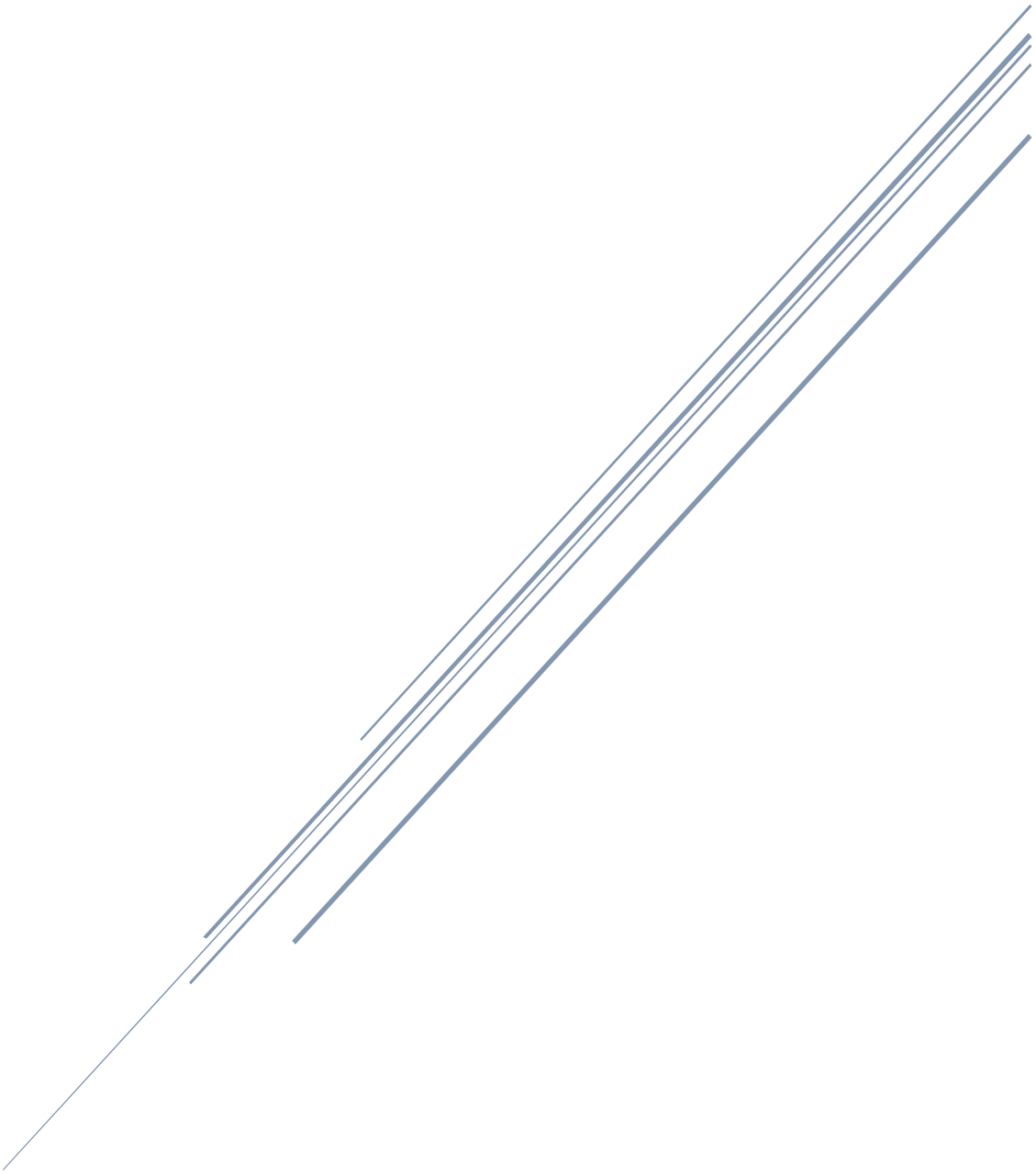
BACKEND CHALLENGE DESIGN SPECIFICATION

Project title: Atlan Form



Made By: Saurabh Yadav

Table of Contents

Introduction

The "Atlan Form Demo" project aims to address the challenge of data collection by providing a comprehensive platform for organizations, regardless of their size, to effortlessly monitor and gather the data they require. The project's primary objective is to democratize data access, empowering end-users with the ability to collect and analyze data effectively.

