Phase 3: Design Patterns and Implementation Phase

Note: *It is expected that all members of the group will make a reasonable contribution in writing parts of this report, that way one group member does not get overstretched. Furthermore, it ensures all members get practice of writing technical reports.*

Please submit 1, 2, 3.1, 3.3 below as a single MS word/PDF document so that it is easier for me to provide comments.
Submit code in 3.2 as a zipped file.
Submit JAVAdoc/PyDOC/etc. style documentation as another zip file

Implementation is a process of translating the Design into a programming source code and generating an executable code. Hence, you will have to map the modules of the design phase or Classes into procedures/functions and writing the algorithms in the source code. Also, if necessary, the Spec and RA must be modified to reflect the changes in your design.

1. Introduction

1.1. Purpose

Put here why you wrote this documents including when, who was involved and contributed what - *contributions of each member listed individually to this phase of project and to this report, etc.*

Purpose: *The Design Patterns and Implementation Phase document provides an overview of the progress made in our project, highlighting the design patterns incorporated, essential files utilized, a summary of the different source codes within our project, and an assessment of contributions amongst our team members.*

Members: *Elliot Fayman, Ryan Kao, Christina Mourad*
Contribution:

Elliot*:* My contribution was many towards the interpreter for interpreting the python code that is sent via the SMS. I created the compiler class and extensively tested the compiler class. I also assisted in the creation of the Flask API.
Ryan*:* My contribution to this project was working on the backend of the application which is the API. I also participated in writing several parts of the report.
Christina: My contribution to this phase was finalizing the Flutter UI that enables the user to input and send text messages. Worked on incorporating various widgets to enhance the functionality and technical aspects of the application.

1.2 Definitions, Acronyms or Abbreviations

For this project, we used Flutter which is a mobile app development framework that allows developers to build natively compiled applications. Flutter also uses the Dart programming language which provides a rich set of customizable widgets and tools to help create the UI. MVC is a design pattern that is short for Model, View and Controller.

1.3 References

Any references used in this document

- ***Jira***
    - *https://webcraft.atlassian.net/jira/software/projects/TEX/boards/1*

- ***GitHub***
    - *https://github.com/elliotfayman/TextPy*

1.4 Overview

This project helped us become familiar with how working as a team in a workplace would feel like, such as having to communicate and schedule weekly meetings, splitting the work into equal parts, and meeting deadlines. For creating the messenger app, we had to create a home screen for users to access the app and implement codes that allow the user to send an SMS message to other users. This project consists of multiple programming languages and an API. Flutter and Dart was used for the front end of the app while the backend was created with Python, Flask, and Twilio.

2. Design Patterns

- *Model-View-Controller (MVC) pattern:*
  The Flask web framework follows the MVC architectural pattern, where the application logic is divided into three interconnected components - the Model, View, and Controller. The Flask application's routes are defined in the controller, which is responsible for handling incoming requests and returning responses. The homepage() function represents a controller, which returns the rendered HTML template as a view. The templates are used to represent the presentation layer, which is the responsibility of the View component.

- *RESTful API design pattern:*

Flask-RESTful is an extension for Flask that makes it easy to build RESTful APIs. The API object is created from the Flask app instance and used to add the API resources. The test() function is a Flask-RESTful Resource that returns a JSON response to the client. RESTful design patterns follow a set of conventions to build APIs that are scalable, easy to understand, and maintainable.

- *Factory pattern:*
  The Flask application instance is created using a factory function Flask(__name__). The factory pattern is a creational design pattern that provides an interface for creating objects, but it allows subclasses to alter the type of objects that will be created.

- *Singleton pattern:*
  The API instance is created as a singleton object, so there is only one instance of the object shared throughout the application. The Flask application uses the singleton pattern to create and maintain a single instance of the API object, which is used to register RESTful resources.

  Stateless Widget pattern:
  The 'HomeScreen', 'SecondPage', and 'Stories' classes are part of the stateless widget pattern. These classes are immutable and cannot change their internal state once they are instantiated.

- *Builder Pattern:*
  The 'MaterialPageRoute' class uses the builder pattern. The 'MaterialPageRoute' class creates a new page route by building a widget. It does this by calling the 'builder' method, which returns a widget. The 'MaterialPageRoute' then uses this widget as the content of the page route.

- *Encapsulation:*
  Encapsulation is implemented in the Compiler class to hide the implementation details of the code and output instance variables and provide public methods to access and manipulate them. This ensures that the class is self-contained, and that the code and output data are protected from unwanted modifications or access.

- *Single Responsibility:*
  The Compiler class has a single responsibility of compiling code, and any changes to its implementation will only affect this functionality. This ensures that the class is focused, understandable, and maintainable, and that it adheres to the "Single Responsibility Principle" of good software design.

- *Dependency Injection:*

Dependency Injection is implemented in the Compiler class to inject the code to compile via the set_code() method. This allows the code to be decoupled from the class and makes the class more reusable and flexible.

- *Command:*
  The compile() method uses the subprocess.run() function to execute the Python interpreter with the code to compile as a command.

  Microservices:
  The application uses a microservice design pattern using two different miniature applications, the flask API and the flutter mobile application. These two microservices communicate via API calls.

