TextPy

By Webcraft

 Elliot Fayman, Christina Mourad, Ryan Kao

So What is TextPy?


TextPy is an **innovative** messaging application that utilizes SMS messaging to transmit discrete



packets of Pythonic code between two microservices, a Flask backend and a Flutter frontend.



These packets are **compiled** and **interpreted** before being sent back as output, resulting in a

seamless and *efficient* communication process.

Some Background...

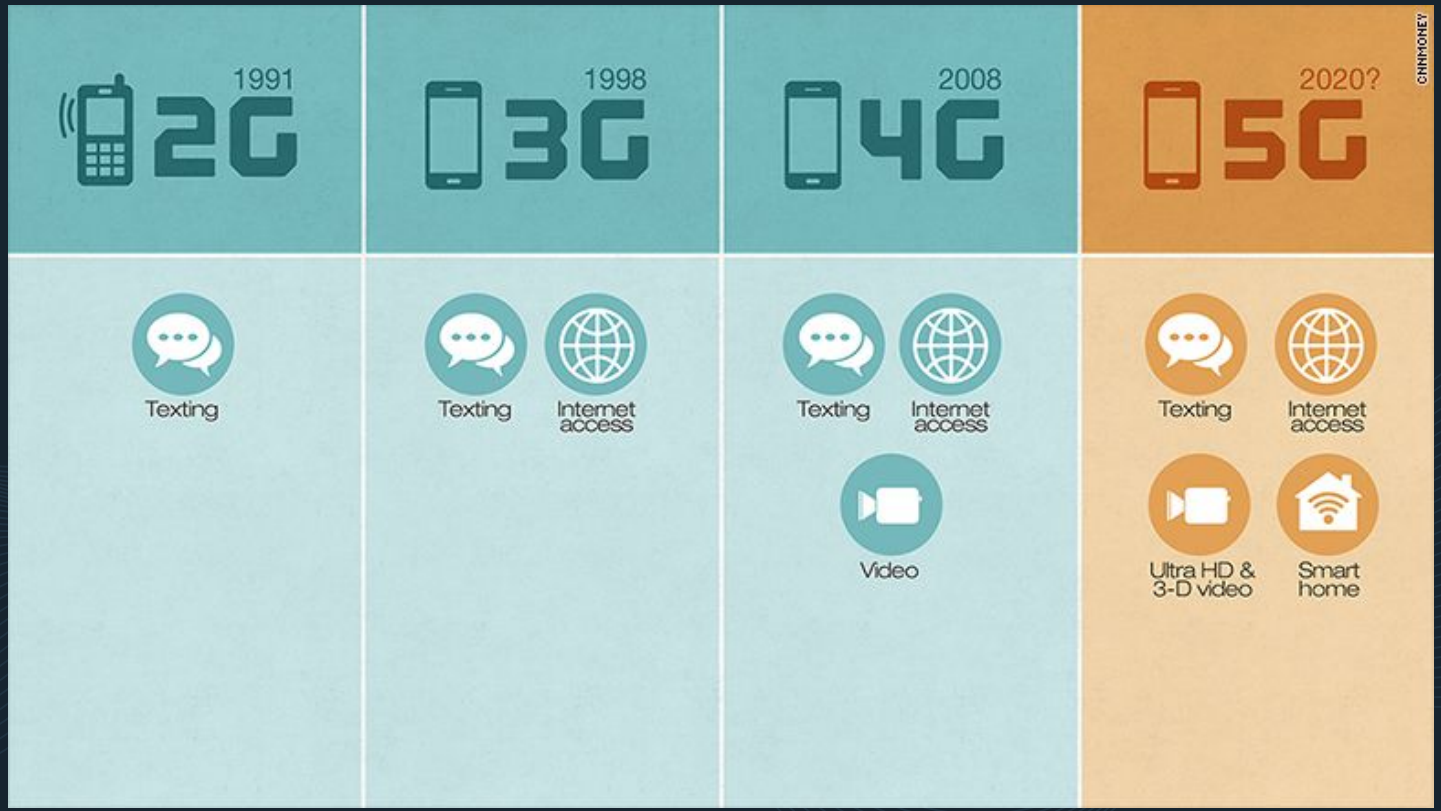
IP Based Communication

Any type of communication that uses the Internet Protocol (IP) to transmit data over the internet, including email, chat, VoIP (voice over IP), and video conferencing. Typically requires a reliable internet connection, which is often provided through a wired or wireless network, such as Wi-Fi.