Purpose:

The purpose of the "Atlan Form Demo" project is to offer a user-friendly solution for data collection. It serves as a practical and accessible platform that enables individuals and organizations to create customized forms based on their specific data collection needs. By incorporating diverse options for form question framing and data retrieval from the audience, the project caters to a wide range of use cases.

Main Goals

- **Democratize Data:** The project's central goal is to democratize data access by putting the power of data collection in the hands of end-users, allowing them to make informed decisions based on collected insights.
- **Versatile Form Creation:** "Atlan Form Demo" seeks to provide a versatile form creation process that accommodates various data collection scenarios, enabling users to tailor forms according to their requirements.
- **Seamless Data Collection:** The platform's design prioritizes user-friendliness, making the data collection process seamless and efficient for both form creators and respondents.

Features

- **Custom Form Creation:** Users can create forms tailored to their unique data collection needs, incorporating various question types and response formats.
- **Diverse Question Framing:** The project offers a vast array of question framing options to ensure that each form accurately captures the desired information.
- **Audience Data Retrieval:** Respondent data is efficiently collected and accessible to the form creators for analysis and decision-making.
- **User-Friendly Interface:** The platform is designed with ease of use in mind, making it accessible to individuals with varying technical expertise.

Target Audience

The "Atlan Form Demo" project is intended for organizations of all sizes seeking an efficient and user-friendly solution for data collection. Its versatility caters to diverse industries and use cases, empowering users to create forms that align with their specific objectives.

Document Overview

The document overview for the design document of the application is intended to provide a comprehensive understanding of its key features and functionalities. It aims to guide readers on how the application works, detailing the various modules and their purposes. The document will delve into the components used in the application and how data flows between them, highlighting the APIs employed. It will offer insights into the chosen architecture design and provide in-depth descriptions of each model. Additionally, the document will include step-by-step instructions for installing the package on the reader's system, granting them the feasibility to interact with the user interface (UI) and graphical user interface (GUI) components. Through this detailed overview, readers will gain a clear understanding of the application's inner workings and be well-equipped to navigate its functionalities effectively.

Architectural Design

Section Overview

The Architectural Design section of the software design document provides insights into the application's design, with a specific focus on the Backend and a brief overview of the Frontend components. The Backend part of the application is built using the **Django Framework**, which has played a significant role in shaping its structure and functionalities. On the other hand, the Frontend part leverages Jinja templates and **Vanilla JavaScript (JS)** to create a user-friendly interface. The application utilizes the **SQlite3** database for storing and managing its data. Additionally, to extend its capabilities, the application integrates third-party APIs such as **Twilio** and **Google Sheet.**

The Architectural Design section outlines the foundation of the application's development, detailing the technologies and frameworks used in both the Backend and Frontend. It provides an essential understanding of how these components interact and collaborate to achieve the application's objectives. Moreover, the section highlights the key database choice and the strategic inclusion of third-party APIs to enrich the application with additional features. This section serves as a guide for developers and stakeholders, offering valuable insights into the design and construction of the application's core architecture.

General Constrains

It requires a basic system hardware with installed OS. This application is platform independent and able to run most of the operating system. Hardware with 2GB RAM and i5 Processor is sufficient to run smoothly.

To run proper it requires some necessary packages to be installed:

- Python package
- Django Framework
- gspread outh2client
- twilio
- asgiref
- dj-database-url
- gunicorn
- psycopg2
- pytzsqlparse
- whitenoise

Selecting Python as the core programming language for the entire software system offers a multitude of advantages, providing a robust foundation and powerful support for development. Python's versatility and extensive libraries make it an excellent choice for building scalable and maintainable applications. Moreover, its clean and readable syntax simplifies development, enhancing overall productivity. The choice of Python enables seamless addition of new sub-applications within the main application, promoting a modular architecture and facilitating rapid development. Furthermore, Django, a widely used web framework for Python, enhances the process with built-in features like Django REST framework, allowing easy plug-and-play of additional features, including external APIs. This approach significantly reduces development effort and time, accelerating the integration of new functionalities. Additionally, Django's self-integrated database, SQLite, provides a quick and efficient solution for data management, ensuring seamless data storage and retrieval with its lightweight nature. This combination of Python as the backbone language and the utilization of Django framework with SQLite database yields a powerful and cohesive environment for software development, fostering flexibility, extensibility, and high development productivity.

