Contents

Introduction

This project aims partially automate Lanus system and to implement a convolutional Neural network to create a parametrized model of a house. This document will discuss the design of the project including the project plan, ER diagrams and database design and interactions.

Project Plan

In Figure 1 a Gantt chart can be seen. This represents the timeline of the project. Showing when and how long each development takes place. A brief description can also be seen in Figure 2.

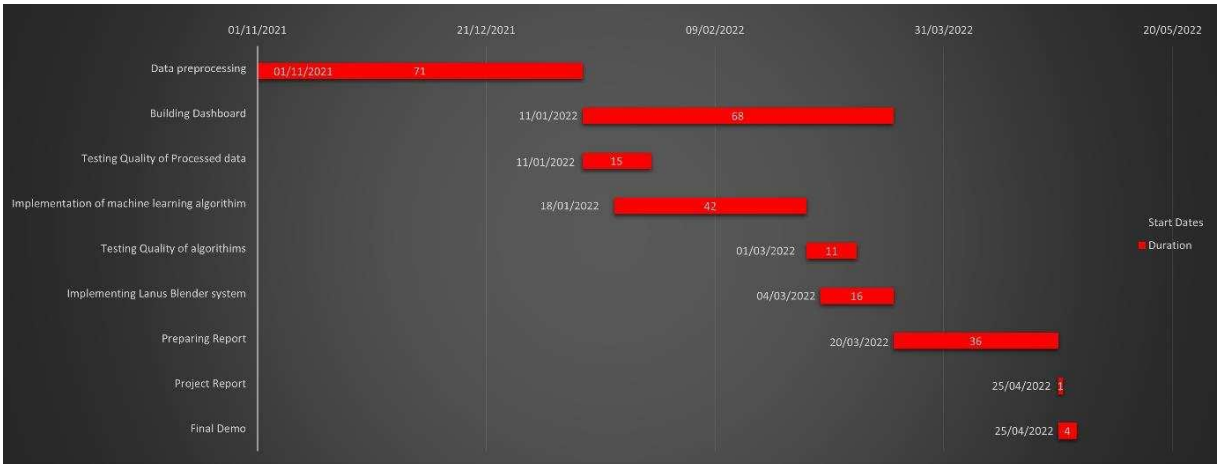

Figure 1, Gantt chart for project plan

TASK NAME	START DATE	DUE DATE	DAYS	DESCRIPTION
Data preprocessing	01/11/2021	11/01/2022	71	Turn raw data from database into usable data for machine learning process
Building Dashboard	11/01/2022	20/03/2022	68	Create Dashboard which can visualize premade data and allows users generate new data
Testing Quality of Processed data	11/01/2022	26/01/2022	15	Test Quality of data to and re-process data if it's unsuitable
Implementation of machine learning algorithim	18/01/2022	01/03/2022	42	Create machine learning algorthim to make accurate prediction of roof shape at minimum
Testing Quality of algorithims	01/03/2022	12/03/2022	11	Testing accuracy of predictions
Implementing Lanus Blender system	04/03/2022	20/03/2022	16	Link Lanus blender system to the project's data
Preparing Report	20/03/2022	25/04/2022	36	Finalize design, specification and research documents. Create website displaying documents
Project Report	25/04/2022	26/04/2022	1	**Project Report is due**
Final Demo	25/04/2022	29/04/2022	4	**Final Demo is due**

