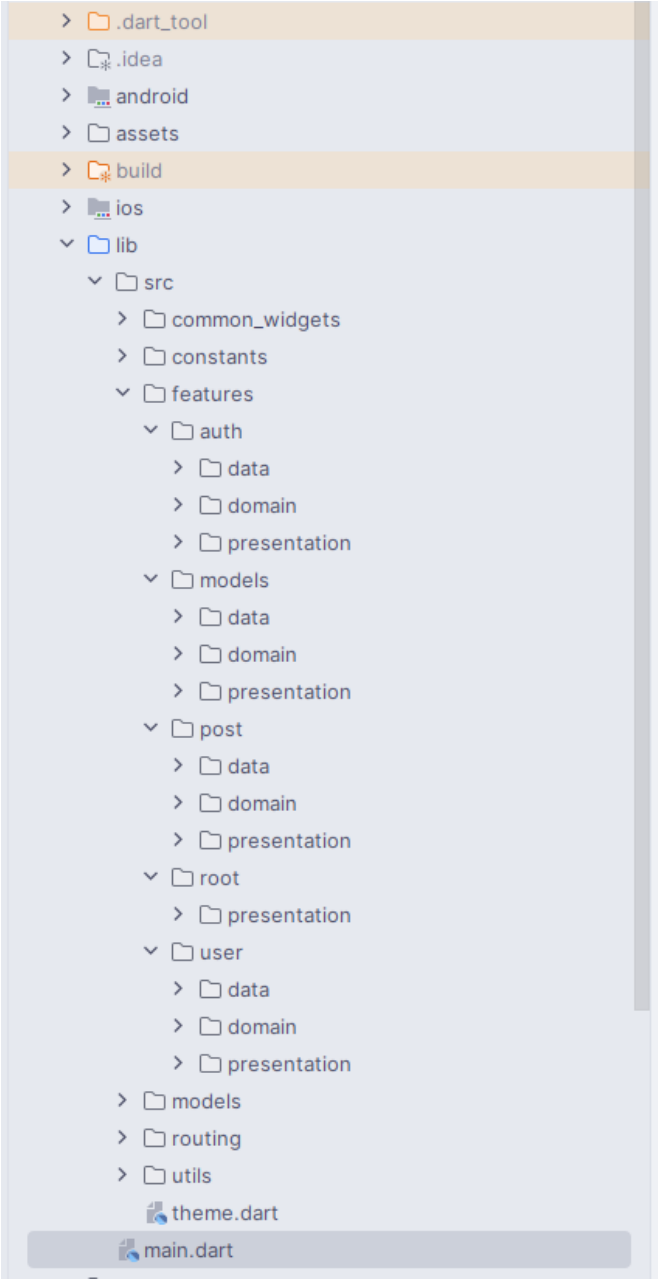


FIGURE 10. Flutter project structure using data, domain, application, and presentation layers (Bizzotto 2022)

The domain layer includes application-specific model classes that hold the data derived from the data layer. These model classes are equivalent to well-structured data repositories and comply to the tight standards outlined below. The first principle is immutability; model classes are naturally immutable, which ensures data consistency and prevents unintentional updates. The second feature is serialization; each model class has serialization logic, which allows for seamless data transformation between app-specific formats and external data sources. Methods like “fromJson” and “toJson” make this process easier. Finally, model classes support the equality operator and the “hashCode” function, which ensure consistent object comparison and quick data retrieval. These model classes serve as an intermediate between raw data and the application’s fundamental logic, ensuring data integrity and supporting efficient data management. (Bizzotto 2022.)

The application layer, though optional, serves as a crucial coordinator, assisting seamless interactions between controllers and multiple data sources within the data layer. These interactions, despite involving different data sources, utilize a unified model class to ensure data consistency. The presentation layer, the face of the application, is composed of two essential elements: widgets and controllers. Widgets act as the visual components that bring data to life on the screen, while controllers manage the dynamic states of the UI, handling loading indicators, success snack bars, and other interactive elements. (Bizzotto 2022.)

In the features-first approach, each layer will be stored in its own directory within each feature. Features vary between not only the screen that is displayed to the user, but also the abilities that the application provides (Bizzotto 2022). For example, PICTURE 5 illustrates the application structure, where the “features” directory includes “auth”, “models”, “post”, “root”, and “user”. The “auth” directory is used to manage authentication processes like login, registration, and logout. The “models” directory is utilized for model-related tasks such as selecting a model, creating a new post from selected model, and fetch models from the back-end. The “post” directory handles post-related tasks such as creating and deleting posts. “root” contains simply a screen for anonymous users. “user” covers the repository, domain, and presentation layers, which are responsible for handling authenticated user operations such as getting and updating user information and fetching saved posts.



PICTURE 5. Project structure of the front-end, captured from project development.

Shareable code, such as reusable widgets or utilization functions, can be stored outside of the features directory in distinct folders such as “utils”, “common_widgets”, “constants”, and “routing”. As an instance, the “routing” directory includes the application’s global router, which can be accessed from any widget by the path name. CODE 6 on the left side demonstrates a bit of routing configuration within the application. From a child widget such as “model card”, the user has access to the “create new post” tab by calling “GoRouter.of(context).push(“/upload-image”)” as seen as on the CODE 6 on the right side. All of the project’s files will use the same naming convention, “snake_case”. Screen files that are used to display the widget on the screen finish with “_screen”, whereas controller files end with “_controller”. This provides consistency and makes it simpler to search for files.

```dart
final _navigatorKey = GlobalKey<NavigatorState>();
GoRouter? _previousRouter;

@riverpod
GoRouter router(RouterRef ref) {
  final authState = ref.watch(authRepositoryProvider);

  return _previousRouter = GoRouter(
    initialLocation:
        _previousRouter?.routerDelegate.currentConfiguration.fullPath,
    navigatorKey: _navigatorKey,
    routes: [
      GoRoute(
        path: '/',
        builder: (context, state) => GettingStarted(),
      ),  // GoRoute
      GoRoute(
        path: '/login',
        builder: (context, state) => LoginScreen(),
      ),  // GoRoute
      GoRoute(
        path: '/register',
        builder: (context, state) => RegisterScreen(),
      ),  // GoRoute
      GoRoute(
        path: '/home',
        builder: (context, state) => RootScreen(),
      ),  // GoRoute
      GoRoute(
        path: '/upload-image',
        builder: (context, state) => UploadImageScreeen(),
      )  // GoRoute
    ],
    redirect: (context, state) {
      final isAtTheHomePage = state.fullPath == '/';
      final isAtTheLoginPage = state.fullPath == '/login';
      final isAtTheRegisterPage = state.fullPath == '/register';
      final isAuthenticated = authState.token != '';
```

```dart
class ModelCard extends ConsumerWidget {
  const ModelCard({super.key, required this.model});

  final Model model;

  @override
  Widget build(BuildContext context, WidgetRef ref) {
    _handleChooseModel(Model model) async {
      ref.read(selectedModelProvider.notifier).state = model;
      GoRouter.of(context).push("/upload-image");
    }

    return InkWell(
      onTap: () {
        _handleChooseModel(model);
      },
      child: Container(
        decoration: BoxDecoration(
          borderRadius: BorderRadius.circular(10),
          border: Border.all(
            width: 2,
            color: GetColors.grey,
          ),  // Border.all
        ),  // BoxDecoration
        child: ClipRRect(
          borderRadius: BorderRadius.circular(8),
          child: Image.asset(
            model.url,
            fit: BoxFit.cover,
          ),  // Image.asset
        ),  // ClipRRect
      ),  // Container
    );  // InkWell
  }
}
```

CODE 6. Global routing configuration using “GoRouter” package and Model card widget, placed inside “models/presentation”, captured from project development.