Data Design

In the application, the Data Design section revolves around the concept of Object-Relational Mapping (ORM). Django's data design primarily utilizes the ORM approach, which allows

developers to interact with the underlying database in a high-level and Pythonic manner. This design philosophy enables the application to work with database records as Python objects, abstracting away the complexities of SQL queries and database schema management.

The key components of the Data Design in Django are centered around models, which are Python classes representing database tables. Each attribute of the model class corresponds to a database field, and Django provides a variety of model fields to define the type of data each attribute represents. This ensures data validation and consistency within the database.

The database abstraction layer in Django allows developers to write database-agnostic code, making it possible to switch between different database management systems without rewriting code. The use of QuerySets enables developers to interact with the database for CRUD operations, retrieving, filtering, and manipulating data without the need for raw SQL queries.

Additionally, Django's migration system manages changes to the database schema over time, automatically generating and executing SQL commands to keep the database schema in sync with changes to the models.

The classes used in the ‘’model.py’’ file are:-

1. **User**
2. **Choices**
3. **Questions**
4. **Answer**
5. **Form**
6. **Responses**

Program Structure

This application is supported with multi-sub applications in which we can easily integrate any other additional feature without altering the code of the sub application.

As currently this application is integrated with only one sub-application associated with two third-party application i.e. Google Sheet API and Twilio API.