Figure 2, Description

System Architecture

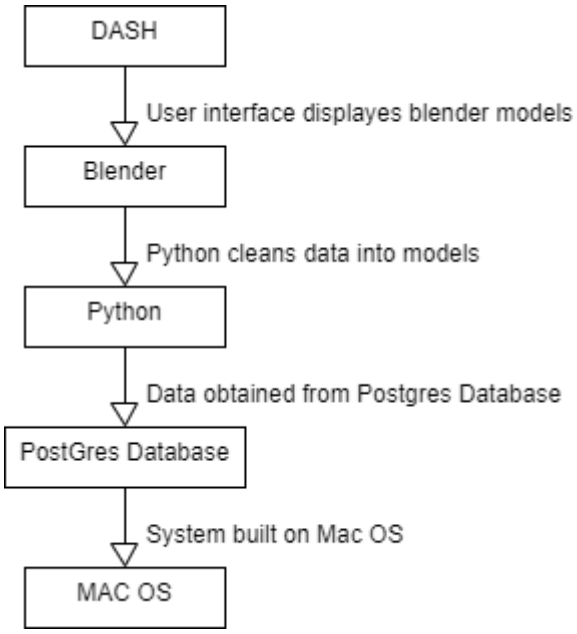

Figure 3, System architecture

Figure 3 describes the communication of the components within the system architecture. The of a back-end algorithm built with Python which connects to a Postgres database. The system was built to run a dashboard on a Mac OS. Dash will serve as a front-end user interface. This will provide the user will a simple interface which allows them to visualize and graph the data as well as generate new data based on an input address.
Python will serve as the 'glue' which connects the database with the front-end model creation. Python is used in the system to process large quantities of data and the automatic building of models in blender. Python was chosen because it provides acceptable performance alongside rapid development and a large array of extremely helpful libraries, especially when processing data. It is also largely cited as the easiest and faster language to implement machine learning models. PostgreSQL will serve as the database manager for the system. This was chosen due to its PostGIS features which allow for rapid manipulation of the data inside the database as well as some other very useful geometric functionality that saves development time on the project.
The Mac operating system was chosen mainly because it is the system that Lanus system is built on and is, therefore, easier to link to their parallel projects.

Database design

The design of the database was built for simplicity and ease of use. This database is acting as a one-way stream of data outbound. The system only needs to extract data. The main reason a database was used in the system was to allow for quick extraction of geometry inside the large NPS dataset.

Tables

Only one table was necessary for the system and that was the nps_cropped_lynmouth table. This table stores the gid, id, and geometry for each site stored in a cropped version of the NPS data. The data was cropped to just the necessary square kilometer before creating this table to allow for faster extraction of the data. In Figure 4 the SQL command used to create this table can be seen.

```sql
-- Table: public.nps_cropped_lynmouth

-- DROP TABLE public.nps_cropped_lynmouth;

CREATE TABLE IF NOT EXISTS public.nps_cropped_lynmouth
(
    gid integer NOT NULL DEFAULT nextval('nps_cropped_lynmouth_gid_seq'::regclass),
    id double precision,
    geom geometry(MultiPolygon,27700),
    CONSTRAINT nps_cropped_lynmouth_pkey PRIMARY KEY (gid)
)

TABLESPACE pg_default;

ALTER TABLE public.nps_cropped_lynmouth
    OWNER to postgres;
-- Index: sidx_nps_cropped_lynmouth_geom

-- DROP INDEX public.sidx_nps_cropped_lynmouth_geom;

CREATE INDEX sidx_nps_cropped_lynmouth_geom
    ON public.nps_cropped_lynmouth USING gist
    (geom)
    TABLESPACE pg_default;
```

Figure 4, Creation SQL commands

SQL Statements

Multiple different SQL statements are used to extract geometry from the NPS dataset. The statements allow for the use of PostGIS functionality. Functions like ST_GeomFromText and ST_AsText are used through the following statements to transfer the geometry between text and a PostGre geometry object which cannot be read by humans or python.
Firstly ST_Contains is used to query the NPS dataset for the site geometry when given the X and the Y of the geolocated address. This statement can be seen in Figure 5

```
SELECT ST_ASText(geom) FROM public."nps_cropped_lynmouth"
    WHERE ST_Contains(ST_AsText(geom), ST_GeomFromText('POINT(x y)'))
```

Figure 5, ST_Contains

ST_DWithin was also implemented to find the neighbors of a given site. This returns all geometries within a certain distance(d) of an X and Y location. The statement used in the project can be seen in Figure 6.

```
SELECT ST_ASText(geom) FROM public."nps_cropped_lynmouth"
    WHERE ST_DWithin(ST_AsText(geom), ST_GeomFromText('POINT(x y), d'))
```

Figure 6, ST_Dwithin

ST_Transform doesn't directly link to the NPS dataset instead is a PostGIS function that lets users transform the SSRID of their geometry (Geom). The following statements seen in Figures 7 and 8 were used in the project to do just that.

```
SELECT ST_ASText(ST_Transform(ST_GeomFromText(geom, 27700), 4326) as wgs_geom)
```

Figure 7, ST_Transform example

```
SELECT ST_ASText(ST_Transform(ST_GeomFromText(geom, 4326), 27700) as wgs_geom)
```

Figure 8, ST_Transform example 2

ST_Area was used in the same fashion to find the internal area of a given geometry. The statement for this can be seen in Figure 9 below.

```
SELECT ST_ASText(ST_Area(ST_GeomFromText(geom 4326)))
```