4.2 Perform data fetching and mutation.

Data flows throughout the application from the repository layer to the display layer via the controller. The project combines Riverpod 2.0 as state management solutions. The entire process of fetching data consists of four steps, as illustrated in FIGURE 11. The widget watches an “AsyncNotifierProvider”, which then calls a repository function to receive the data. This provider can be created by Rivepod code generation. The repository retrieves data from the data source or persistent data on the device. Even if users end their activities, their persistent state can be saved and retrieved using “SharedPreferences” from a key. When a response is received, it is parsed and returned by the repository. The widget gets data as “AsyncValue<Model>” and transfers it to the user interface. (Bizzotto A. 2023.)

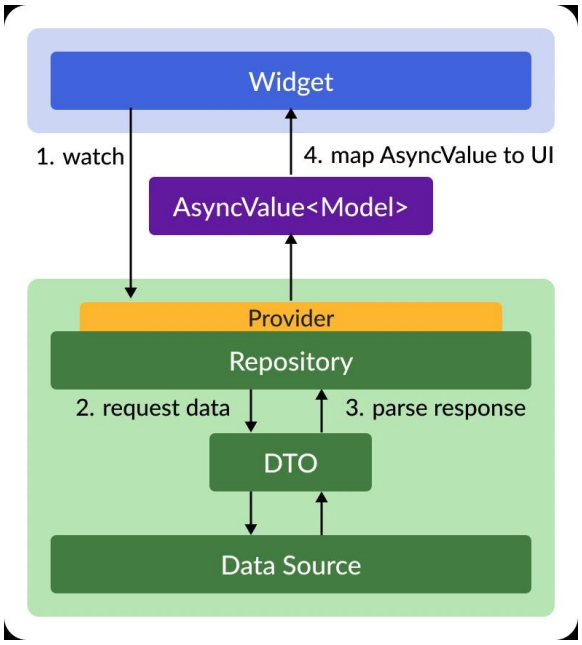


FIGURE 11. Simplified architecture when fetching data (Bizzotto 2023)

PICTURE 20 on the left side displays the home screen widget which may obtain the user data and posts from calling their providers. Instead of employing a stateful or stateless widget, it needs to be a “ConsumerWidget” that is able to be consumed “ref” in the “build()” method. Using “ref.watch()”, the user interface can stream to changes in the state and react accordingly. When the values for the user's information and posts are missing or loading from the repository, the front end displays the loading indicator instead. User is defined as an “AsyncNotifierProvider” with the Riverpod annotation as in CODE 7 on the right side. In its “build()” method, user data is first obtained from local storage, after which it is retrieved from the API using an authenticated token. “AsyncNotifierProvider” will return a “Future” value for the user, which represents asynchronous operations, and return the value when completed. It enables the widget to respond to state changes while the value is loaded.

```dart
class HomeScreen extends ConsumerWidget {
  const HomeScreen({super.key});

  @override
  Widget build(BuildContext context, WidgetRef ref) {
    final double topSafePadding = MediaQuery.of(context).padding.top + 10;

    final user = ref.watch(userRepositoryProvider).value;
    final posts = ref.watch(postsRepositoryProvider).value;
    final openOverlay = ref.watch(openOverlayProvider);

    return posts == null || user?.lastUseModels == null
        ? const Center(child: CircularProgressIndicator())
        : Stack(
            children: [
              CustomScrollView(
                slivers: [
                  SliverAppBar(
                    floating: true,
                    snap: true,
                    expandedHeight: 200,
                    automaticallyImplyLeading: false,
                    flexibleSpace: FlexibleSpaceBar(
                      background: Column(
                        crossAxisAlignment: CrossAxisAlignment.start,
                        children: [
                          Padding(
                            padding: EdgeInsets.only(
                              top: topSafePadding,
                              left: 10,
                            ),  // EdgeInsets.only
                            child: Text(
                              "Last used",
                              style: Theme.of(context).textTheme.bodyLarge,
                            ),  // Text
                          ),  // Padding
                          const SizedBox(
                            height: 16,
                          ),  // SizedBox
                          LastUsedModelsList(list: user!.lastUseModels!),
```

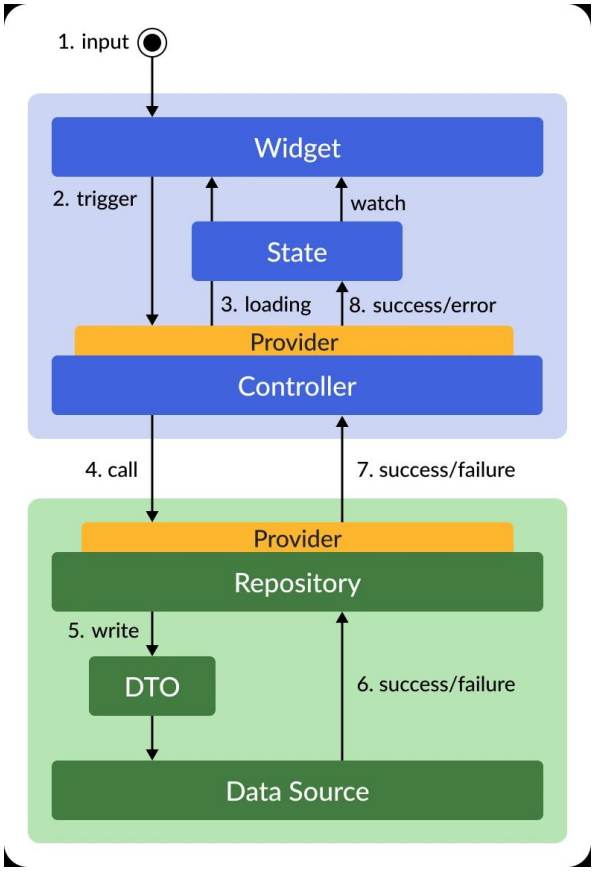
```dart
@riverpod
class UserRepository extends _$UserRepository {
  @override
  Future<User?> build() async {
    final storage = SharedPrefs();
    final savedUserJson = storage.user;

    // Prefer to use local model list
    if (savedUserJson != null) {
      Map<String, dynamic> raw = jsonDecode(savedUserJson);
      User saved = User.fromJson(raw);
      return saved;
    }

    // Fetch models list from online source
    final result = await _fetchUser();

    // Save models to local
    final userJson = jsonEncode(result);
    storage.user = userJson;

    return result;
  }
```

CODE 7. A piece of home screen code where is only authenticated users can visit and User repository, captured from project development

On the data mutation, PICTURE 21 show steps for edit or update data from the presentation layer to the data layer. It consists of steps happen in this order an input event takes place such as the user submitting a form. The input data is passed to the controller that will handle it. The controller sets the state to “loading” so the widget can update the user interface. The controller calls an asynchronous method, passing the data to be written to the repository. The repository converts the data to a data transfer object and writes it asynchronously to the data source. The response is received, or an exception is thrown. The success or error status propagates back to the controller, which can handle it. The controller sets the state to “success” or “error”, and the widget updates the UI.

PICTURE 6 illustrates the steps for editing or updating data from the presentation layer to the data layer. It comprises of steps that occur in this order, such as when a user submits a change request. The input data is routed to the controller that will process it. The controller toggles the state to “loading” so that the widget may modify the user interface. The controller invokes an asynchronous method, passing the data that will be written to the repository. The repository turns the data to a data transfer object and asynchronously sends it to the data source. The response is received, or an exception is

raised. The success or error status is communicated back to the controller, which can address it. The controller changes the state to “success” or “error”, and the widget updates the user interface accordingly. (Bizzotto 2023.)