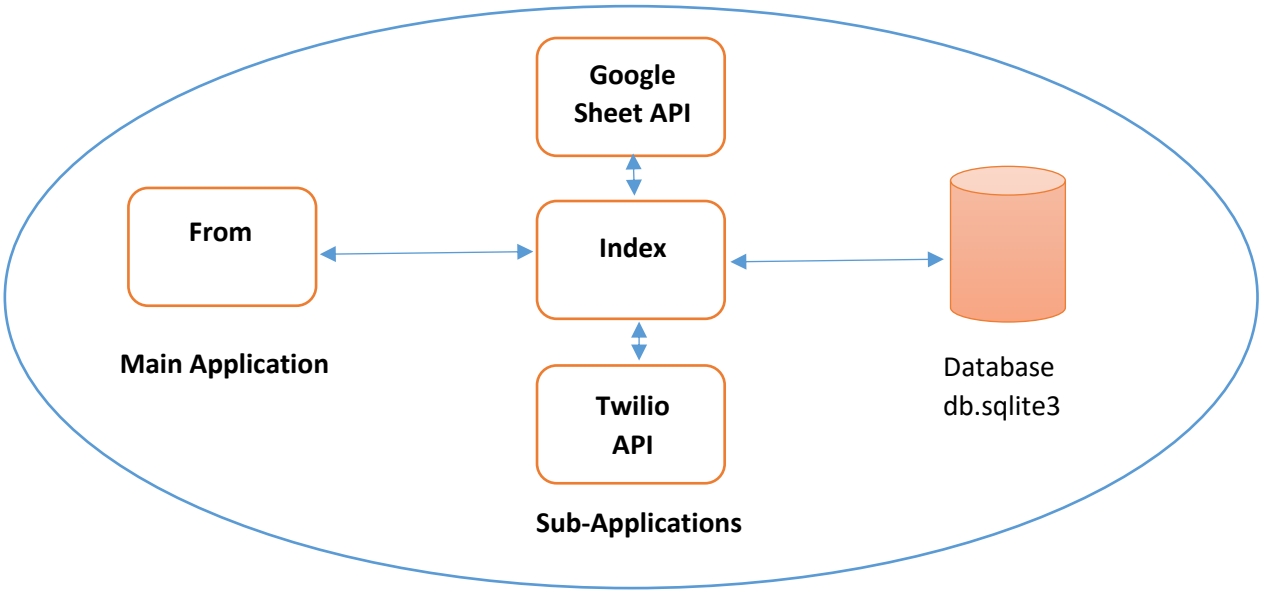


Fig. 1 Shows backend program division

Detail Design

Section Overview

The "Detail Design" section of the design document offers an in-depth examination of the various modules utilized in the application, along with the specific functions employed within each module. This comprehensive exploration sheds light on the intricate data flow between the modules and the corresponding output generated as a result. Additionally, this section elucidates the input parameters required for the seamless execution of each module, outlining the crucial role they play in producing the desired output. By providing a