# Technologies and architecture

## Dynamic functionalities

The most important dynamic functionality in our service filtering of dog walker based on location. This functionality depends on the dog owner address. The following sequence explains the flow of the web service of the dog owner point of view

1. Dog owner sign up for the service and or sign in.
2. Dog owner fills his details if it’s not already filled (address location…)
3. Dog owner starts to search Dog owner by location. Django-google-maps API are used to display maps on the web service and provide the basic hooks into google maps V3 API.
4. Dog owner chooses a Dog walker from the map. After choosing the Dog walker, dog owner can book the time.

The following sequence explains the flow of the web service of the dog walker point of view.

1. Dog walker sign up for the service or sign in.
2. Dog walker fills his details if it’s not already filled (address location, available time …)

As stated before, our web service tries to connect dog owner with dog walker and not vice-versa. So most of the work from the dynamic point of view, it’s done on the dog owner side.

# Technologies

Our web service is a standard HTML5 application that can be run with modern web browser. We are using as backend Django which is using Model-View-Template design pattern. Django’s views correspond to the controllers in traditional MVC and Django’s templates correspond to the views in traditional MVC. The Front end is build using twitter Bootstrap engine. Bootstrap is a free and open-source CSS framework directed at responsive, mobile-first front-end web development. It contains CSS- and JavaScript-based design templates for typography, forms, buttons, navigation, and other interface components.

Table 1. Backend software requirements.

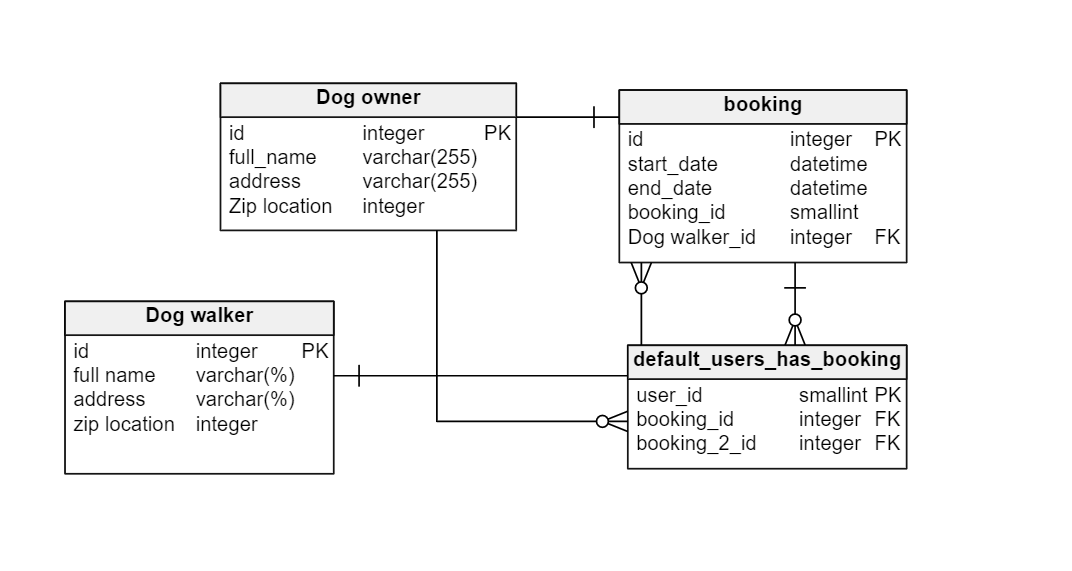
|  |  |  |  |
| --- | --- | --- | --- |
| Name | Version | Usage | purpose |
| python | 3.7.3 | Backend language | Main language for the backend |
| Django | 3.0.3 | Web framework | MVC framework to build the service |
| Django-Google-maps | 0.12.1 | Google map API | Provide basic hooks into google maps V3 API. |
| Sqlite3 | 3.31.x | Database engine | Provide easy to use functionality |
| PostgreSQL | 9.x.x | Production user data | Will be used when hosting it to a server |

Table 2. Front-end software requirements.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Version | Usage | Purpose |
| Bootstrap | 4.4.1 | Front end framework | Help to build responsive website compatible with mobile |
| CSS | 4 | Style sheet | Embedded in Bootstrap. |
| JavaScript | x | DOM manipulation | Embedded in Bootstrap. |

# Database

For database engine, we are using Sqlite3. In the development phase, we are using Sqilite3 database since it is very easy and straightforward to use with Django. At the end, we will use PostgreSQL, since it will be easy to host it in the service.(Not decided yet). This is the simplified database model for our web service. As previously stated, most database operation is done on the DOG owner side.



# Security

Django's template system protects you against the majority of XSS attacks by escaping specific characters that are "dangerous" in HTML. Django comes with a user authentication system. It handles user accounts, groups, permissions and cookie-based user sessions. Authentication support is bundled as a Django contrib module in **django.contrib.auth**. By default, the required configuration is already included in the **settings.py** generated by [**django-admin startproject**](https://docs.djangoproject.com/en/3.0/ref/django-admin/#django-admin-startproject), these consist of two items listed in your [**INSTALLED\_APPS**](https://docs.djangoproject.com/en/3.0/ref/settings/#std:setting-INSTALLED_APPS) setting. If this application came to the real use, we’d need a more robust security solution and need to integrate an encrypted connection to our web service.