SMS Based Communication

A type of mobile messaging that uses the Short Message Service (SMS) protocol to send and receive text messages over cellular networks. SMS messaging is widely used due to its high reliability and near-universal availability across mobile devices.

Cellular Networks



So Why TextPy...

Discrete Messaging

Independent of
Internet Connectivity

Near-Universal
Availability

Our Tech Stack

Dart

twilio

GitHub

Flask

Flutter

Jira

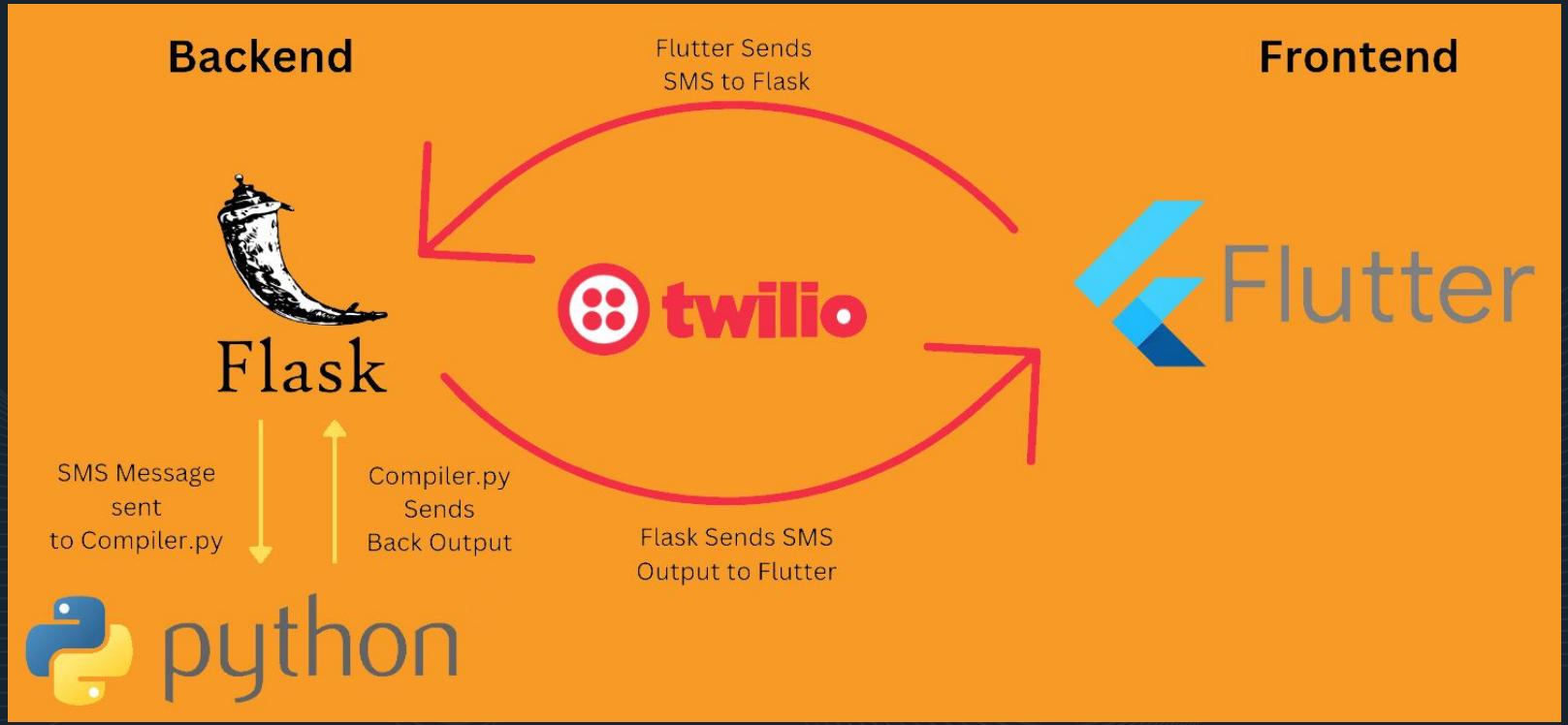
Requirements

- Create a Messaging platform
- Create a Mobile App User Interface
- Physically Separate Frontend & Backend
- Construct Infrastructure for Supporting complex message formats
- Develop an Intuitive UI
- Transmit and Receive Data via SMS
- Interpret Pythonic Code
- Create a Easily Scalable Product
- Minimize Software Bottlenecking
- Build Support for conventional internet based communication