detailed overview of the application's internal workings, this section serves as a valuable resource for developers and stakeholders, facilitating a comprehensive understanding of the software's architecture and functionality.

Components

Framework division

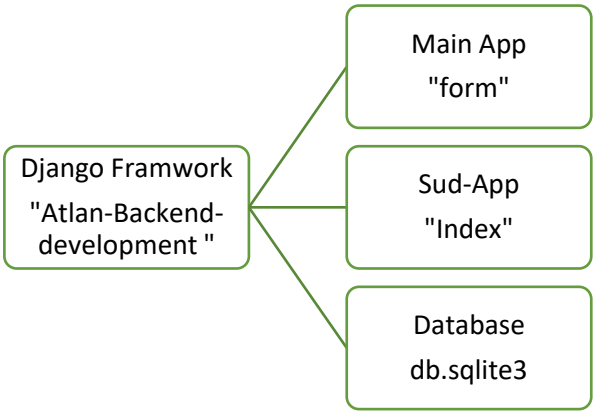


Fig. 2. Represents the backend components with the respective name of the folder

1. Main App division “Form”

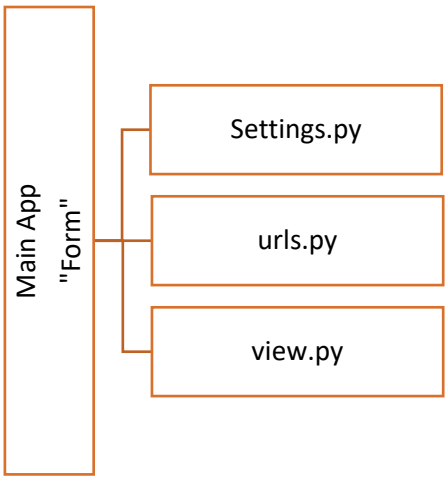


Fig. 3. Represents the Main app files

2. Sub-App division “index”

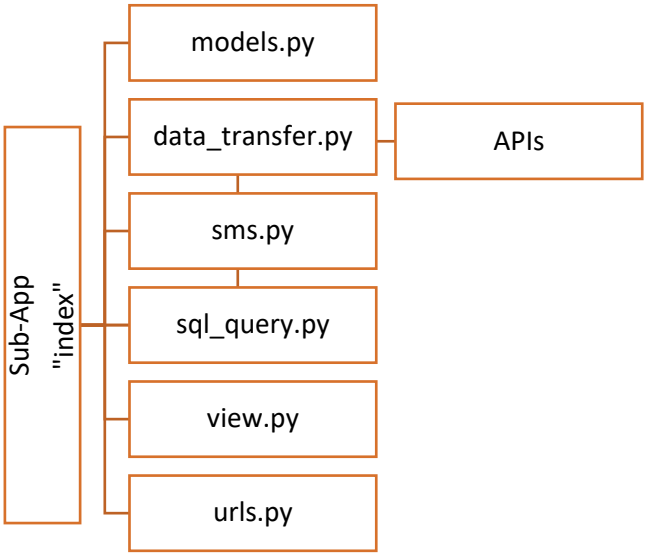
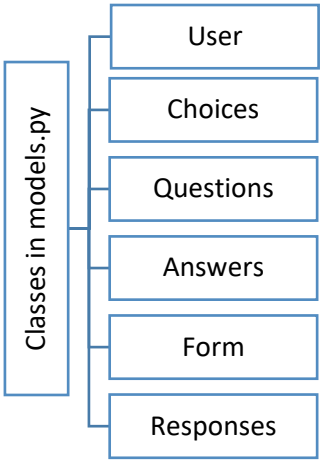
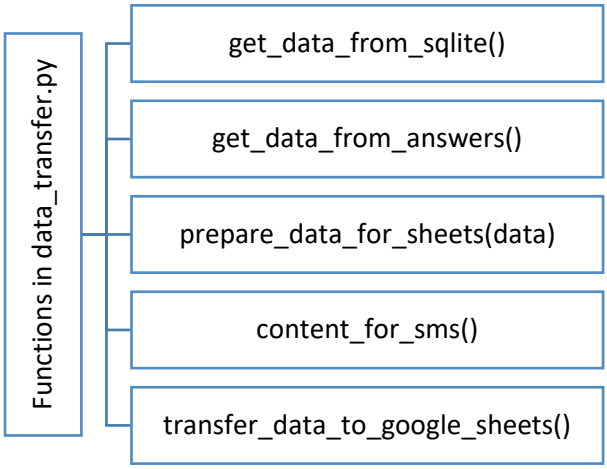