Figure 9, ST_Area

UML Diagrams

System Sequence Diagram

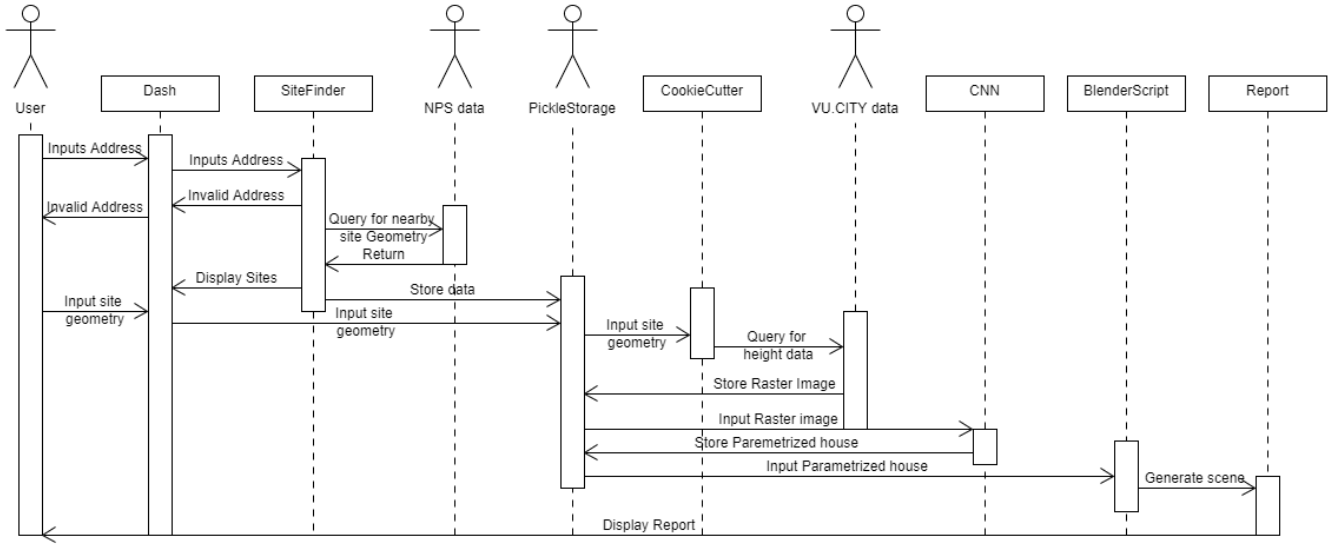

Figure 10, System Sequence Diagram

Sequence Diagrams

Site Finder

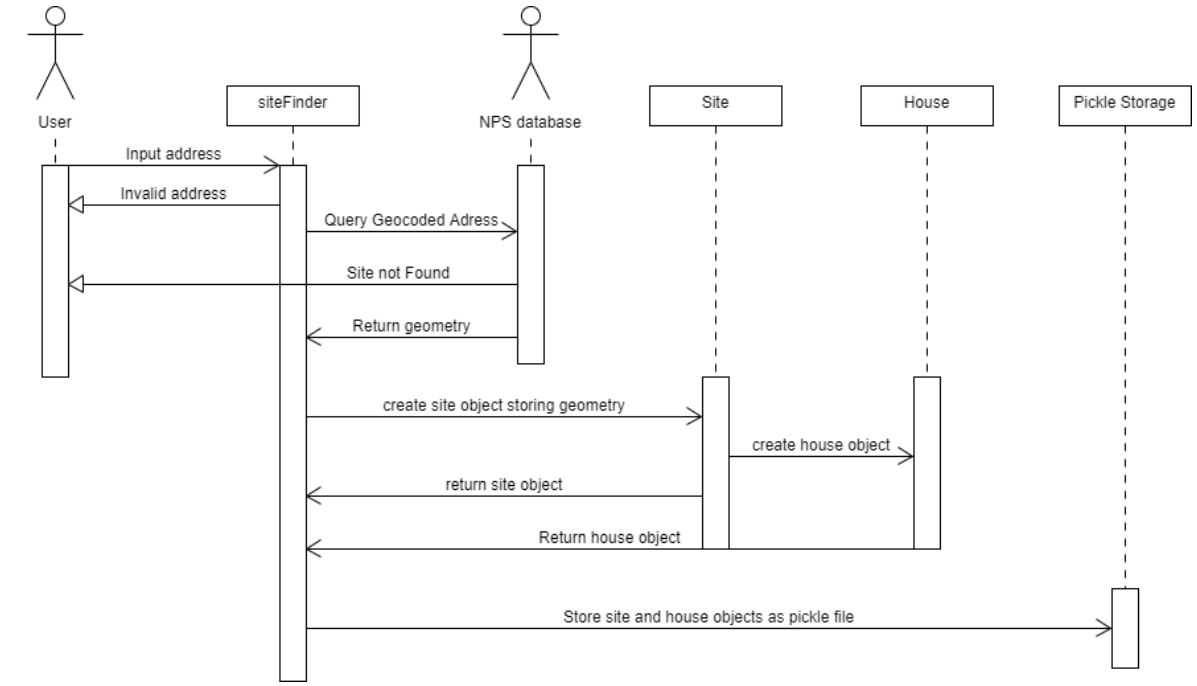

