**NostraDomicile**

Ochaun Marshall, Richard Andrews, Christian Simaan, Jeremy Hutton

**Project Description:**

**Goal:**

The goal of the NostraDomicile Project is to create a web application whose two main functions are to predict whether a house will sell in a specific area based on the home’s attributes, and given a zip code, what are the most important factors leading to a sale in that area.

**Description:**

NostraDomicile will accomplish this goal by retrieving and storing housing market information using a Zillow API and MySQL database, using machine learning to evaluate housing data and determine factors influencing home sales in a particular area, and creating a user-friendly interface for users to view data about factors influencing home sales and create data visualizations about houses on the market based on user preferences.

**Requirements:**

The following is a list of requirements for the NostraDomicile project:

**Functional:**

* Users able to input attributes and location for predictive home sale analysis (1st-tier)
* Users able to filter homes listed by attributes and location(1st-tier)
* Users able to create visualizations for housing data based on filters (1st-tier)
* Users able to view most influential factors in home sales for a given area (1st-tier)
* Users able to read a blog comprised of articles by real estate experts.(1st-tier)
* Rank of Realtor sales in the area.(2nd-tier)
* Predictive home sale analysis based on price-point (2nd-tier)
* Statistical home purchase analysis based on attributes and location (2nd-tier)
* Suggest home alterations to potentially enhance sale value (3rd-tier)

**User Interface:**

* Users able to enter “About” page that gives detailed explanation of functions of the web application and its goal.
* Users able to enter “Blog” page which will contain articles by experts in the field of real estate.
* A text area which takes a zip code which will be used throughout the session for various predictions and data visualizations.
* User selects from “factor” drop down boxes which allow a user to enter their own home’s attributes which will be used throughout the session in order to see if their house would be predicted to sell.
* User may select a button which will return most important factors to home sales in that area.
* Checkboxes and a submit button which will request different data visualizations.
* Text area that takes a price amount from user to predict home sale based on that price and attributes.
* “Get Price Estimate” button which returns a prediction on a price for which a house might sell.
* “Get Suggested Home Improvement” button which returns possible home improvements and the value gain for such an improvement.

**Usability:**

* The application will load within 1-2 second interval.
* All buttons will conform to the same style.
* Any text area, checkbox, or drop-down box will have helpful instructions.
* Any function of the web application may be reached within 2-3 clicks.
* Any subsequent page with the application will adhere to the same style.

**Performance:**

* Any system process taking longer than 1 second will display a “processing” dialog box.
* Any system process taking longer than 8 seconds will display an estimate of the time remaining on the task.
* 95% of response time should be less than 5 seconds for processes involving queries from the database.
* Database will respond to 1,000 reads per hour.
* Data Visualizations will appear after user request within 2 seconds.
* Must support concurrent users.

**System Interface:**

* Application must be successfully hosted and displayed by cloud service(AWS).
* Front end of web application must successfully query database upon user request.
* Database must successfully return requested data run through machine learning algorithm, statistical analysis and data visualization program and front end must successfully display request.

**Security:**

* Home data in DB can’t be altered except by authorized automated scripts or administrators.
* Realtor rankings and underlying data in DB can’t be altered except by automated scripts or administrators.
* Realtor blog articles can’t be placed, removed or altered except by administrators.