Fig. 3. Represents the Sub-App files

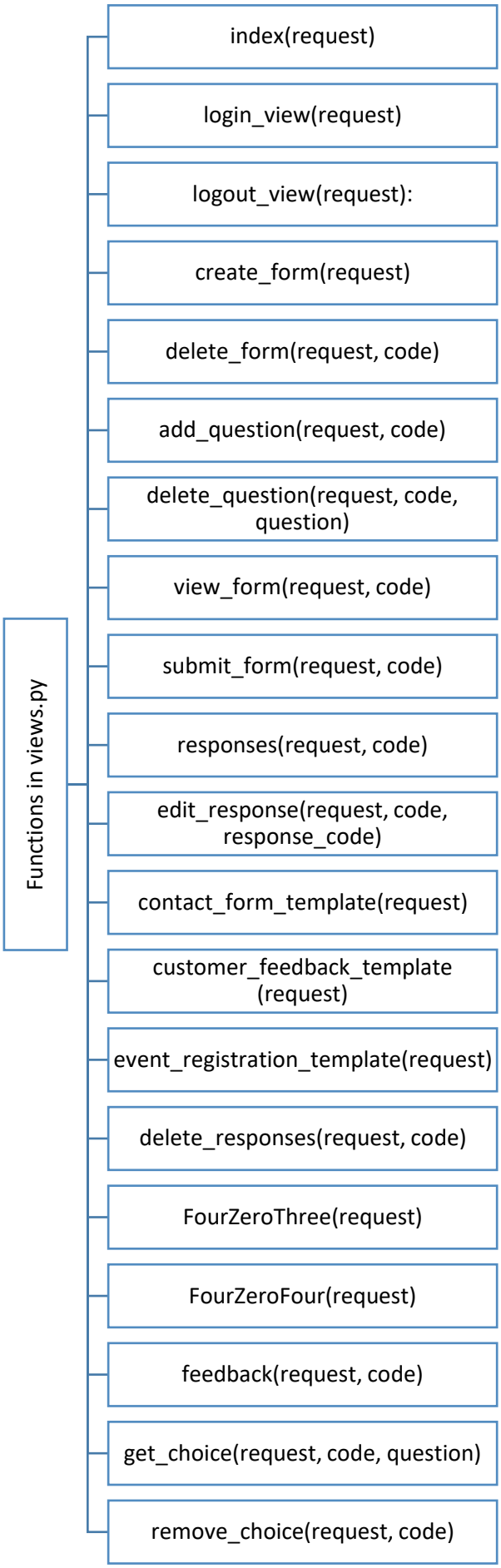
2.1. Class Division in Models.py



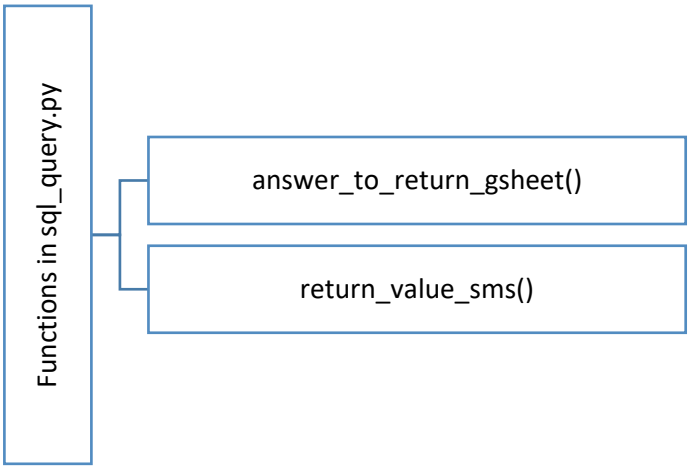
2.2. Function Division in data_transfer.py