PICTURE 6. Overview of the classes needed when performing a data mutation (Bizzotto 2023)

CODE 8 on the left side show the “SelectImageOverlay” widget, which previews a selected post from an authenticated user’s dashboard. It is preferable to use “HookConsumerWidget” instead of “ConsumerWidget” when applying Flutter hooks with Riverpod. Flutter hooks are an implementation of React hooks. “useState” and “useFuture” are used to save the user interface state for asynchronous operations. When an authenticated user sends a delete request, the “snapshot” enters a loading state and displays the loading indication using the comparison “snapshot.connectionState == ConnectionState.waiting” as seen as PICTURE 22 on the right side. The “_handleDelete” function reads the “homeScreenControllerProvider” notifier and interacts with data layer to delete the post. Passing “_handleOnSuccess” in the method of the controller if the response is successful and “_handleOnFail” otherwise. If success or failure occurs, a snack bar displays the message for the user.

```dart
class SelectImageOverlay extends HookConsumerWidget {
  const SelectImageOverlay({super.key});

  @override
  Widget build(BuildContext context, WidgetRef ref) {
    final double screenWidth = MediaQuery.of(context).size.width;
    final double previewImageWidth = screenWidth * 0.7;

    final isPending = useState<Future<void>?>(null);
    final snapshot = useFuture(isPending.value);

    final openOverlay = ref.watch(openOverlayProvider);
    final selectedImage = ref.watch(selectedImageProvider);

    _handleOnSuccess() {
      ShowSnackBar(context: context, message: "Delete post successfully!");
    }

    _handleOnFail() {
      ShowSnackBar(
          context: context, message: "Delete post fail! Please try again!");
    }

    _toggleOverlay() {
      ref.read(openOverlayProvider.notifier).state = !openOverlay;
    }

    _handleDelete() {
      final future = ref
          .read(homeScreenControllerProvider.notifier)
          .deletePost(onSuccess: _handleOnSuccess, onFail: _handleOnFail);
      isPending.value = future;
    }
                  StyledIconButton(
                    type: ButtonTypeEnum.secondary,
                    onIconTap: _handleDelete,
                    icon: Icon(
                      Icons.delete,
                      size: 30,
                      color: GetColors.red,
                    ),  // Icon
                  ),  // StyledIconButton
                ],
              ),  // Row
            ],
          ),  // Column
        ),  // Center
        if (snapshot.connectionState == ConnectionState.waiting) ...[
          const LoadingOverlay(),
        ]
      ],
    );  // Stack
  }
}
```

CODE 8. Select Image Overlay widget, captured from project development

CODE 9 displays the “HomeScreenController”, which is a “AsyncNotifierProvider” with no operations in the “build()” method. All business logic is asynchronous and independent from user interface interactions, so it is placed inside repository, not controller. Before the asynchronous operations are completed, the status will be set to loading with “AsyncLoading()” function. After getting the response from the repository layer, the controller can carry out the interaction using an if condition. In the code, the chosen picture is saved into another provider called "selectImageProvider". It is gathered as an argument for removing a post in the repository layer.

```dart
@riverpod
class HomeScreenController extends _$HomeScreenController {
  @override
  Future<void> build() async {}

  Future<void> deletePost(
      {required VoidCallback onSuccess, required VoidCallback onFail}) async {
    final selectedImage = ref.read(selectedImageProvider);
    state = const AsyncLoading();
    state = await AsyncValue.guard(() async {
      final res = await ref
          .read(postsRepositoryProvider.notifier)
          .deletePost(id: selectedImage!.id);

      if (res.status) {
        onSuccess();
      } else {
        onFail();
      }

      ref.read(openOverlayProvider.notifier).state = false;
    });
  }
}
```

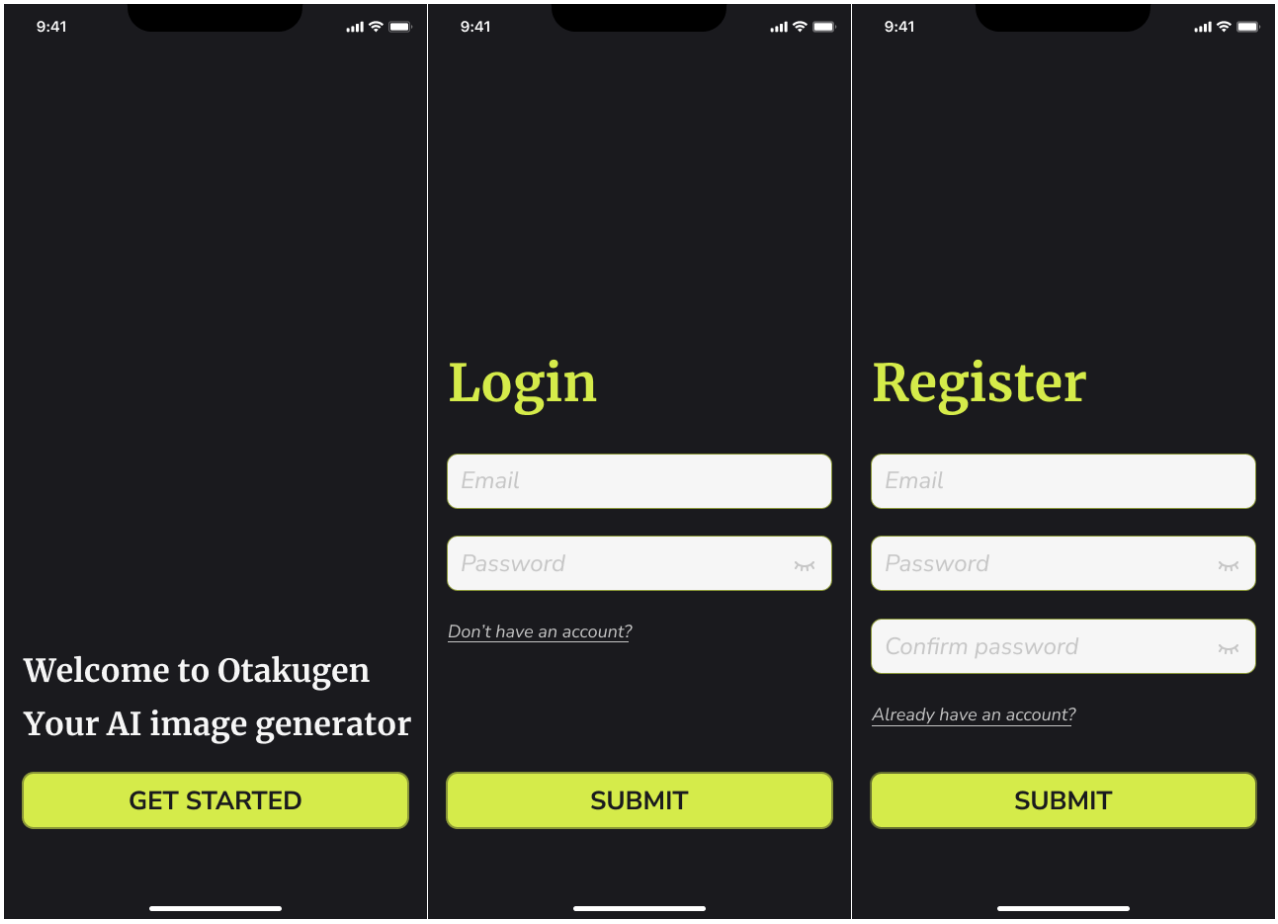
CODE 9. Home Screen widget, captured from project development