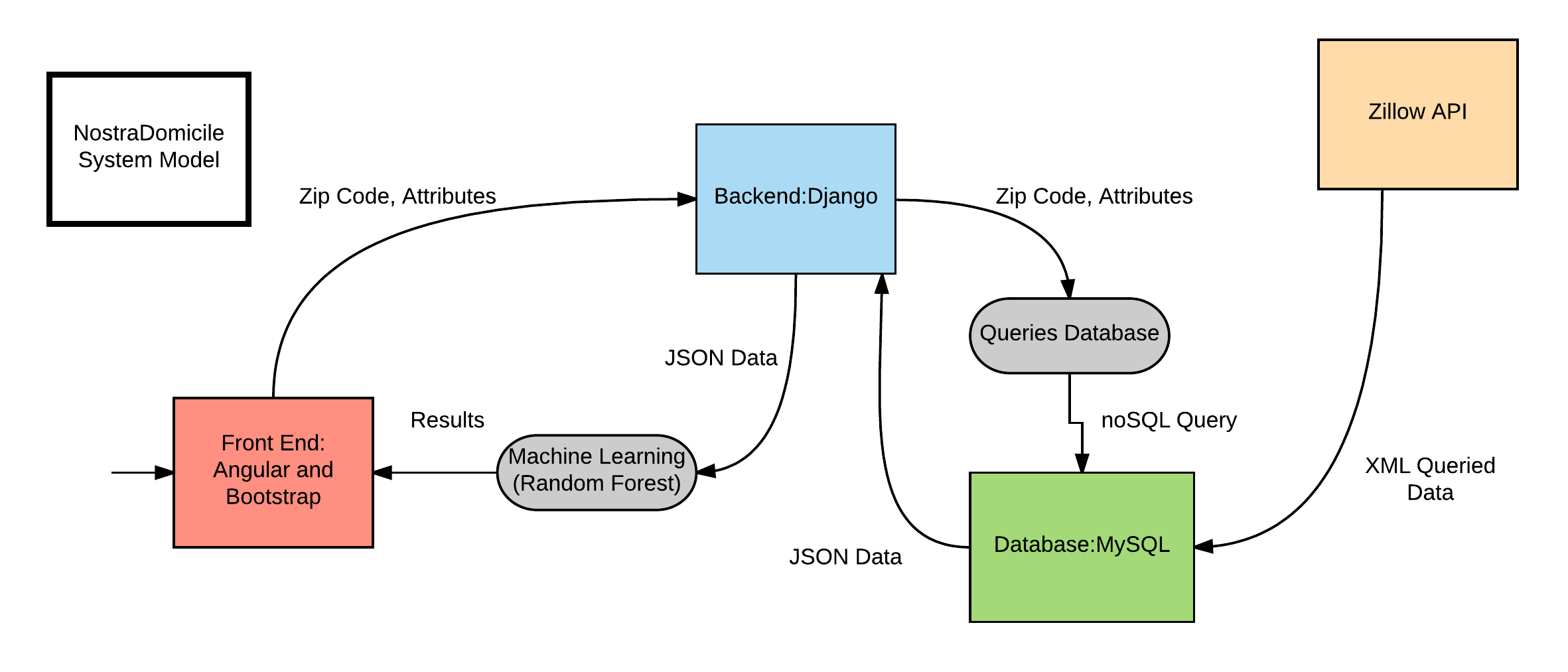


Fig.1

**System Model (New/Modified):**

The profile of the system(Fig.1) is a web application comprised of a database, backend, frontend, machine learning algorithms, and a cloud hosting service. The figure above shows the data flow of the two primary functions of our project. The user can enter either a zip code or a combination of a zip code and their own home’s attributes in the front end of the web application, this information will be sent to the backend of the website, which will query the database. After creating a data frame comprised of home data from a certain zip code or attributes, the frame will be run through machine learning algorithms via the backend, and the results will be displayed on the front end of the website. An alternative to this model is shown below(Fig.2). Instead of offering predictions and data visualizations based on statistical analysis, this model is based solely on statistical analysis. We chose the model with machine learning as our two primary functions are prediction based and the random forest algorithm lends itself perfectly to binary classification and the return of most important factors relating to that classification. This alternative model also contains different frameworks and programming languages. We decided upon our model containing a front end built with Bootstrap and AngularJS, backend framework with Django, machine learning in Python, and a MySQL database because we felt that as a team we had more experience with these languages and frameworks and we would have a higher chance of success in fulfilling the tenants of the project if we adhered to their usage.