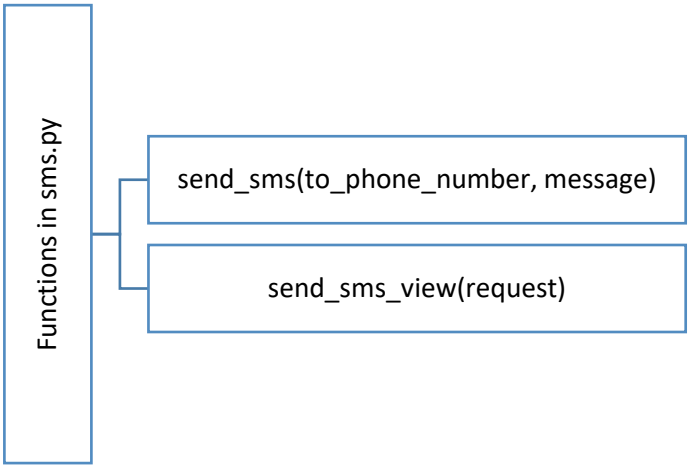
2.3. Function Division in views.py



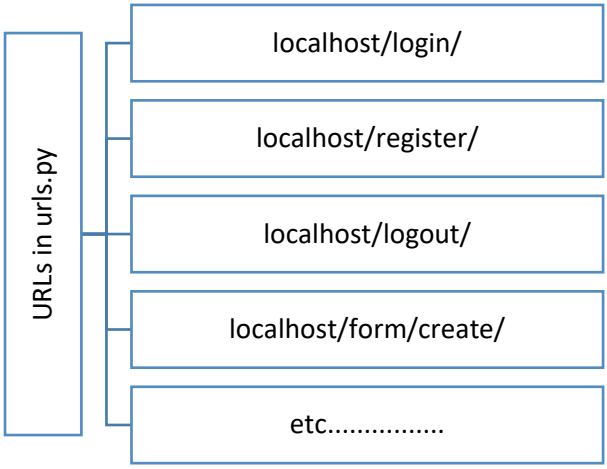
2.4. Function Division in sql_query.py



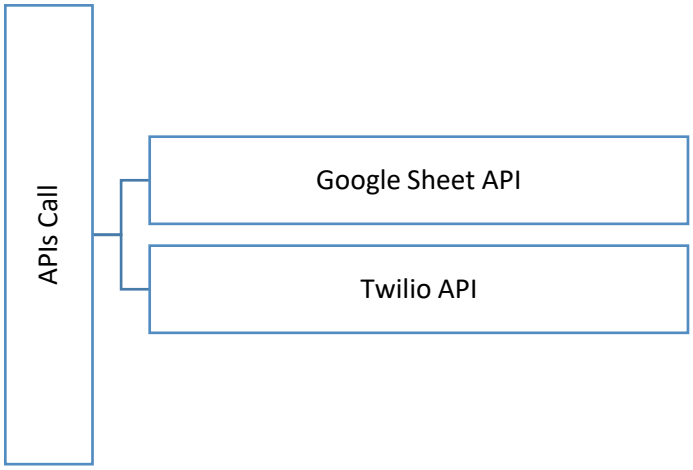
2.5. Function Division in sms.py



2.6. URLs Division in urls.py



2.7. APIs



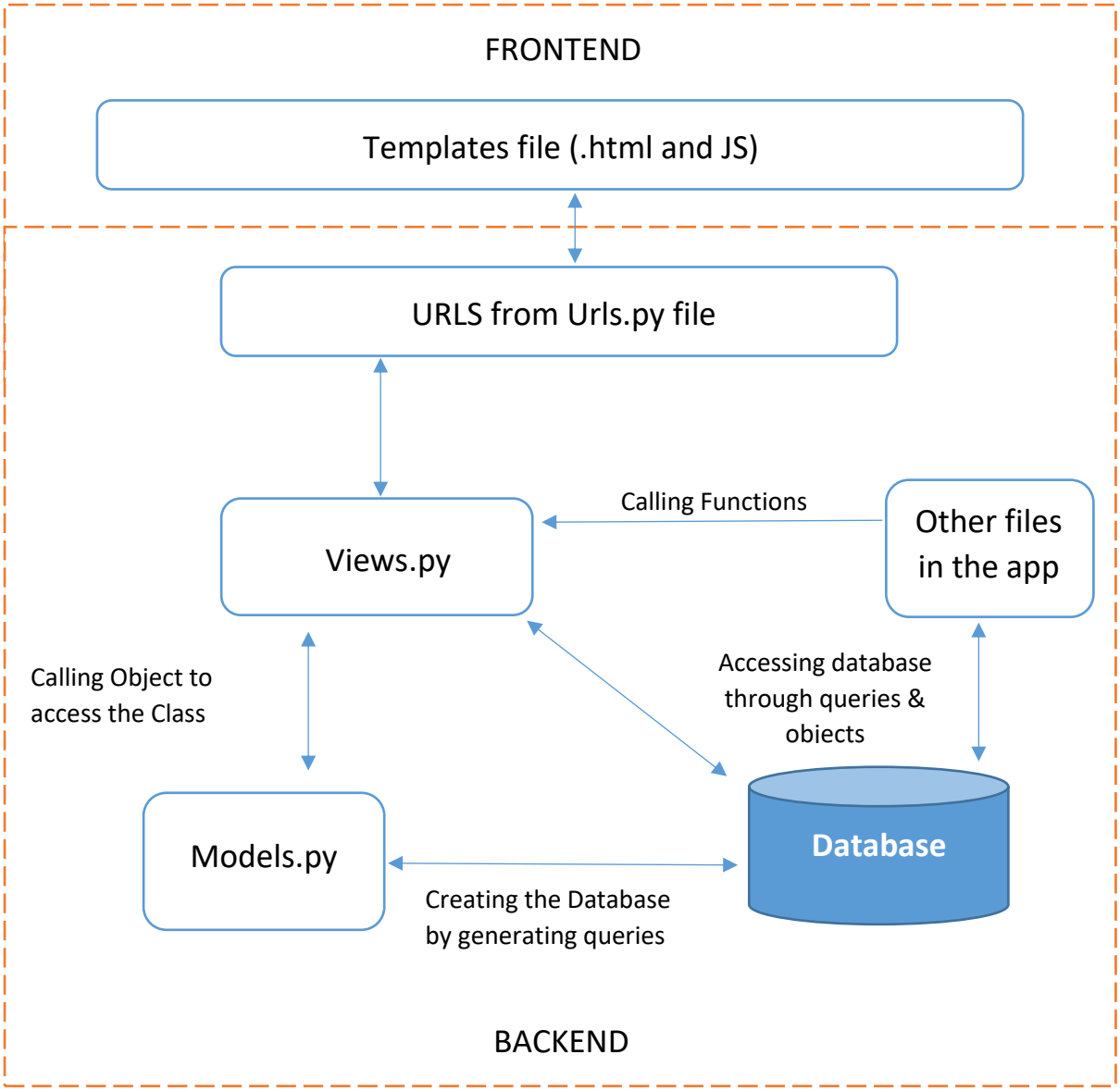
Details

The "Brief Detail of Each File" section in the software design document provides a concise and technical overview of the application's files, functions, and classes. It outlines the purpose and functionalities of each file, along with the data flow between functions and classes, and the resulting output they produce. This section acts as a valuable reference for developers, fostering a deeper understanding of the application's internal structure, logic and classes use in the app.