CODE 10 displays the delete feature inside the post repository. To delete an image, an “id” is required. All logic is enclosed in a try-catch block to handle exceptions when something goes wrong. Furthermore, removing a post is an asynchronous action. It is necessary to define the function using “Future” and “async”. The term “await” refers to waiting for the operation’s answer. First, an API call is used to delete a post from the online data source. If it is successful, it has to synchronize with the application state. Because the repository’s state is “AsyncValue”, the state has to also use “AsyncData” to wrap the data even though it is received as a list of posts “List<Post>”. Finally, the repository will respond to the controller, allowing the user interface to interact with the users.

```dart
Future<RepositoryResponse> deletePost({required String id}) async {
  try {
    if (state.value != null) {
      final storage = SharedPrefs();

      // Delete post in online source
      final res = await _deletePost(id);

      if (res.status == 200) {
        // New local state update
        final newUserPosts = state.value!.where((i) => i.id != id).toList();

        // Update local storage
        storage.posts = jsonEncode(newUserPosts);

        state = AsyncData(newUserPosts);

        return RepositoryResponse(
            status: true, message: "Delete post successfully!");
      }
    }

    return RepositoryResponse(
        status: false, message: "Failed to delete post!");
  } on Exception catch (e) {
    return RepositoryResponse(
        status: false, message: "Failed to delete post!");
  }
}
```

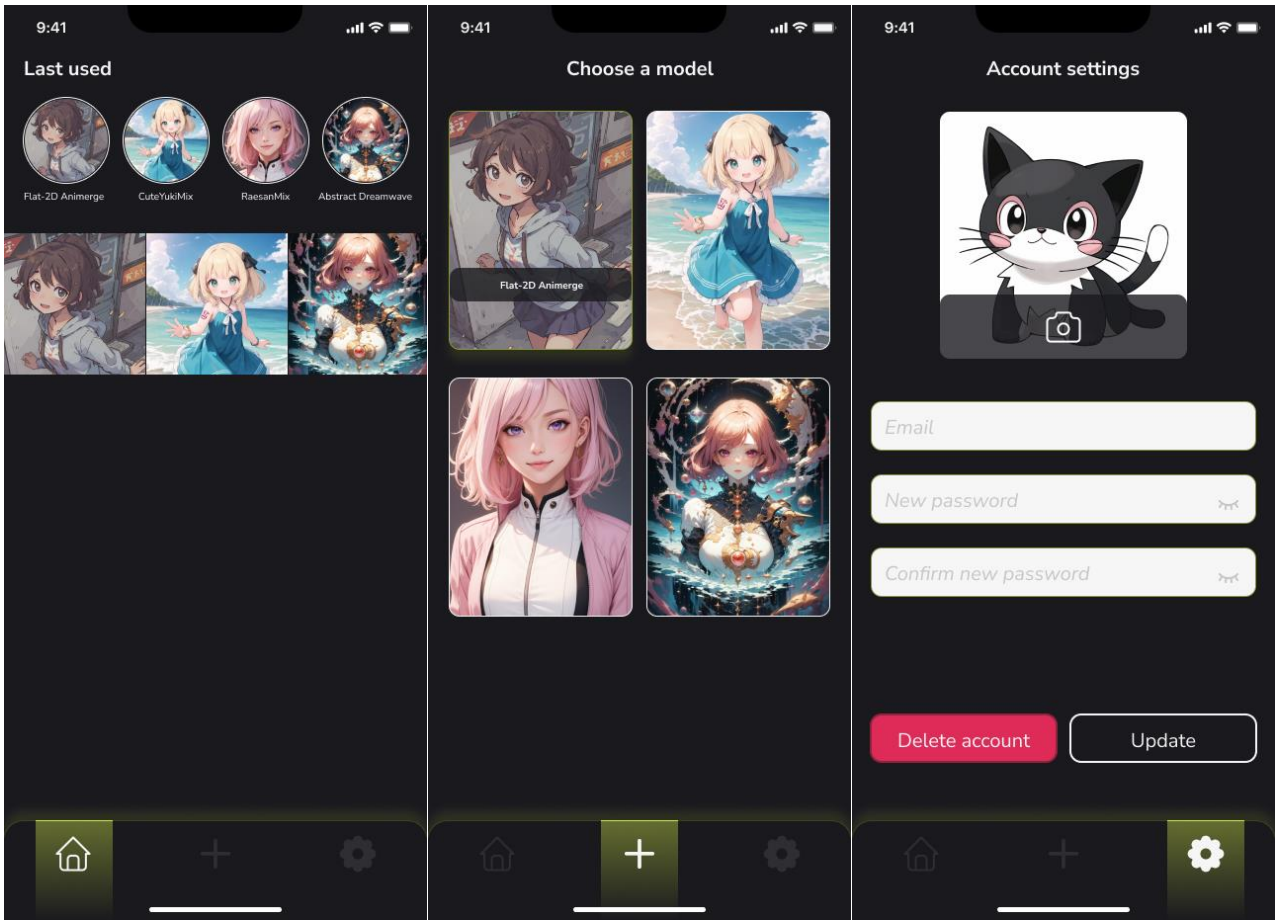
CODE 10. Delete post function in post repository, captured from project development.

4.3 Application screens



PICTURE 6. Prototype of welcome screen, login screen, and register screen in Figma, captured from project development

PICTURE 6 illustrates the screens that all users view when they initially launch the application. Login and registration are screens found in the “auth” feature directory. Anonymous users can only access their existing accounts or create new ones. A registered user cannot visit the login or register screens since they are instantly forwarded to the home screen. PICTURE 7 also displays the home screen, a new post screen, and a user settings screen that is only accessible to authorized users. There is a navigation bar at the bottom of three screens, with three icons representing each screen. The home panel displays user-saved posts as well as the most recently used post-creation models. The application provides some models of Stable Diffusion as art styles for creating new posts. Finally, the setting screen allows the authenticated user to modify their password, deactivate their account, and update their avatar.

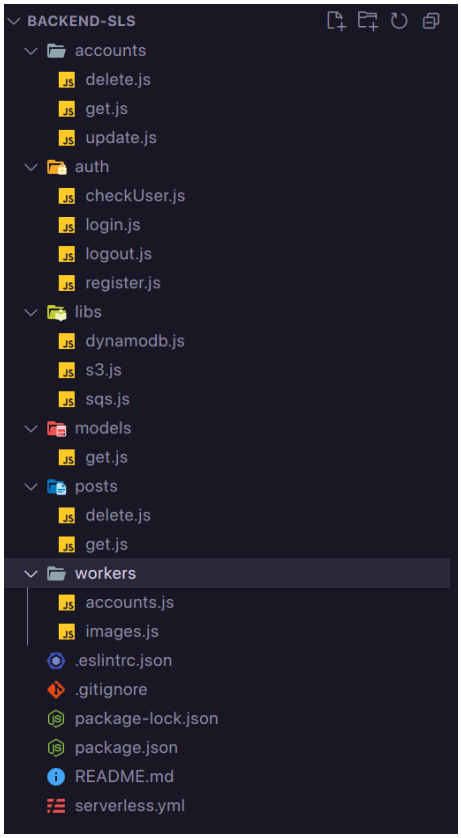


PICTURE 7. Prototype of authenticated user's home screen, create new post screen, and setting screen in Figma, captured from project development.

5 BACK-END IMPLEMENTATION

The final section details the application's back-end implementations, which rely on technologies discussed in previous sections, such as Amazon Web Services (AWS) and the Serverless framework. The first section of this chapter goes over the project structure, which covers the API endpoint organization and utility functions. The second section of this chapter describes how the back-end creates and updates information in AWS resources and event-driven architecture using "serverless.yml" file. It illustrates how user endpoints activate Lambda functions, which interact with DynamoDB tables and streams to store and update user information. The final section of this chapter shows how to transform uploaded photographs into "anime" art with AUTOMATIC1111 Stable Diffusion Web UI API. It demonstrates how the image endpoints trigger a Lambda function, which then calls an external API to apply the "anime" style to the image.