3. Technical Documentation and Source Code

3.1 Important files

Compiler.py
The Compiler class in this Flask application provides the functionality to compile and execute Python code. It is initialized with no arguments and contains three methods: set_code(), compile(), and get_output().

- set_code(code: str): sets the code that is to be compiled and executed. It takes a string code as input and does not return any value.
- compile(): compiles the code set by set_code() and executes it. It does not take any input and does not return any value.
- get_output() -> str: returns the output of the compiled code. It takes no input and returns a string.

Main.py
main.py is the main script of the Flask application. It initializes the Flask app object, the Compiler object, and the Twilio Client object. It defines four routes: /, /home, /test, /compile, and /sms.

- The / and /home routes render the home page template.
- The /test route tests the connection to the server and returns a JSON response indicating whether the connection was successful or not.
- The /compile route receives a POST request with Python code as the value of the code parameter, compiles and executes it using the Compiler class, and returns the output as a JSON response.
- The /sms route receives an SMS message, compiles the code in the message using the Compiler class, and sends the output back to the same phone number using the Twilio Client object.

Twilio
Twilio is a cloud communication platform that provides a suite of APIs for building SMS, voice, and messaging applications. In this Flask application, the Client class from the twilio.rest module is used to send and receive SMS messages. The twilio_phone_number variable holds the phone number associated with the Twilio account, which is used to receive incoming SMS messages. The Client object is initialized with the Twilio account SID and auth token, which are provided by Twilio upon account creation.

Main.dart
main.dart is a Flutter application that creates a user interface with two screens. The first screen, HomeScreen, displays an image with two text elements, and a button that navigates to the second screen, SecondPage. The second screen, SecondPage, displays a list of text elements, a stories card, and a button that navigates to the third screen, ChatPage. The ChatPage screen displays a list of messages and a text input field to send messages.

3.2 Source Code with documentation

Link to Flask Source Code: TextPy/Flask at main · elliotfayman/TextPy (github.com)
Link to Flutter Source Code: TextPy/Flutter/webcraft_app at main · elliotfayman/TextPy (github.com)
Link to Flask Documentation: TextPy/Flask/doc at main · elliotfayman/TextPy (github.com)

a) Module/Class name: Compiler
b) Date of the code: 4/18/2023
c) Programmer's name: Elliot Fayman
d) Brief description of the class/module: This class represents a code compiler that compiles Python code using a Python interpreter.
e) Brief explanation of important functions in each class, including its input values and output values:

- ***init(self, code):*** Initializes the Compiler object with the provided code. It sets the _code and _output attributes to empty strings.
- ***set_code(self, code):*** Sets the code to be compiled.
- ***get_code(self):*** Returns the current code.
- ***set_output(self, output):*** Sets the compiled output.
- ***get_output(self):*** Returns the compiled output.
- ***compile(self):*** Compiles the code using a Python interpreter. It takes the code provided by set_code method as input and returns the output of the compiled code.

f) Any important data structure in class/methods: There are no data structures used in this class.

g) Briefly describe any algorithm that you may have used and why did you select it upon other algorithms where more than one option exists:
The subprocess module in Python is used to spawn new processes, execute commands, and connect to the input/output pipes of those processes. The subprocess.run() method was used to run the command 'python -c <code>' to compile the code passed by the user. If the return code of the process is 0, then it is considered a success, and the output is stored. If the return code is non-zero, then the error message is stored. The reason for selecting this algorithm is that it is a simple and effective way to execute the command-line arguments, and it provides a way to capture the output of the command

a) Module name: Main
b) Date: 4/18/2023
c) Programmer's name: Ryan
d) Brief description: This module defines a Flask web application that compiles and executes Python code, and includes the routes for the app. It also includes a Twilio client for sending and receiving SMS messages.
e) Important functions:

- ***homepage():*** Renders the home page template.
- ***test():*** Tests the connection to the server.
- ***compile_code():*** Compiles the code received from the POST request using the Compiler class and returns the output as a JSON response.
- ***sms():*** Receives an SMS message, compiles the code, and sends the output back to the same phone number.

f) Important data structure: The app object of the Flask class is used to define the routes and start the server.
g) Algorithm: No specific algorithm was used in this module

a) Module/class name: HomeScreen
b) Date: 4/18/2023
c) Programmer's name: Christina
d) Brief description: This class represents the home screen of the application and contains the UI design for the welcome screen.
e) Important functions:

- ***build():*** This function returns the scaffold widget with the UI design for the home screen. It takes the context as input and returns the scaffold widget as output.
- ***onPressed():*** This function is called when the "Click to Start" button is pressed, and it navigates the user to the SecondPage screen.

f) Data structures: There are no important data structures used in this class.
g) Algorithms: There are no specific algorithms used in this class.

a) **Module/Class name:** SecondPage
b) **Date:** 4/18/2023
c) **Programmer's name:** Christina
d) Brief description of the class/module: This is a StatelessWidget class that defines the layout for the second page of the app. This page displays a list of users and has a "Next Page" button that takes the user to the chat page when clicked.
e) Important functions:

- ***build():*** This function returns a Scaffold widget with a custom scroll view that displays a list of users and has a floating action button that takes the user to the chat page when clicked. The function takes in a BuildContext object as an argument.

f) Any important data structure in class/methods:

- ***CustomScrollView:*** A widget that allows users to create scrollable lists that can be customized with slivers. The SecondPage class uses this widget to create a list of users that can be scrolled through.