Overall Design 

- Code Base Broken up into two microservices: front and backend
- Front and backend communicate via API that facilitates SMS messaging
- The front end handles the main UI while backend receives and compiles code

Design Model



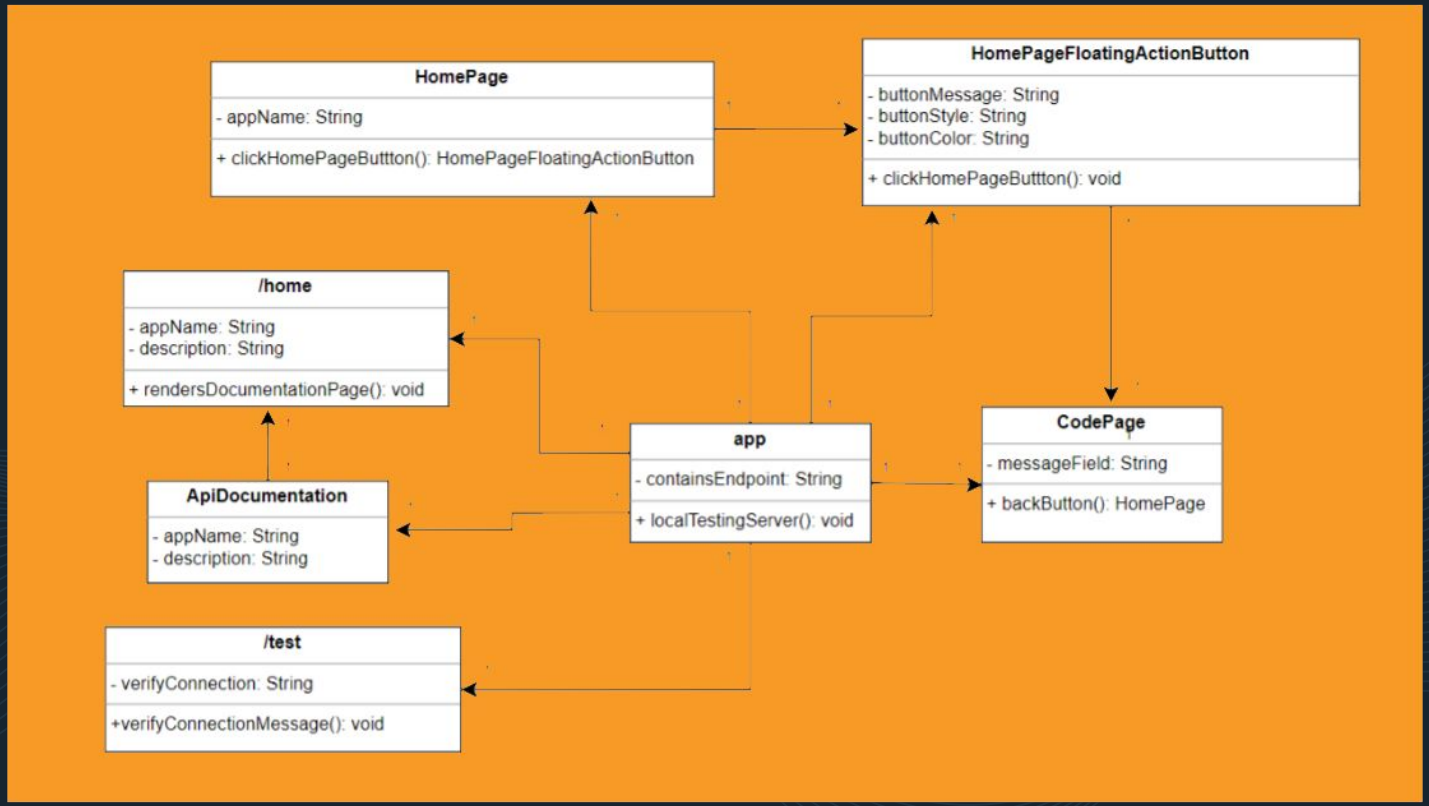
Design Patterns 

- Object Oriented Program - Modular, Reusable, well-organized, and better understood.