5.1 Project structure



PICTURE 8. Preview of back-end directory, captured from project development

The back-end is organized into these repositories: “accounts”, “libs”, “models”, “posts”, and “workers”. Each folder contains files that serve each separate purpose as shown in PICTURE 8. “Accounts” contains API handling for users’ account operation such as deleting, updating, and getting. “Libs” is the library directory where the common function is placed. It can be set up for AWS DynamoDB connection, or AWS S3 storage connection. “Models” contains the API handlers for get AI model from the client. In the front end, there is a tab called “Create Art” as the “plus” icon in the middle of the bottom navigation bar. When the users go to that page, the front end will fetch the available models from the back-end using models’ handler.

Posts means users’ generated image that have been saved in the database. “Posts” contains files that manage getting and deleting users’ post. Only authenticated users can get and view their posts. “Workers” contains background handlers for AWS SQS. When a new event comes to SQS, it will be handled by an assigned handler. For example, a new user created a new account for the services. There is a notification that user is created and a handler to download new avatar for that account will be triggered. CODE 11 shows four queues of AWS SQS that receive the event notifications and the handler for each event. As an example, “user-created-queue” is managed by “workers/accounts. createAvatar” which operates as the earlier explanation.

```
constructs:
  users-created-queue:
    type: queue
    worker:
      handler: workers/accounts.createAvatar
      fifo: true
      environment:
        BUCKET_NAME: ${construct:avatars.bucketName}
        AVATAR_API: ${self:custom.avatarApi}

  users-deleted-queue:
    type: queue
    worker:
      handler: workers/accounts.deleteAvatar
      fifo: true
      # batchSize: 10
      environment:
        BUCKET_NAME: ${construct:avatars.bucketName}

  images-processing-queue:
    type: queue
    worker:
      handler: workers/images.process
      fifo: true
      environment:
        RAW_BUCKET_NAME: ${construct:images-raw.bucketName}
        PROCESSED_BUCKET_NAME: ${construct:images-processed.bucketName}

  images-deleted-queue:
    type: queue
    worker:
      handler: workers/images.delete
      fifo: true
      # batchSize: 10
      environment:
        PROCESSED_BUCKET_NAME: ${construct:images-processed.bucketName}
```

CODE 11. AWS SQS queues definition in “serverless.yml” file, captured from project development

5.2 “serverless.yml” file

The “serverless.yml” file serves as the central hub for defining all aspects of the project’s back-end, surrounding its services, routes, and handlers. In this particular project, the file is organized into seven distinct sections: general information, “custom”, “provider”, “constructs”, “functions”, “plugins”, and “resources”. The general information section provides basic details about the project, such as the organization and application name within the Serverless platform. Developers can use the “custom” section to define global environment variables that can be utilized throughout the file. The “provider” section governs the AWS environment, including deployment stage, region, and security policies. The “constructs” section eases the creation of AWS resources like SQS queues and S3 storage. The “functions” section empowers developers to define API endpoints and their corresponding handlers. External plugins, such as “serverless-offline” and “serverless-lift”, are managed under the “plugins” section. Finally, the “resources” section serves as the repository for defining additional AWS resources such as DynamoDB.

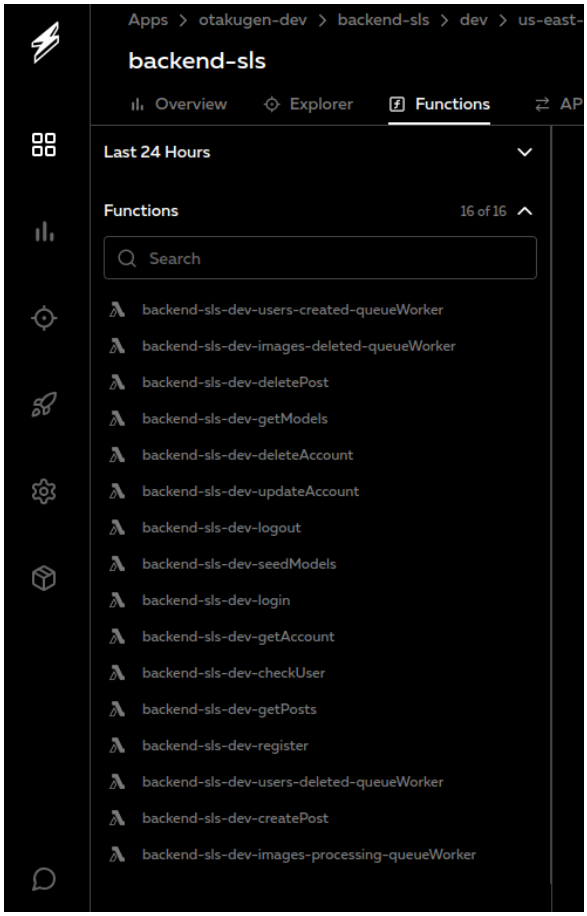


FIGURE 13. All API endpoints in Serverless Framework platform, captured from project development

As same the project structure, the API structure is organized into these main sections' "authentication", "accounts", "posts", and "models". FIGURE 13 shows all endpoints what have been used in the project in Serverless Framework platform while FIGURE 14 captured endpoints for accounts and posts in AWS API Gateway. All the endpoint is starting with "/api". "Authentication" routes go next with "auth" after the prefix. For example, "/api/auth/login" is used to handling login function using "POST" method. "Accounts", "posts", and "models" are used to handling users' accounts, posts, and AI models accordingly. In some definitions, there is a field named "environment". It is the environment variables for the AWS Lambda function. CODE 12 shows a piece of code where the register function consumes the variable "QUEUE_URL". "QUEUE_URL" is defined in "serverless.yml" file as shown in PICTURE.

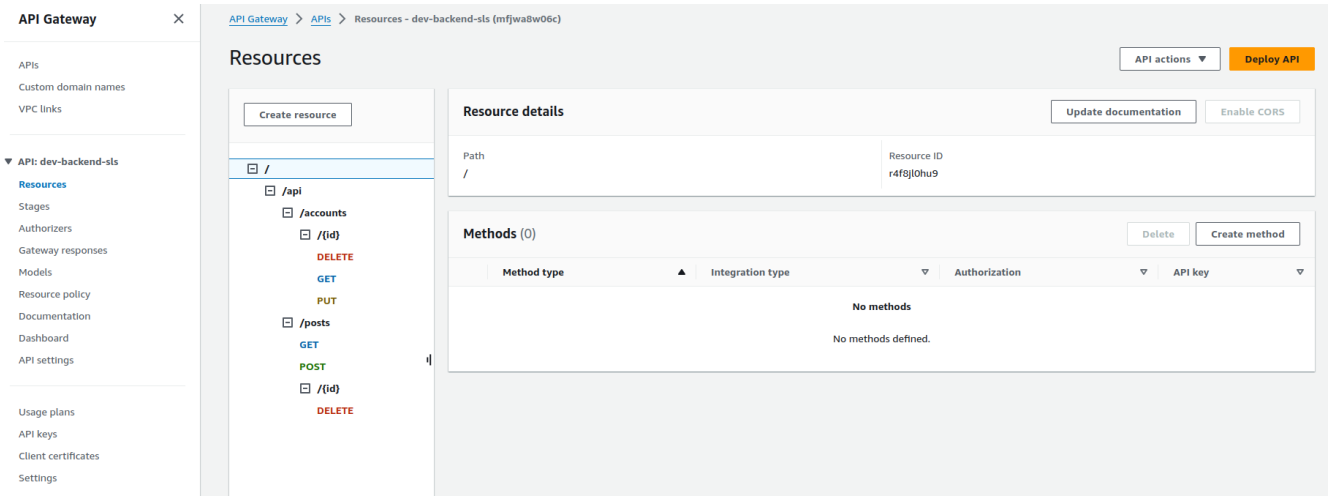


FIGURE 14. API endpoints for accounts and posts captured in AWS API Gateway, captured from project development

```
    // Send sqs message
    const queueCmd = new SendMessageCommand({
      QueueUrl: process.env.QUEUE_URL,
      MessageBody: JSON.stringify({
        id: userId,
      }),
    });

    const queueRes = await sqs.send(queueCmd);

    if (queueRes.$metadata.httpStatusCode !== 200) {
      callback(null, {
        statusCode: 500,
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          message: "Failed to created new avatar",
        }),
      });
      return;
    }

    callback(null, {
      statusCode: 200,
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        message: "User created successfully",
      }),
    });
  } catch (error) {
    console.error(error);
```