Figure 11, Site Finder Sequence Diagram

Cookie Cutter

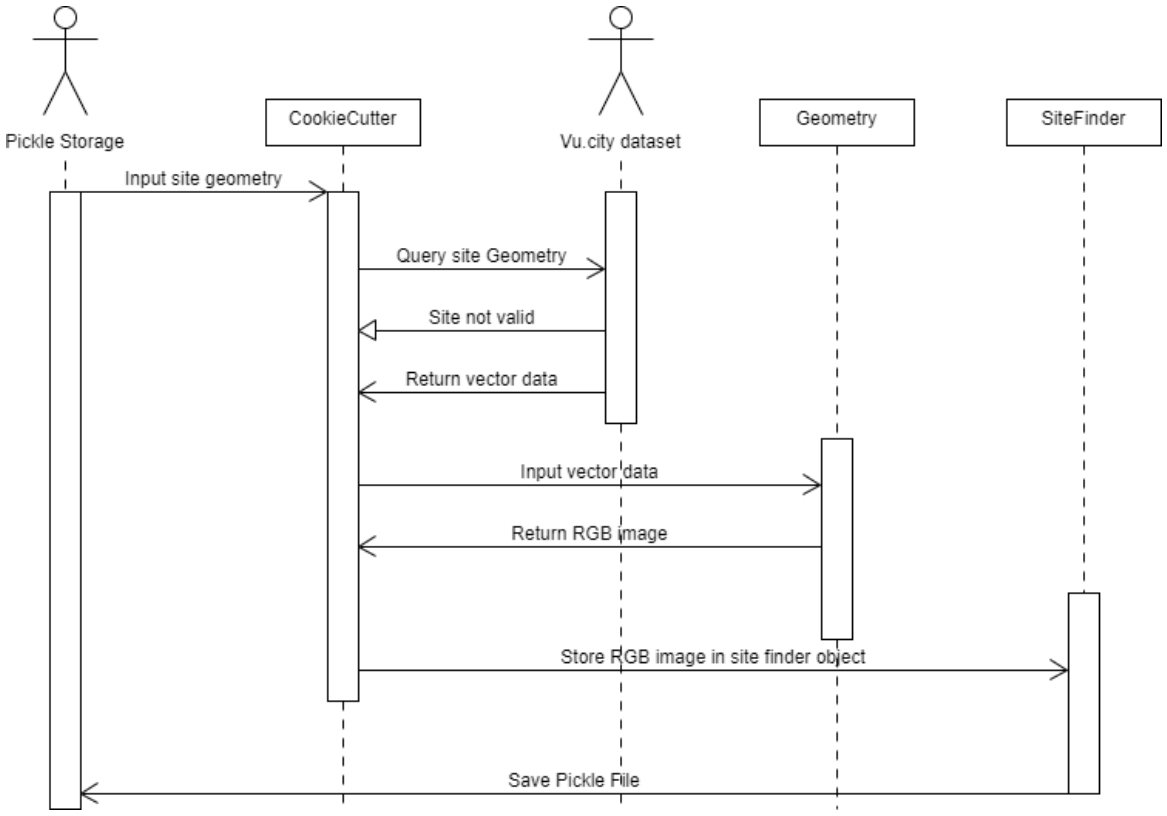

Figure 11, Cookie Cutter Sequence Diagram

Image classifier

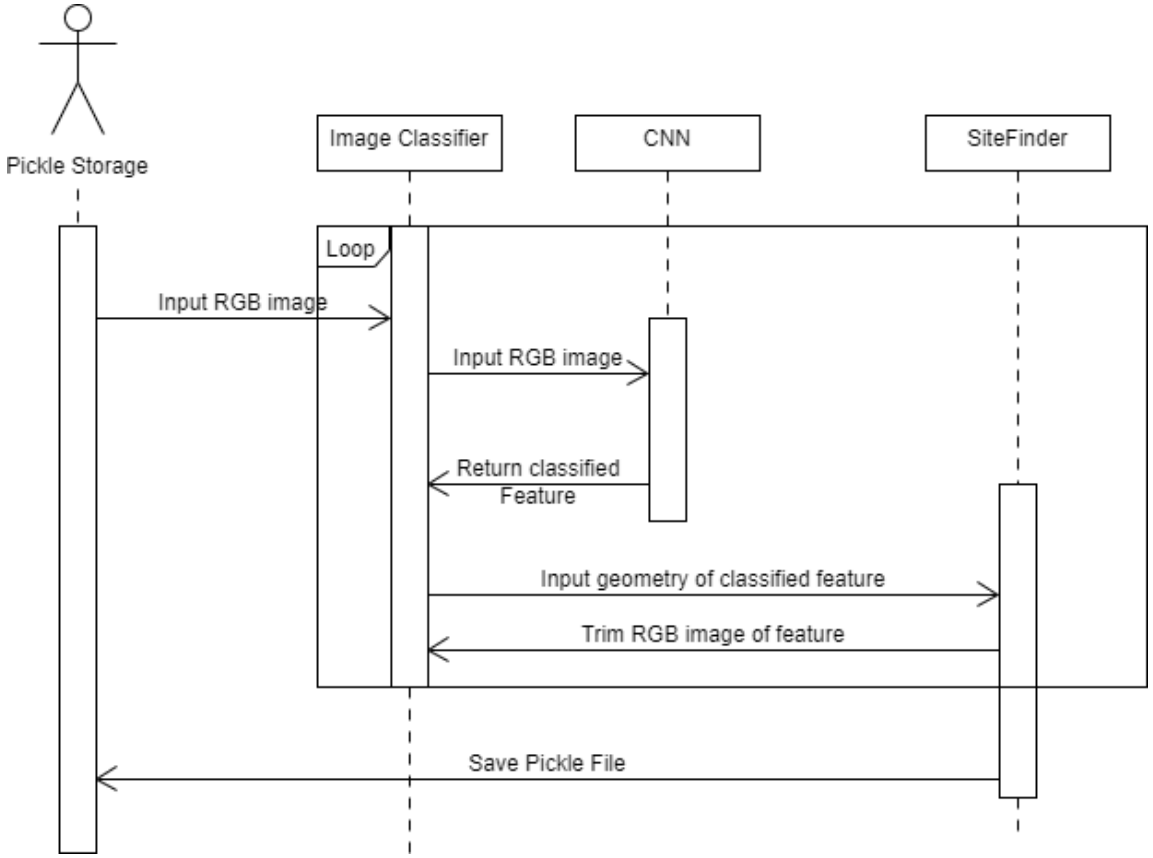