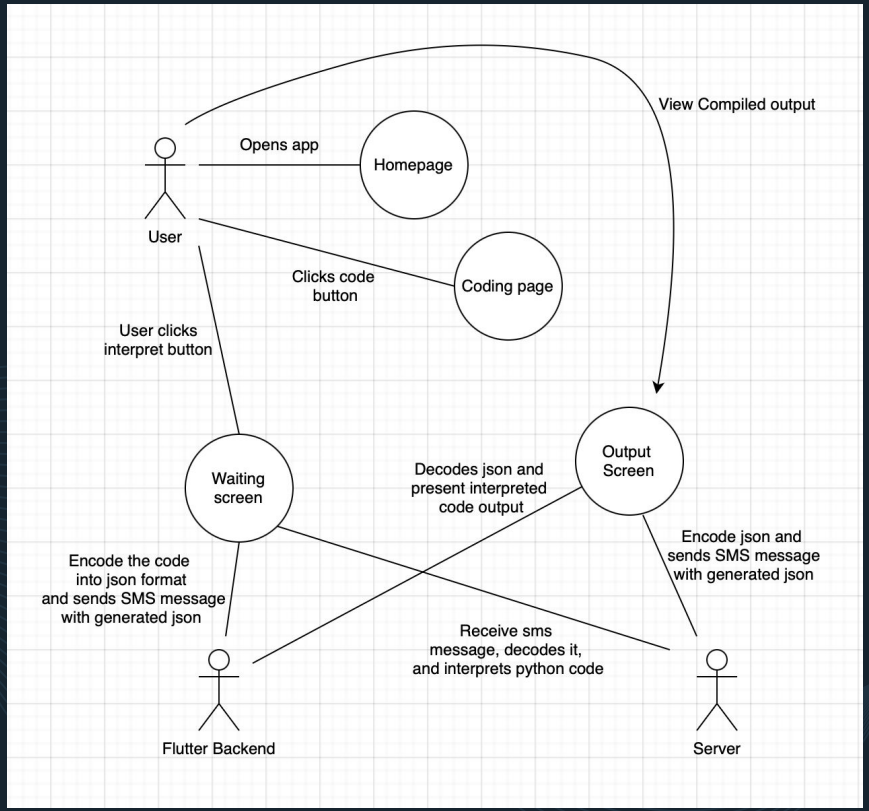
- Model, View, & Controller - Clear separation of easily readable parts.

- Restful API - Scalable, Flexible, and well documented for our use case.

UML Class Diagram



UML Use Case Diagram



Flask Backend

Backend

- **Flask:**

Flask is a micro web framework written in Python. It is designed to be simple, lightweight, and modular, with a small core that can be extended as needed with various third-party libraries. Flask is ideal for building small to medium-sized web applications and RESTful APIs.

- **Python:**

Python is a high-level, general-purpose programming language that emphasizes code readability and simplicity. It is widely used in various fields such as web development, data science, artificial intelligence, and automation.

Composition of Flask App

Main.py

- Contains main API functionality
- Has several endpoints
- Interacts with the front end directly

Compiler.py

- Contains two instance variables: code & output
- Each instance variable has getters and setters
- Compile method employs subprocess library in order to interpret python code

Flask Application Endpoints

[get] / - Renders html page detailing api details and use case

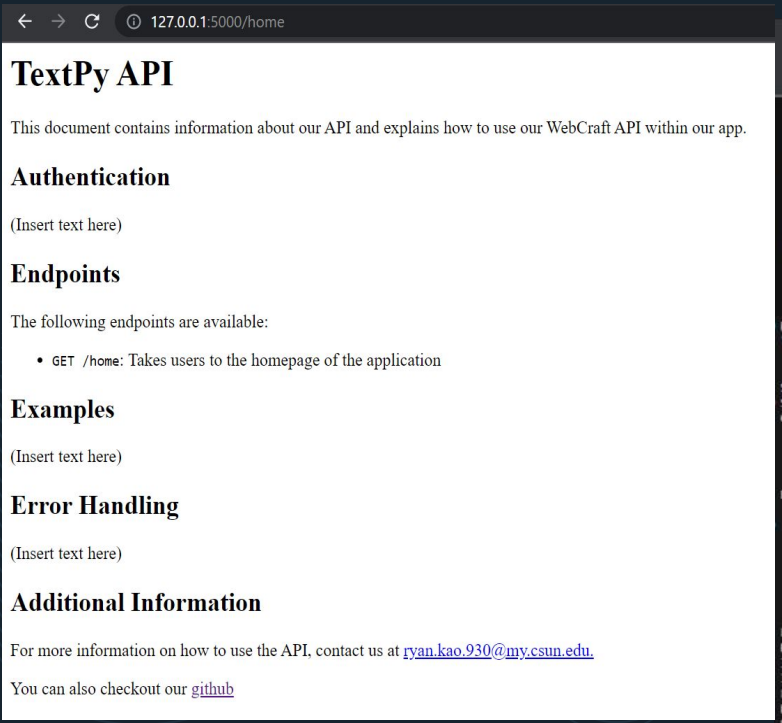
[get] /home - Same functionality as ‘/’