CODE 12. A piece of code for handling register function in “register.js” file, captured from project development

Each route could manage all HTML request methods such as “GET”, “POST”, “PUT”, “DELETE”, ... It is defined in “serverless.yml” as the field named “method” inside “events”. The back-end should send status codes to show if an HTTP request is done. There are five types of status codes. The first type “1xx” gives information. The second type “2xx” means success. The third type “3xx” redirects to another location. The fourth type “4xx” indicates a client error. The last type “5xx” signals a server error (Mozilla Developers 2022). CODE 13 shows an example for return status code 400 if the client is sending payload that missed email or password when a user tries to login. Status code “400” means “bad request”.

```
module.exports.handler = async (event, _context, callback) => {
  try {
    const data = JSON.parse(event.body);

    // Validate body
    if (!data.email || !data.password) {
      callback(null, {
        statusCode: 400,
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          message: "Missing email or password",
        }),
      });
      return;
    }

    // Check if email exists
    const scanCmd = new ScanCommand({
      TableName: tableName,
      FilterExpression: "email = :email",
      ExpressionAttributeValues: {
        ":email": { S: data.email },
      },
    });

    const scanRes = await dynamoDB.send(scanCmd);

    if (scanRes.Items.length > 0) {
      callback(null, {
        statusCode: 400,
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          message: "Email already exists",
        }),
      });
      return;
    }
```

CODE 13. A piece of code for return “Bad Request” status, captured from project development.

In “serverless.yml” file, there is a section to define all the resources in Amazon Web Services. CODE 13 shows resources listed inside “resources”. They are three tables which are used for saving users, posts, and models information. The type of database is AWS DynamoDB. The key to the schema is “id” which type is hash string. By defining a constant name at the top of the file in “custom” section, developer can pass it into multiple places as environment variables. In CODE 14, the name of user table can be assigned when using the syntax “${self:custom.userTableName}”.

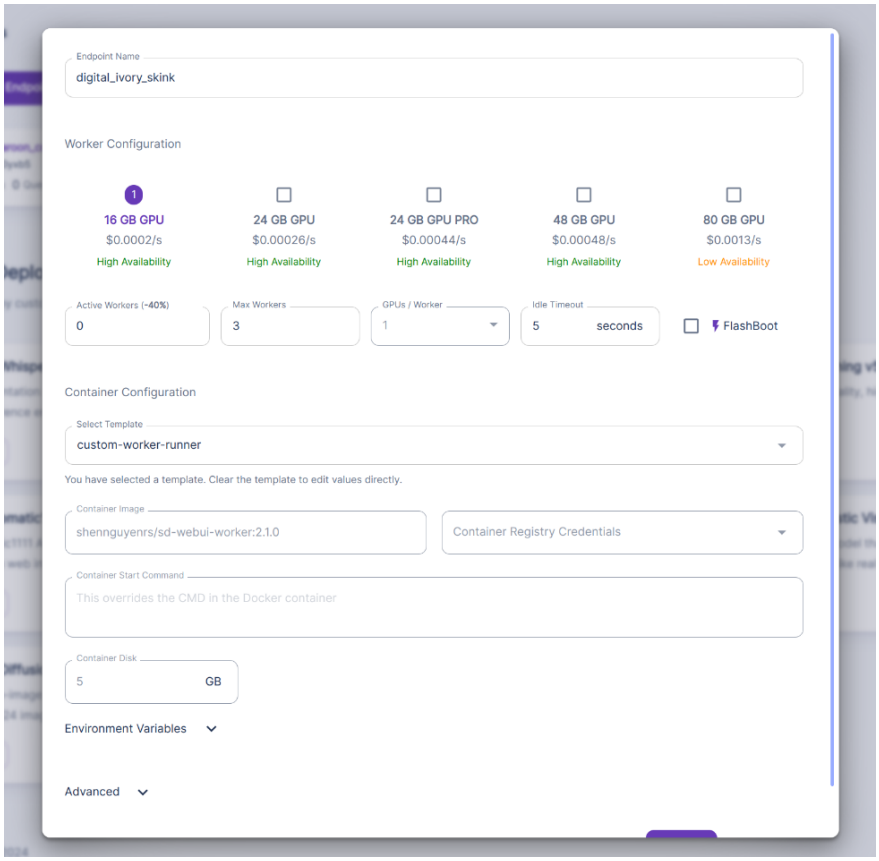
```yaml
resources:
  Resources:
    UsersTable:
      Type: AWS::DynamoDB::Table
      Properties:
        AttributeDefinitions:
          - AttributeName: id
            AttributeType: S
        KeySchema:
          - AttributeName: id
            KeyType: HASH
        BillingMode: PAY_PER_REQUEST
        TableName: ${self:custom.userTableName}

    PostsTable:
      Type: AWS::DynamoDB::Table
      Properties:
        AttributeDefinitions:
          - AttributeName: id
            AttributeType: S
        KeySchema:
          - AttributeName: id
            KeyType: HASH
        BillingMode: PAY_PER_REQUEST
        TableName: ${self:custom.postTableName}

    ModelsTable:
      Type: AWS::DynamoDB::Table
      Properties:
        AttributeDefinitions:
          - AttributeName: id
            AttributeType: S
        KeySchema:
          - AttributeName: id
            KeyType: HASH
        BillingMode: PAY_PER_REQUEST
        TableName: ${self:custom.modelTableName}
```

CODE 14. List of resources of Amazon Web Services inside “serverless.yaml”, captured from project development

5.3 Image generating progress

The conversion from photo to “anime” graphic style requires an external API. The API worker runs in a cloud environment that supports graphics processing units (GPUs). In this project, RunPod serves as the cloud provider for the AUTOMATIC1111 Stable Diffusion Web UI API. This setup offers greater scalability, cost-effectiveness, and ensures optimal performance for image manipulation tasks. PICTURE 9 demonstrates the user interface in RunPod for building an endpoint. In the image, three workers are allocated to handle the requests, and they employ 16 RAM GPUs. More workers can be provided by the user for manual scaling based on real-time demand, ensuring that resources are

efficiently allocated, and costs are optimized. Each worker requires cloud storage to contain the docker image and generated images.