Figure 12, Image Classifier / CNN Sequence Diagram

Blender Model Generation

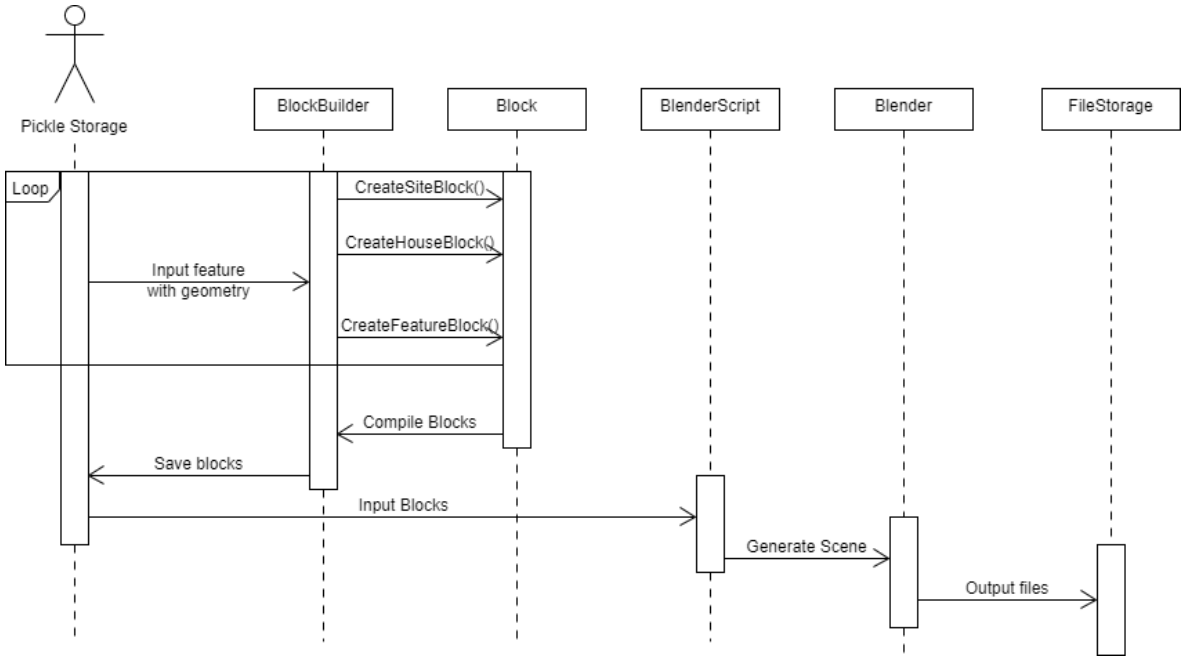

Figure 13, Blender Sequence Diagram

Report Generator

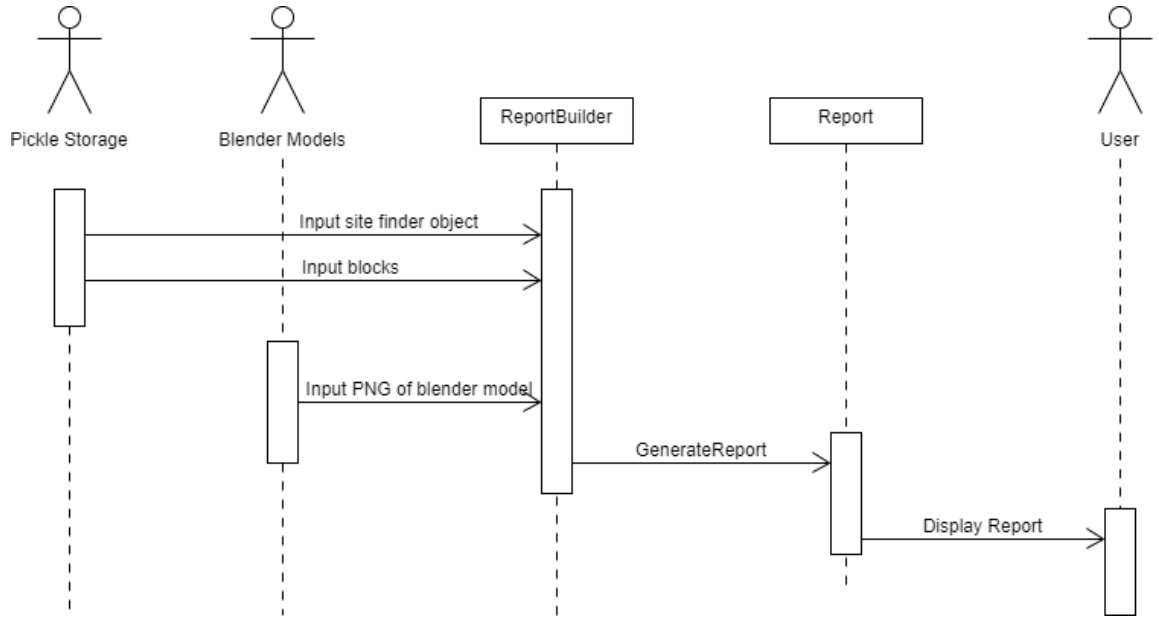