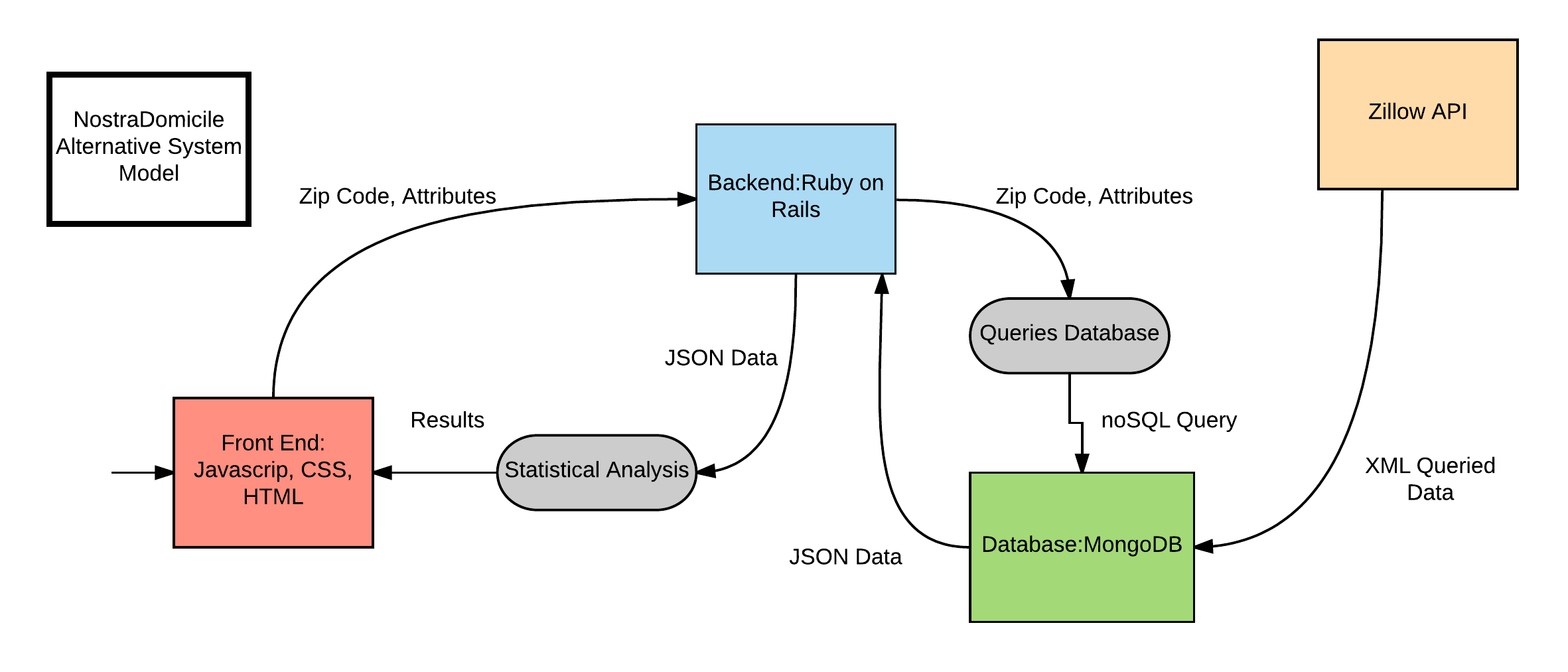


Fig.2

(Changed: Added link to Zillow API.)

Pros and Cons of each primary model (Random forest, Django, MySQL):

|  |  |
| --- | --- |
| Pros | Cons |
| Front end (Angular/Bootstrap): More organized, write less, easier to manipulate DOM, consistent and responsive. | Front end (Angular/Bootstrap): Not as much experience, have to include dependencies of both frameworks. |
| Back end (Django): Would allow us to use python for both the backend and machine learning. | Back end (Django): No experience with django, but we do have experience in python. |
| Hosting (Cloud): Easier to scale, and we would have support staff to help if we ran into problems. | Hosting (Cloud): Cost money, and requires internet access. |
| Database (MySQL): Several members have experience. | Database (MySQL): Doesn't scale as well across multiple servers as NoSQL would. This would be a problem later if we were to try to turn our project into an actual service/website. |
| Machine Learning (Random Forest): Some members have experience. | Machine Learning (Random Forest): A large number of trees may make the algorithm slow for predictions. |