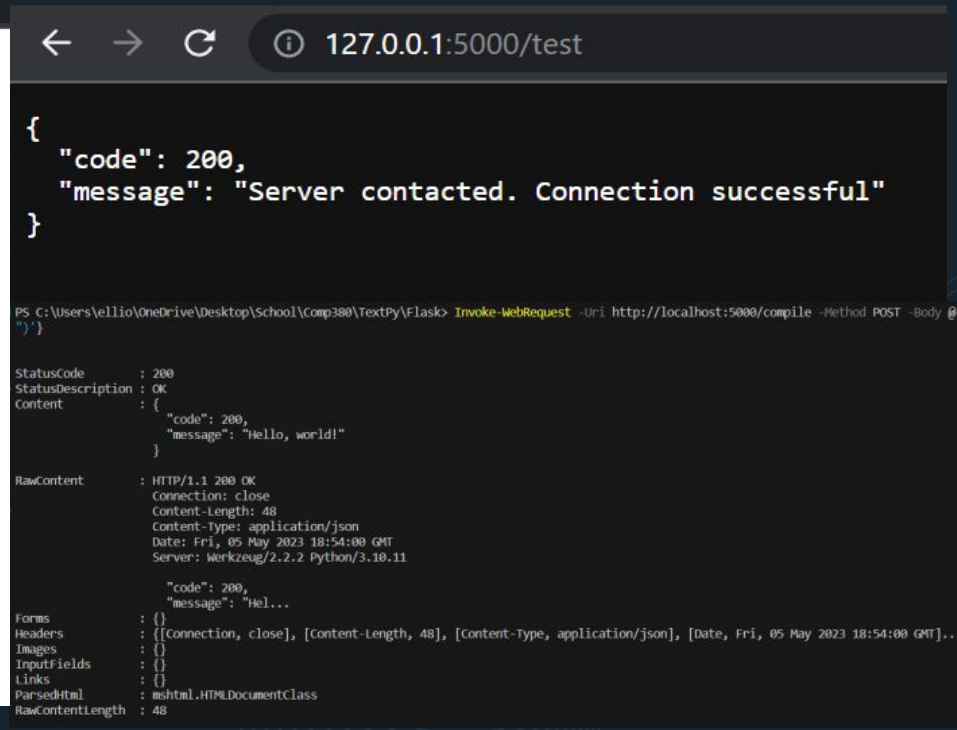
[get] /test - Used to ping api in order to test for a successful connection

[post] /compile - Compiles code directly without sms

[post] /sms - connected to a twilio webhook which automatically calls endpoint when sms message is received

Endpoint Sample Uses





Flutter Frontend

Front End

- Flutter:

Flutter is a Google-created open-source mobile application development framework that enables developers to build high-performance apps for mobile, web, and desktop with a single codebase.

- Dart:

Dart is an object oriented programming language used for building high-performance web,mobile, and desktop applications with features such as classes, inheritance, and encapsulation.

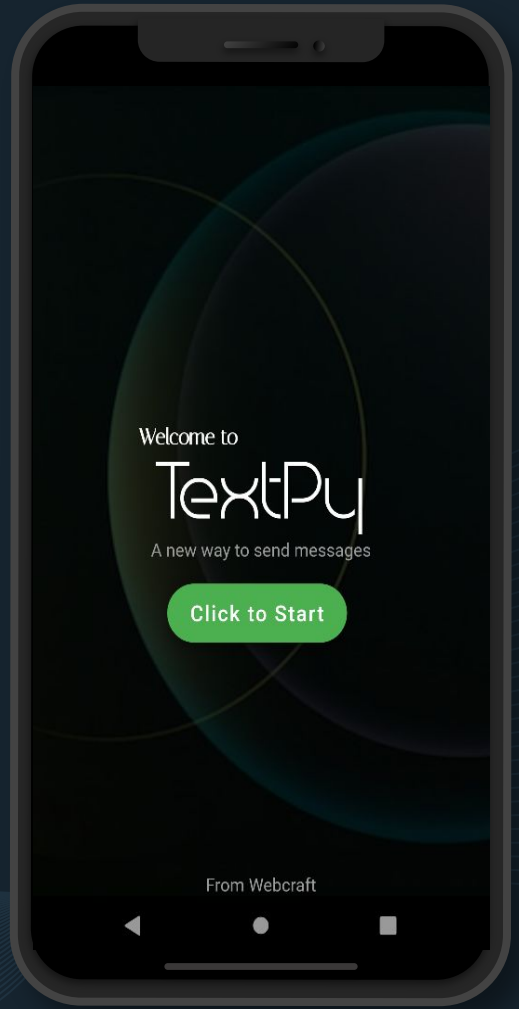
Flutter uses the Dart programming language and provides a set of pre-built widgets and tools for creating responsive user interfaces.