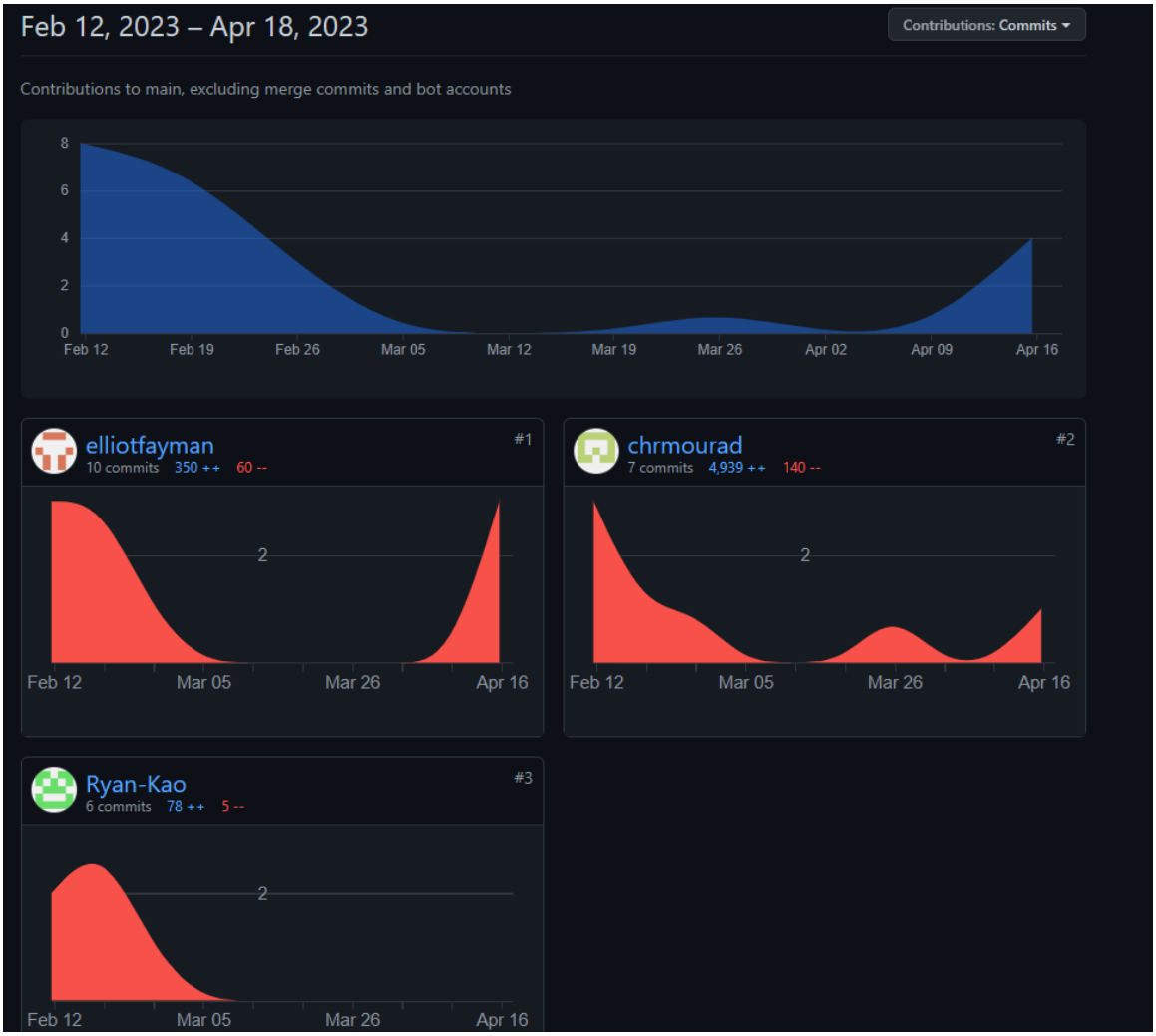
g) Algorithms: No algorithm was used in this class.

a) **Module/Class name:** ChatPage.
b) **Date:** 4/18/2023
c) **Programmer's name:** Christina
d) Brief description of the class/module: ChatPage is a class that represents the chat page UI in a messaging application. It contains a list of messages and a text field to send a new message.
e) Important functions:

- ***build()*** function creates the UI for the ChatPage. It returns a Scaffold widget, which contains a Column widget with two children: an Expanded widget to display the list of messages, and a Container widget with a TextField to send a new message.

f) Any important data structure in class/methods: messages is a List<Message> that contains the messages to be displayed in the chat.
g) Algorithms: There are no algorithms used in this class.

3.3 Individual Contributions



Percentage Estimate of Code Contribution:

33.33% distribution evenly amongst Elliot Fayman, Ryan Kao, and Christina Mourad.