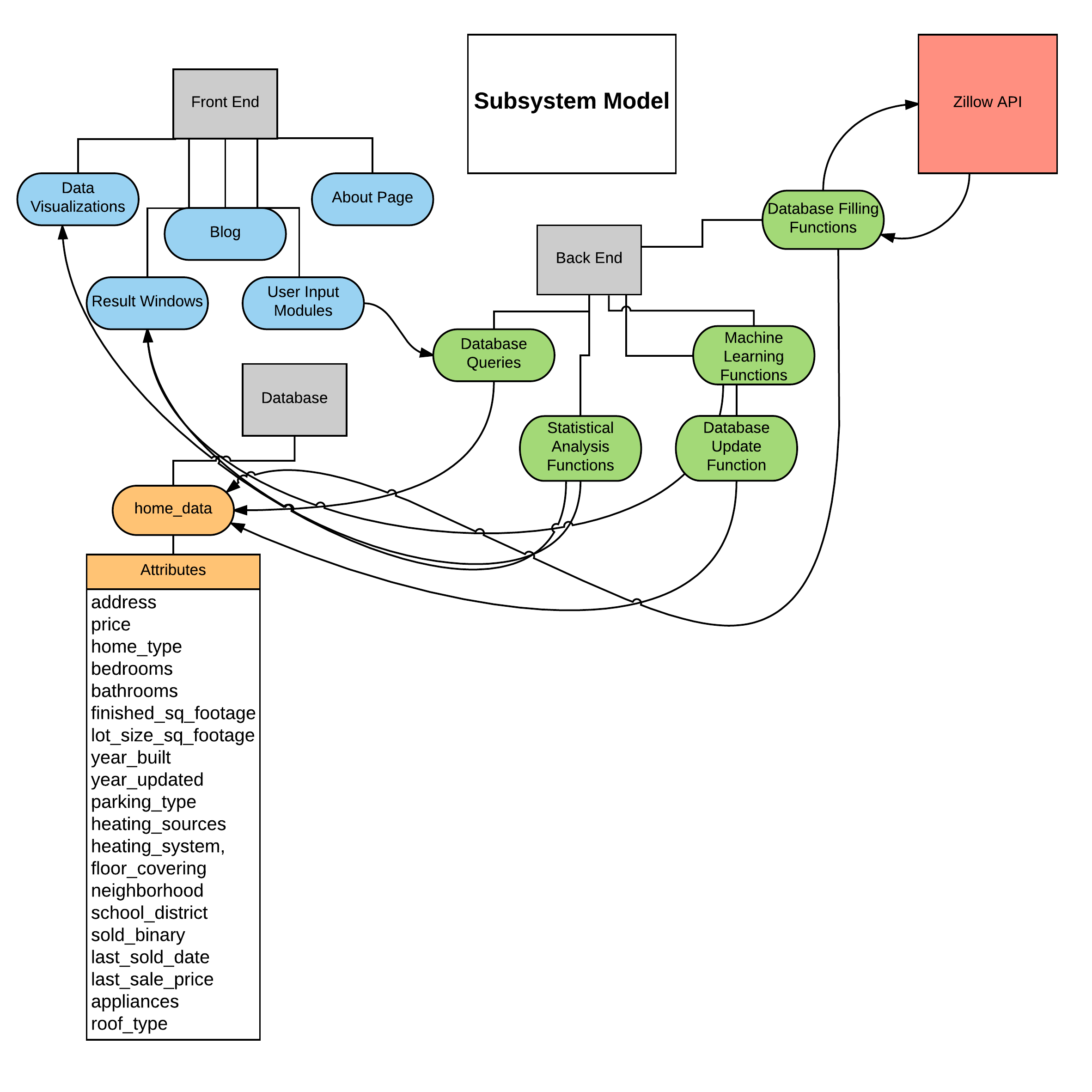
Pros and Cons of alternative model (Statistical analysis, Ruby on Rails, MongoDb):

|  |  |
| --- | --- |
| Pros | Cons |
| Front end (HTML/CSS): No, or not as many, dependencies. Can write only what we need instead of following MVC. | Front end (HTML/CSS): Not as responsive, not as easy to manipulate DOM as a framework may be. |
| Back end (Ruby on Rails): Some experience with this framework. | Back end (Ruby on Rails): Wouldn't as easily tie in with machine learning, because it uses separate languages. |
| Hosting (Localhost): More control, free, and more security. | Hosting (Localhost): Might have a problem displaying/presenting project. Limited to each person's computer, so not as freely available should we not have said computers available. |
| Database (MongoDB): Scales better across multiple servers if we needed scaling at a later date. | Database (MongoDB): No experience with NoSQL databases. |
| Statistical Analysis: We don't have to use python, so overlap of languages doesn't become a factor in backend language choice. | Statistical Analysis: We don't have as much experience with statistical analysis. |

(Changed: Added pros/cons list.)