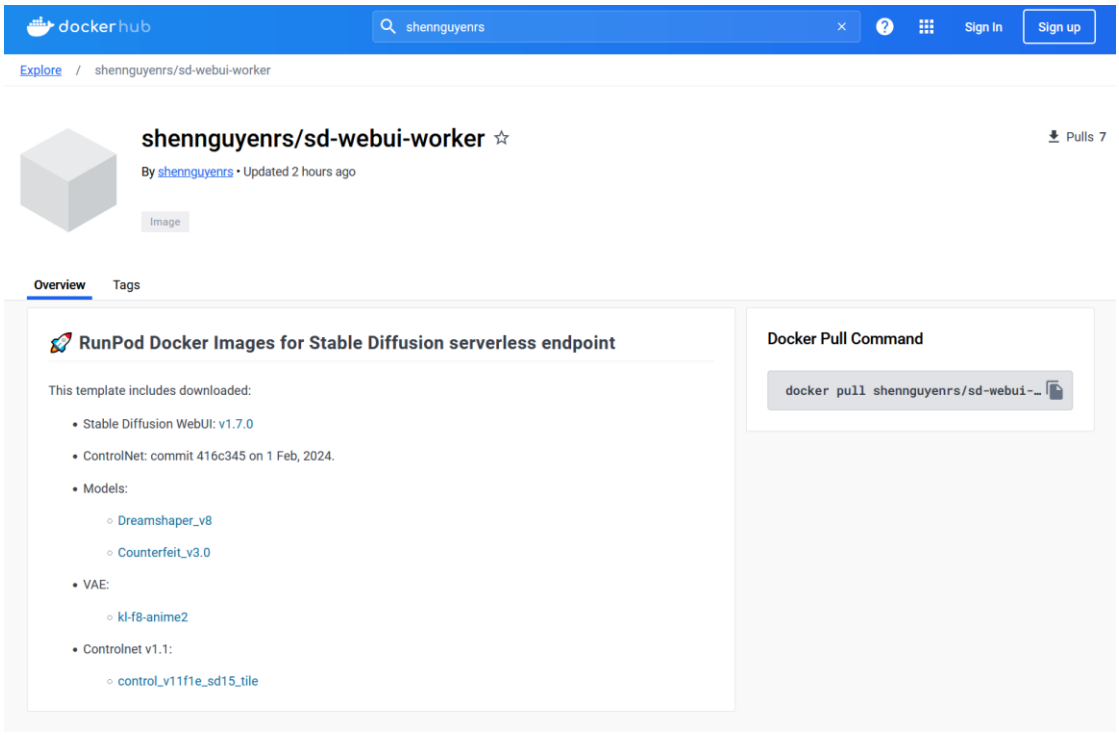


PICTURE 9. Create new endpoint API for Stable Diffusion models, captured from project development

Each worker runs a Docker container that has the image for the AUTOMATIC1111 Stable Diffusion Web UI API. This image uses the Python application server as a serverless function. It is designed to just deploy on the RunPod platform. PICTURE 35 shows that the Docker image is publicly available on Dockerhub. It also provides Stable Diffusion models, such as “Dreamshaper_v8” and “Counterfeit_V3.0”. Furthermore, it includes the “Controlnet v1.1” extension and the “Controlnet Tile” model, which assist in creating a new image taking the input photo as a reference. When building a new endpoint, the user can choose an image from the “Select Template” page.

The serverless function of Stable Diffusion API requires a payload of type PICTURE 10 from AWS services in order to generate images. The payload contains two critical components. The first is API method and endpoint, whereas the second is API payload. The payload specifies a “POST” call to the endpoint “/sdapi/v1/txt2img”, indicating that it is communicating with the “text2img” function in the

AUTOMATIC1111 Stable Diffusion Web UI. The first parameter in the payload is the model checkpoint, “dreamshaper_v8”. The prompt is “anime style, 2D,” which outlines the intended artistic style and serves as a guide for the final output. The negative prompt is blank, indicating there are no particular characteristics to exclude from the resulting image.



PICTURE 10. AUTOMATIC1111 Stable Diffusion Web UI API custom image for RunPod with build-in Stable Diffusion models, captured from project development

The next parameter is seed, and its value is “-1”. It enables the algorithm to generate a random seed used in image production. The batch size is set to 1, indicating that only one image should be generated. Denoising strength is used to adjust the amount of noise reduction in the resulting image. The number of iterations used to improve images is influenced by the steps. The “CFG Scale” parameter changes the output’s complexity and detail level. The picture dimensions specify the width and height of the resulting image. No matter what size the original image is, it will be cropped to “480x640” pixels. Sampler names have various effects on the output’s unpredictability and style. The sampler used in CODE 15 is “DPM++ 2M SDE”. The appropriate is suggested on the Stable Diffusion model page.

The next significant component in the payload is the “controlnet” parameters. The input picture is a “base64” encoded image string. Modules and models are required to analyze references and create new artwork. The user can give “controlnet” sufficient power to determine how much the reference will

influence the output by specifying the providing guidance, starting and stopping points. The maximum value for starting and ending advice is one. The values of 0 for “guidance_start” and 0.4 for “guidance_end” in the payload indicate that “controlnet” has to follow the reference image from the first step to 40% of the total steps. Then “controlnet” will allow the Stable Diffusion model to be more creative in response to the prompt in the next steps.

```
"input": {
    "api": {"method": "POST", "endpoint": "/sdapi/v1/txt2img"},
    "payload": {
        "override_settings": {
            "sd_model_checkpoint": "dreamshaper_v8",
        },
        "prompt": "anime style, 2d,",
        "negative_prompt": "",
        "seed": -1,
        "batch_size": 1,
        "denoising_strength": 0.75,
        "steps": 30,
        "cfg_scale": 7,
        "width": 480,
        "height": 640,
        "sampler_name": "DPM++ 2M SDE",
        "sampler_index": "DPM++ 2M SDE",
        "restore_faces": False,
        "alwayson_scripts": {
            "controlnet": {
                "args": [
                    {
                        "input_image": image_content,
                        "module": "tile_resample",
                        "model": "control_v11f1e_sd15_tile [a371b31b]",
                        "weight": 1,
                        "resize_mode": "Crop and Resize",
                        "lowvram": False,
                        "processor_res": 512,
                        "guidance": 1,
                        "guidance_start": 0,
                        "guidance_end": 0.4,
                        "control_mode": "Balanced",
                        "pixel_perfect": False,
                    }
                ]
            }
        },
    },
}
```

CODE 15. Sample payload for conversion from user’s photo into “anime” style, captured from project development

6 CONCLUSION

The thesis provides an overview of building full-stack solution for using AI model to transform photo. It also supplies knowledge about Flutter, Stable Diffusion, Amazon Web Services, Serverless frameworks, and event-driven architecture. There are various areas for improvement for future development, such as expanding the application's cross-platform compatibility, such as iOS, implementing internationalization and localization, and integrating with Google AdMob and Ad Manager advertisements. On the back-end, it could involve implementing an automation unit testing plan and learning best practices for increasing efficiency with the Serverless framework and AWS.

On the front-end, Flutter and Dart are aiming to make applications that work over various stages such as iOS, Windows, and web browsers, in expansion to Android. Be that as it may, the application was outlined and tried on an Android phone, restricting its usefulness for numerous stages. It would be great in case the extend may have been executed and evaluated on Apple devices. It, moreover, makes a difference in the application attracts more potential clients. To attain this, the application should be assessed and optimized for different screen sizes, resolutions, and working frameworks. It, moreover, ought to have planned rules and measures for each stage, such as utilizing local widgets and symbols.

The application has to be able to back various lingos and areas so that around the world clients can utilize the benefits in their neighborhood tongue. This would overhaul the client inclusion and fulfillment, as well as the engaging quality and competitiveness of the application. To attain this, the application has to utilize the Flutter internationalization and localization tools, such as the “intl” bundle and “flutter_localizations” library. It, moreover, should give interpretations for all the content and substance within the application, as well as the suitable designs for dates, numbers, monetary forms, and units. The application ought to be able to show significant and engaging advertisements, as a way to generate income from the work of the designers. This would permit the designers to supply high-quality versatile applications. To realize this, the application ought to utilize the Google AdMob and Ad Manager administrations, which offer different sorts of advertisements, such as standard, interstitial, local, and compensated advertisements. It, moreover, must take after the finest hones and arrangements for joining advertisements, such as choosing the proper advertisement arrangement, situation, and recurrence, as well as regarding the user's security and inclinations.