Figure 14, Report Sequence Diagram

ER-Diagrams Class Diagrams

Relationship

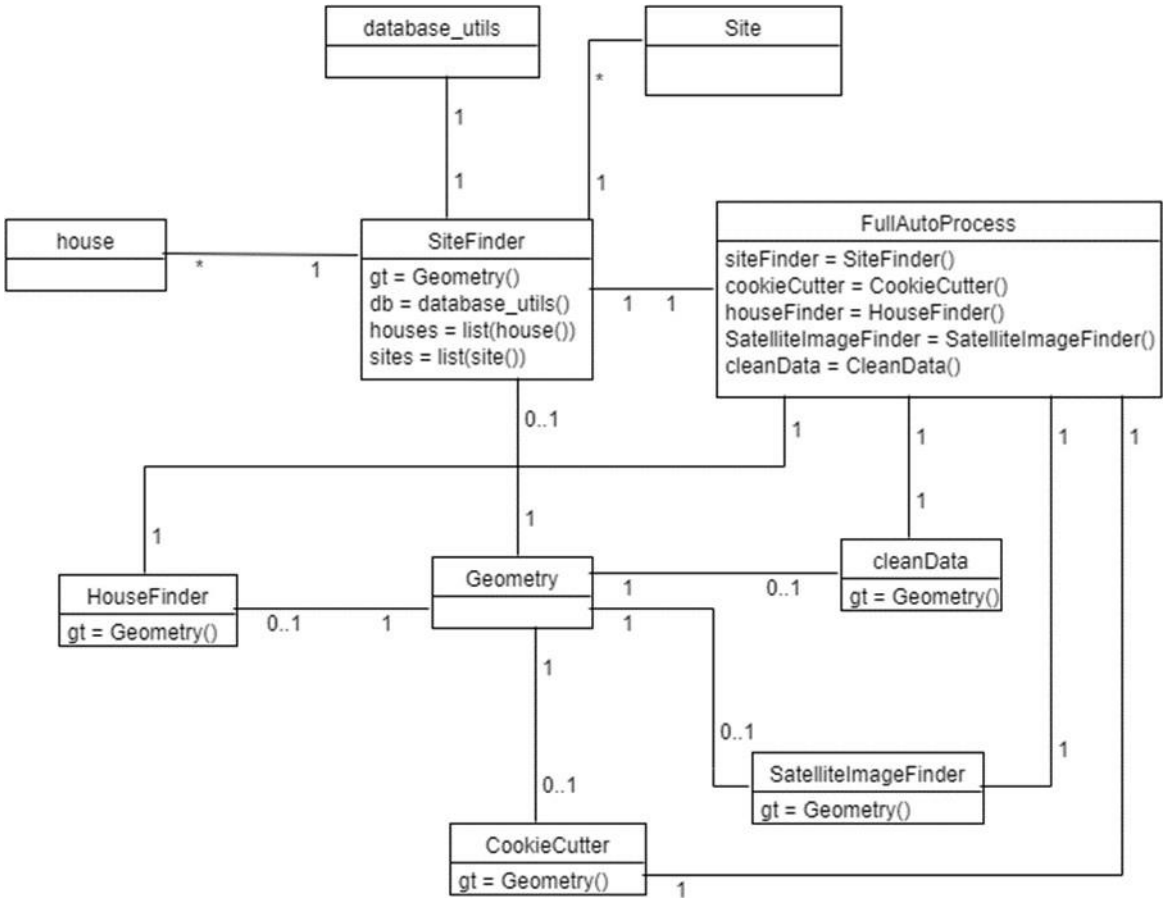

Figure 15, ER Relationship Diagram

Variables and methods of main classes