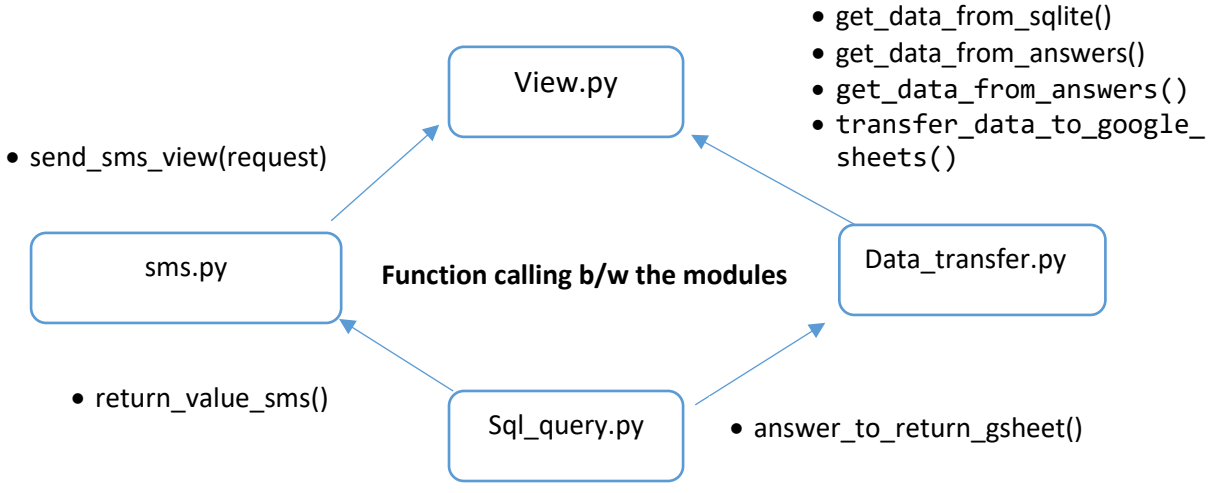
1. View.py file of index app
   In this, every function is defined to refer to the URL pages and store the value from the Form in the database by using objects of the classes defined in the **model.py** file.
   This file contains all the endpoints of the functions each function is called and revoked from this file only.
2. Data_transfer.py
3. This file is majorly focused on managing and calling the **Google Sheet API**. This file monitors the data and decides what and when data is sent to API. It directly interacts with the database and by calling the function from the **sql_query.py** file it filters out the information which is to be represented on the Google sheet.
4. sql_query.py
   This file contains functions that return filtered from the database to the function which is defined in the **data_transfer.py** and **sms.py file** by performing **SQL query**.
5. sms.py
   This file contains two functions that perform the task of calling **Twilio API** and sending SMS to the number who have submitted the form.

Appendices

Data Flow Between the different components in the ‘’index’’ sub-app



Data Flow Between the module



Installation and Links

Steps to Install the App

Step 1. Install the latest package of Python

Step 2. Activate the environment which is named as “env”. (env\Scripts\activate)

Step 3. Install the necessary package mentioned below to avoid any error in execution

- gspread outh2client
- twilio
- asgiref
- dj-database-url
- gunicorn
- psycopg2
- pytzsqlparse
- whitenoise

Step 4: Make an account on twilio app for API call by visiting the link https://www.twilio.com/en-us

Step 5: Make an account on google and Enable the Google Sheet API along Google Drive API. Now download and save the google JSON inside the folder Atlan-backend-development-.

Steps 6: Go to setting.py which is inside the ‘form’ folder (Atlan-backend-development-\form\setting.py) and make the necessary changes for the APIs (line 140 to 146)

Step 7: Run the server (command = python manage.py runserver)

Links

I am still working on the project to cover various approach and improving its functionality.

You can share your GITHUB ID on my mail: saurabhyadav1280@gamail.com

I would be happy to you as a contributor on my private repository and see my various updates.

My Github repository for the project is : https://github.com/Saurabh-0532/Atlan-backend-development-