On the back-end, it may consolidate actualizing a mechanization unit testing course of activity and learning the finest hones for expanding effectiveness with the Serverless system and AWS. This would guarantee the quality and courageous quality of the back-end code, as well as the flexibility, execution, and take a toll optimization of the cloud organizations. Engineers might utilize third-party contraptions such as “AWS SAM CLI” or “localstack” to mimic and test the back-end locally. This would offer assistance to confirm the comfort and rightness of the code, as well as to recognize and settle any bugs or botches. The unit tests need to be run thus at some point as of late sending the code to the cloud, utilizing a ceaseless integration instrument, such as Travis CI or GitHub Actions. The back-end got to take after the driving sharpens for creating and passing on serverless frameworks with AWS assets. In any case, the case store of the framework has not been updated for 10 months. In the long final, the back-end needs to besides optimize the execution and take a toll on the serverless applications, by utilizing methods such as caching, throttling, clumping, and cold start expectation.

REFERENCES

AWS Developers. 2023a. *Amazon API gateway documentation - docs.aws.amazon.com*. Available at: https://docs.aws.amazon.com/apigateway/ Accessed: 27 October 2023.

AWS Developers. 2023b. *Lambda, Amazon*. Available at: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html Accessed: 27 October 2023.

AWS Developers. 2023c. *What is Amazon S3? - amazon simple storage service, What is Amazon S3?* Available at: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html Accessed: 27 October 2023.

AWS Developers. 2023d. *What is Amazon Cloudfront? - amazon cloudfront - docs.aws.amazon.com*. Available at: https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/Introduction.html Accessed: 12 November 2023.

AWS Developers. 2023e. *What is Amazon dynamodb? - amazon dynamodb - docs.aws.amazon.com*. Available at: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html Accessed: 12 November 2023.

AWS Developers. 2023f. *What is Amazon Simple Queue Service?* Available at: https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/welcome.html Accessed: 27 October 2023.

AWS Developers. 2023g. *Whitepapers, Amazon*. Available at: https://docs.aws.amazon.com/whitepapers/latest/aws-overview/introduction.html Accessed: 27 October 2023.

Bizzotto, A. 2022. *Flutter project structure: Feature-first or layer-first?* Available at: https://codewithandrea.com/articles/flutter-project-structure/ Accessed 29 December 2023.

Bizzotto, A. 2023. *How to fetch data and perform data mutations with the riverpod architecture*. Available at: https://codewithandrea.com/articles/data-mutations-riverpod/ Accessed 30 December 2023.

Brownlee, J. 2019a. *A gentle introduction to generative adversarial networks (Gans), MachineLearningMastery.com*. Available at: https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/ Accessed: 26 October 2023.

Brownlee, J. 2019b. *What is natural language processing?, MachineLearningMastery.com*. Available at: https://machinelearningmastery.com/natural-language-processing/ Accessed: 26 October 2023.

Bushwick, S. 2023. *See how AI generates images from text, Scientific American*. Available at: https://www.scientificamerican.com/article/see-how-ai-generates-images-from-text/ Accessed: 25 October 2023.

Chui, M., Hazan, E., Roberts, R., Singla, A., Smaje, K., Sukharevsky, A., Yee, L., Zemmel, R. 2023. *The economic potential of Generative AI: The Next Productivity Frontier, McKinsey & Company*.

Available at: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-economic-potential-of-generative-ai-the-next-productivity-frontier#business-and-society Accessed: 22 December 2023.

Dart Developers. 2023. *Dart overview, Dart*. Available at: https://dart.dev/overview Accessed: 25 October 2023.

Flutter Developers. 2023. *Flutter documentation, Flutter*. Available at: https://docs.flutter.dev/ Accessed: 25 October 2023.

Laukkonen, J. 2023. *What is stable diffusion? A look at how one artificial intelligence model is re-shaping the images you see, Lifewire.* Available at: https://www.lifewire.com/what-is-stable-diffusion-7485593 Accessed: 25 October 2023.

Mozilla Developers. 2022. *HTTP response status codes - HTTP: MDN. HTTP | MDN*. Available at: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status Accessed 14 December 2023.

Nicoletti, L. and Bass, D. 2023. *Humans are biased*. Available at: https://www.bloomberg.com/graphics/2023-generative-ai-bias/ Accessed: 26 October 2023.

Hajian, M. 2021. *Introduction to using Dart in flutter, LogRocket Blog*. Available at: https://blog.logrocket.com/introduction-to-using-dart-in-flutter/ Accessed: 17 October 2023.

Kaur, D. 2021. *Is tiktok a threat to facebook social media dominance?, TechHQ*. Available at: https://techhq.com/2021/03/is-tiktok-a-threat-to-facebook-social-media-dominance/ Accessed: 17 October 2023.

Nguyen, T. 2023. *Flutter vs react native: The ultimate comparison (2023), Frontend Mag*. Available at: https://www.frontendmag.com/insights/flutter-vs-react-native/ Accessed: 25 October 2023.

Raj, G. 2023. *What is CFG scale in stable diffusion and how to use it.* Available at: https://decentralizedcreator.com/cfg-scale-in-stable-diffusion-and-how-to-use-it/ Accessed 28 December 2023.

RedHat.com Editors. 2019. *What is event-driven architecture*. Available at: https://www.redhat.com/en/topics/integration/what-is-event-driven-architecture Accessed: 12 November 2023.

Riverpod Developers. 2023. *Why Riverpod?* Available at: https://riverpod.dev/docs/introduction/why_riverpod Accessed 30 December 2023.

Roemerman, S. 2019 *Council Post: A handy guide to the differences between edge, fog and cloud computing, Forbes.* Available at: https://www.forbes.com/sites/forbestechcouncil/2019/07/15/a-handy-guide-to-the-differences-between-edge-fog-and-cloud-computing/ Accessed: 27 October 2023.

Sanity Developers. 2023. *What is flutter? - framework Overview / intro - glossary, Sanity.io*. Available at: https://www.sanity.io/glossary/flutter Accessed: 25 October 2023.

Serverless Framework Developers. 2023. *Serverless Framework Concept*s. Available at: https://www.serverless.com/framework/docs/providers/aws/guide/intro/ Accessed: 12 November 2023.

Tu, A.H. 2023. *Mastering the automatic1111 user interface: A comprehensive guide for stable diffusion*. Available at: https://www.andyhtu.com/post/mastering-the-automatic1111-user-interface-a-comprehensive-guide#viewer-7sgiq Accessed 28 December 2023.

Vahdat, A. and Kreis, K. 2022. *Improving diffusion models as an alternative to Gans, part 1, NVIDIA Technical Blog*. Available at: https://developer.nvidia.com/blog/improving-diffusion-models-as-an-alternative-to-gans-part-1/ Accessed: 26 October 2023.

Weatherbed, J. 2022. *Artstation is hiding images protesting AI art on the platform, The Verge*. Available at: https://www.theverge.com/2022/12/23/23523864/artstation-removing-anti-ai-protest-artwork-censorship Accessed: 26 October 2023.

Yosifova, A. 2023. *The evolution of CHATGPT: History and future, 365 Data Science*. Available at: https://365datascience.com/trending/the-evolution-of-chatgpt-history-and-future/ Accessed: 17 October 2023.