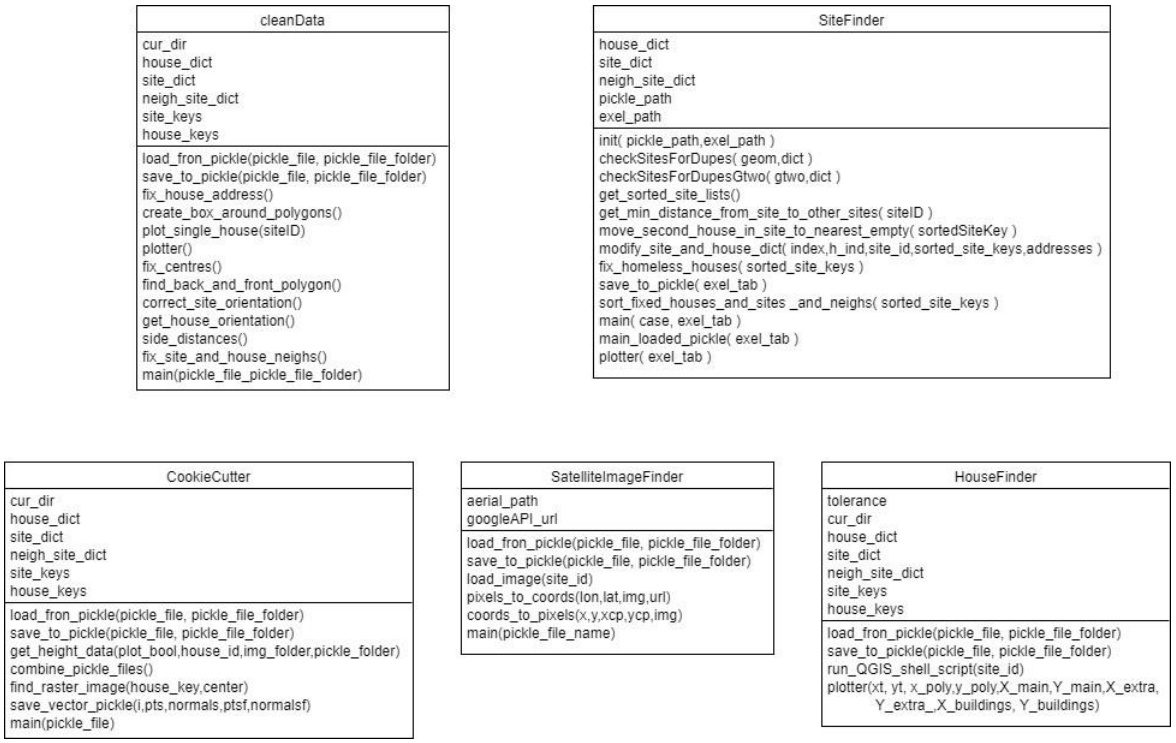

Figure 16, Variables and methods – Main classes Diagram

Variables and methods of siteFinder inherited classes

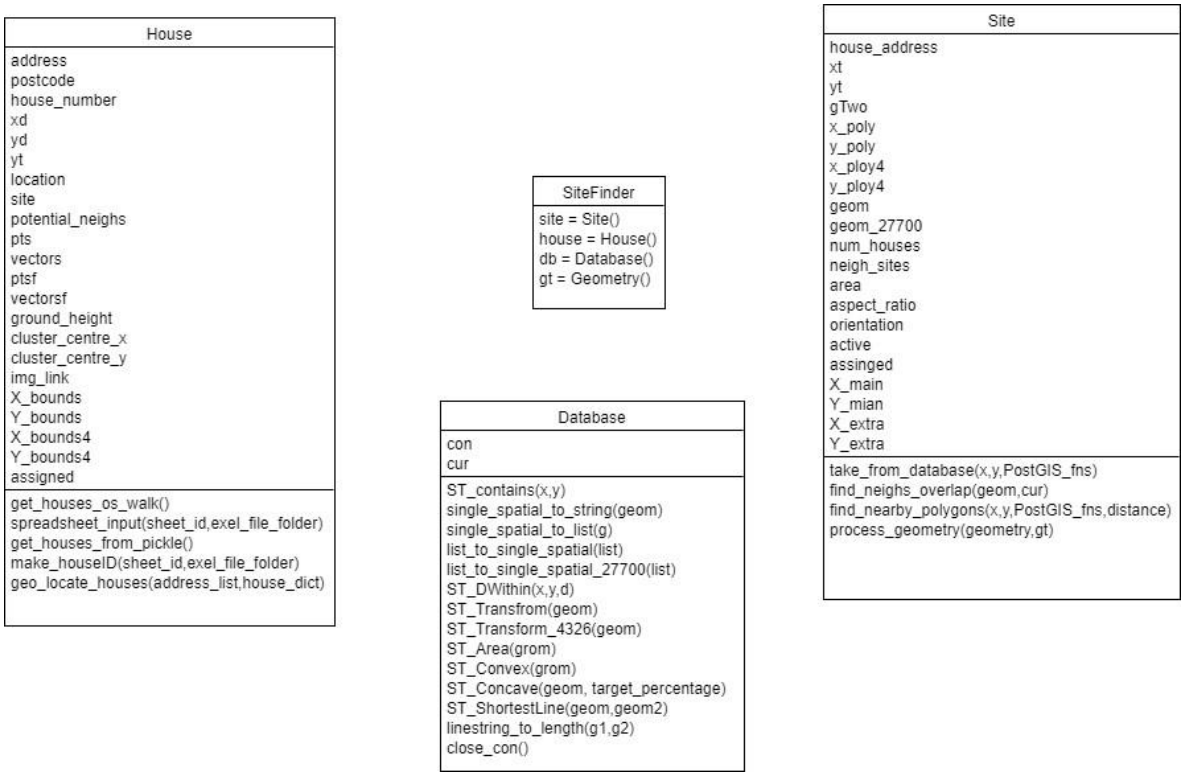