GUI - HomeScreen

HomeScreen:

- Displays a background image with an overlay of several text widgets arranged using a stack widget.

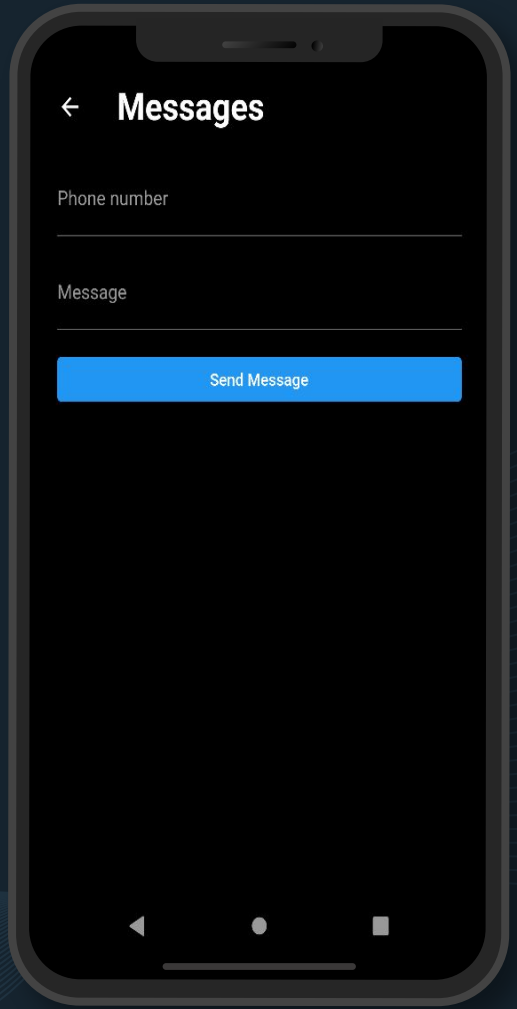
- Contains a *FloatingActionButton* widget that uses the *onPressed* property and *Navigator* class



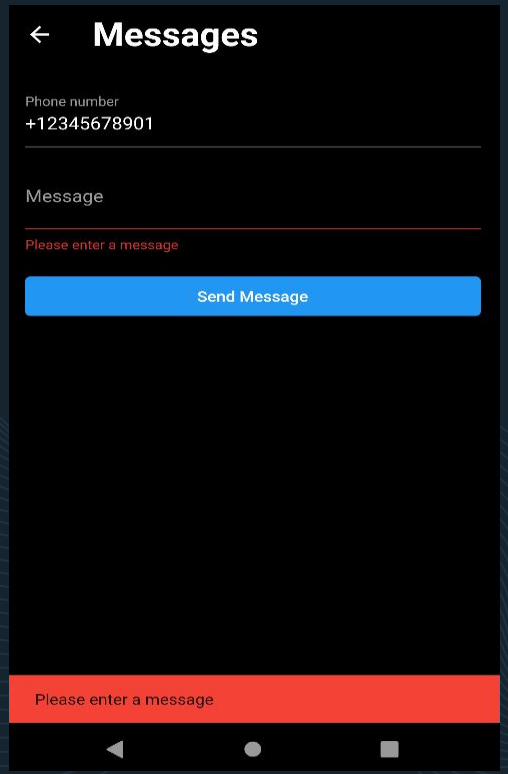
GUI - SecondPage

SecondPage:

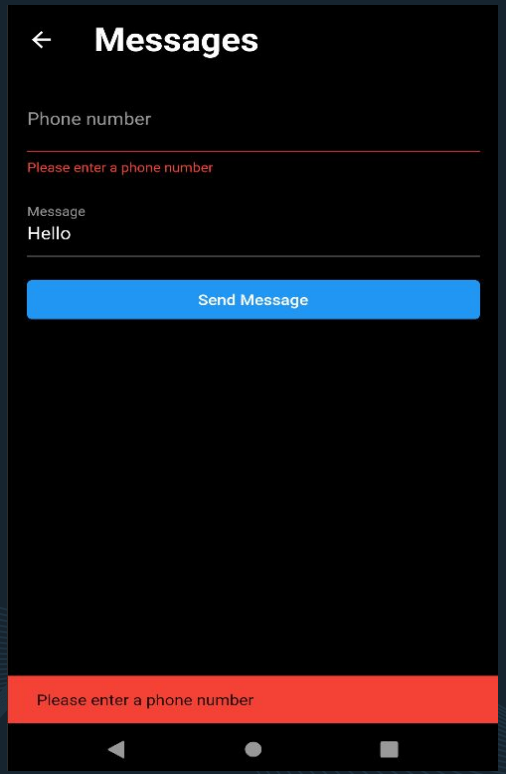
- Phone number TextField widget
- Message TextField widget
- Contains a ElevatedButton widget that utilizes the onPressed property that calls the function errorChecker



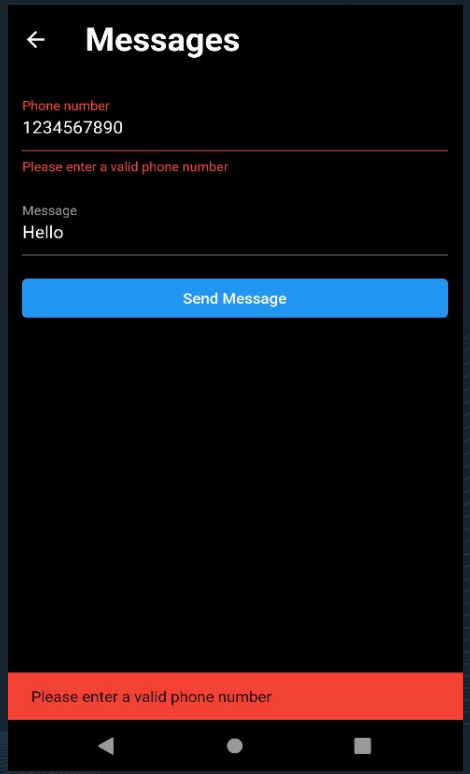
GUI - Errors



Message is empty



Phone Number is empty

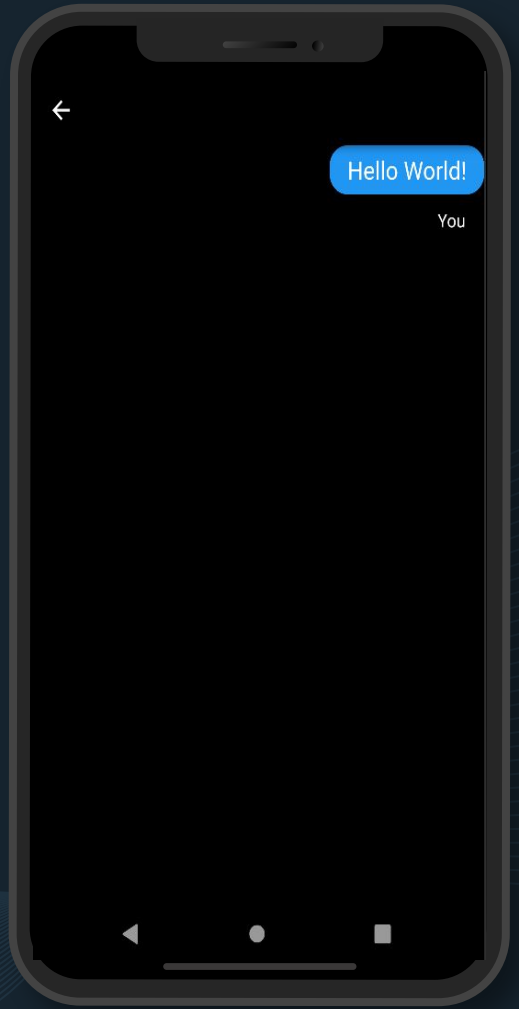


Invalid Phone Number

GUI - ChatPage

ChatPage:

- Displays the user's message through the ListView.builder widget



Unit Testing

Combination of automated testing and validation testing

- Used unittest library in python for automated testing of Flask application

```
PS C:\Users\ellio\OneDrive\Desktop\School\Comp380\TextPy\Flask> python -m unittest
.....................
----------------------------------------------------------------------
Ran 22 tests in 1.763s

OK
PS C:\Users\ellio\OneDrive\Desktop\School\Comp380\TextPy\Flask>
```