Figure 17, Variables, and methods – SiteFinder classes Diagram

Variables and methods of Geometry Class

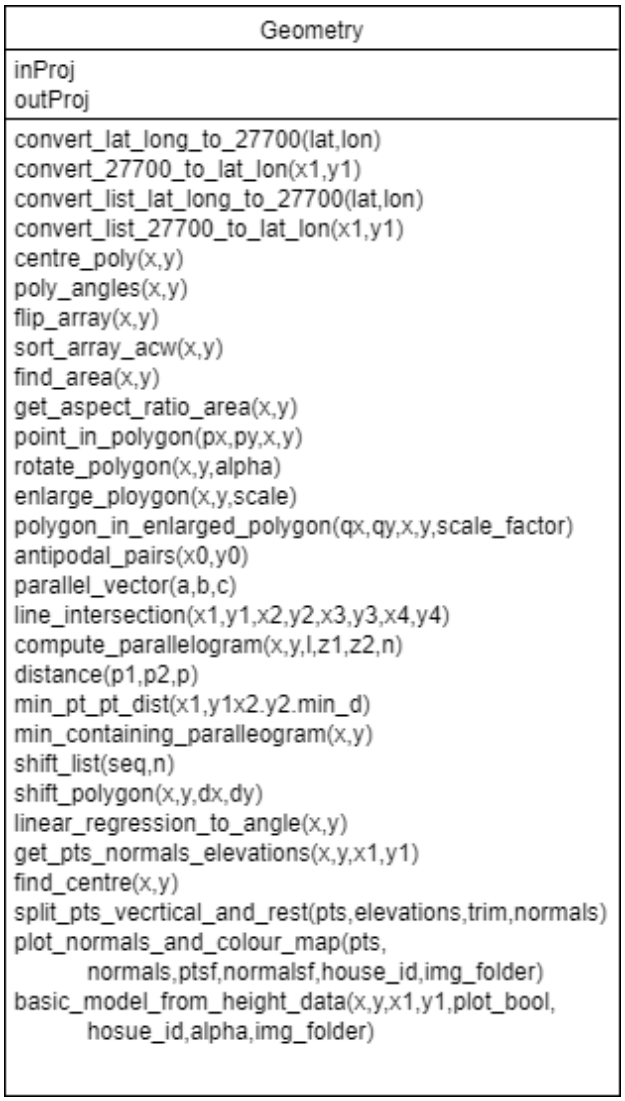

Figure 17, Variables, and methods – Geometry classes Diagram