- Validation tested flutter UI

Documentation

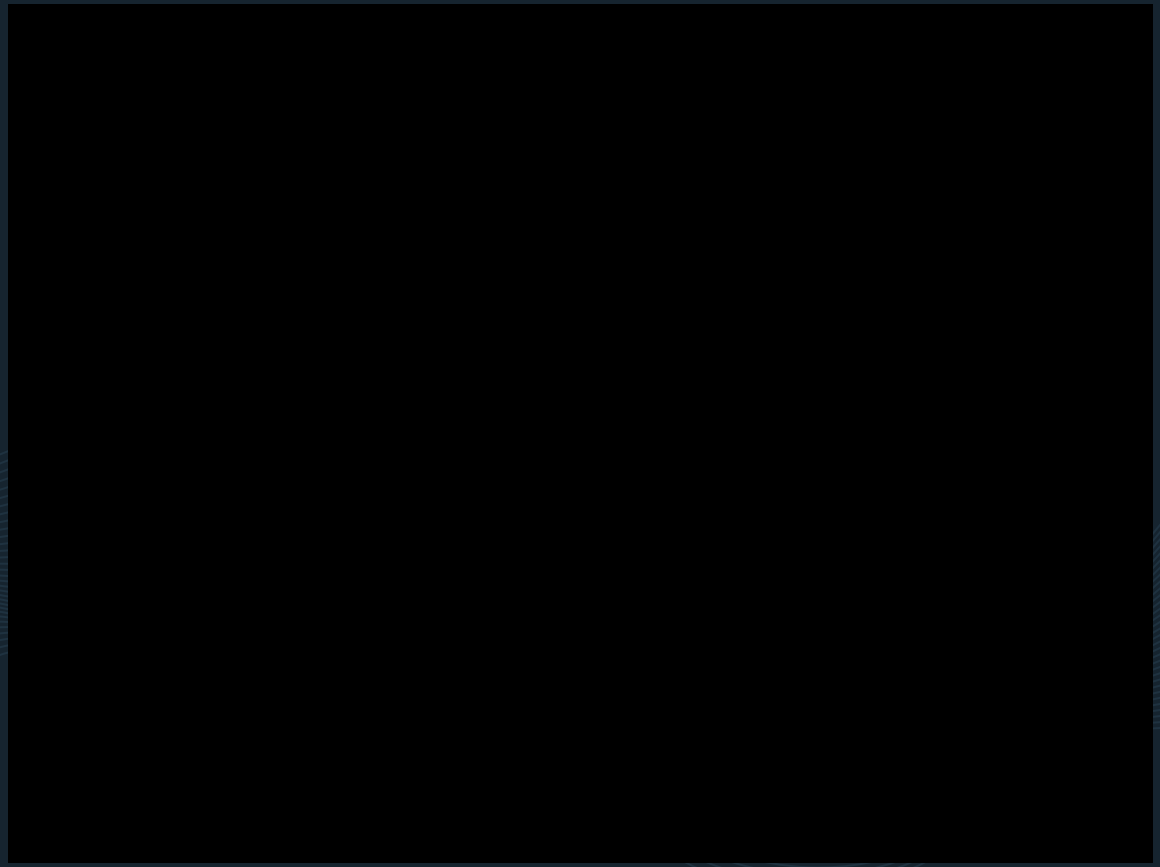
Flask documentation: pydoc

PyDOC

Flutter Documentation: dartdoc

Our Code Base...

End-to-end Demonstration



API Demonstration

What we have learned

- Integrating SMS messaging into our app
- Creating microservice applications
- Working in a scrum agile project (agile development environment)
- Documenting our code and writing Unit Tests

Successes/Challenges

Accomplishments:

- Developing two independent microservices that effectively communicate via SMS
- Able to test and compile our code thoroughly with minimal refactoring

Challenges:

- Integrating two independent microservices
- Collaboration on the same microservice due to version control issues
- Utilizing Twilio's messaging platform

Future plans

- Enhance the app by adding additional interfaces, such as a web app.
- Expand the app's functionality by adding additional infrastructure to send and receive more complex data packets.
- Improve the user experience by incorporating additional quality of life features to the app
- Optimize the app's performance by integrating a database to store server-side user information

THE END

Any Questions?