**Subsystems Model (New/Modified):**

The figure below(Fig.3) represents the subsystems of the major components of the system. The front end of the website will be built using Bootstrap and AngularJS to provide the user with an interface using input modules(text areas, drop-down boxes, and checkboxes), result windows, data visualizations as well as a real estate experts’ blog and about page. The backend of the web application will be developed using Django for the framework, python for the machine learning and statistical analysis functions, and a database update function that calls the Zillow API to maintain the currency of the database. The database will be contain a list of approved users and an administrator, and tables separated by zip code and having the following attributes: address, price, home type, bedrooms, bathrooms, finished sq footage, lot size sq footage, year built, year updated, parking type, heating sources, heating system, floor covering, neighborhood, school district, sold binary, last sold date, last sale price, appliances, roof type.



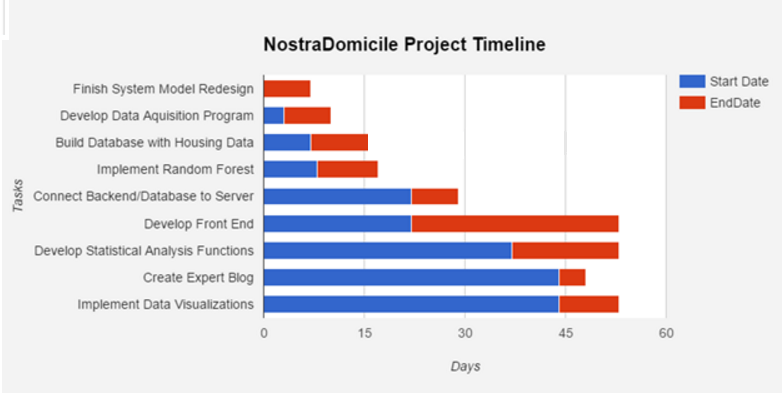
(Changed: Removed users table from database, because we are not storing user information. Also added link to Zillow API.)

Fig. 3

**Responsibilities:**

* Christian: Database/Backend
  + Ochaun: Machine Learning/Front End
* Richard: Backend
* Jeremy: Frontend

**Project Timeline (New/Modified):**



(Changed: Shortened "Build Database with Housing Data" EndDate.)

**Feasibility (New/Modified):**

The project timeline sets out overlapping periods for several major tasks in the project. Although we anticipate that some tasks may be more complicated or difficult than originally anticipated, we don’t believe that all aspects of each task on the timeline will be dependent on preceding tasks in the timeline. Several portions of the project can be worked on in parallel, so a slowdown in one particular area should not cause an across the board delay. Each task in the timeline includes preliminary testing for the portion worked on, so additional time for testing shouldn’t be necessary aside from final testing on the completed application. Overall, we believe that, as long as we don’t see across the board delays in every subtask, the project is feasible on the timeline we’ve set out.

Risk assessment:

|  |  |
| --- | --- |
| Risk | Possible Solution |
| Hitting the max number of API requests. | We can request friends and family members make Zillow accounts in order to use their API keys, at least until our database is fuller. |
| Cost of the server | This is covered by the github student package. If the cost exceeds what the package has allocated, we can then decide upon how we will divvy up the cost. |
| Not having enough time for features. | We will prioritize the features we're working on based on the tier list mentioned above, and understand going in that some features in the third tier will not get finished. |
| One person falling behind on their work and delaying other people's features. | Like mentioned above, most features can be done in parallel. However if there are some that can't, then as we finish features and people have free time, we will help each other with their responsibilities where appropriate. |
| Server downtime | We will work around any unlikely downtime using our localhost. If it begins to become a problem, we can switch hosting providers. |

(Changed: Added risk assessment table.)

**Data Dictionary (New Section):**

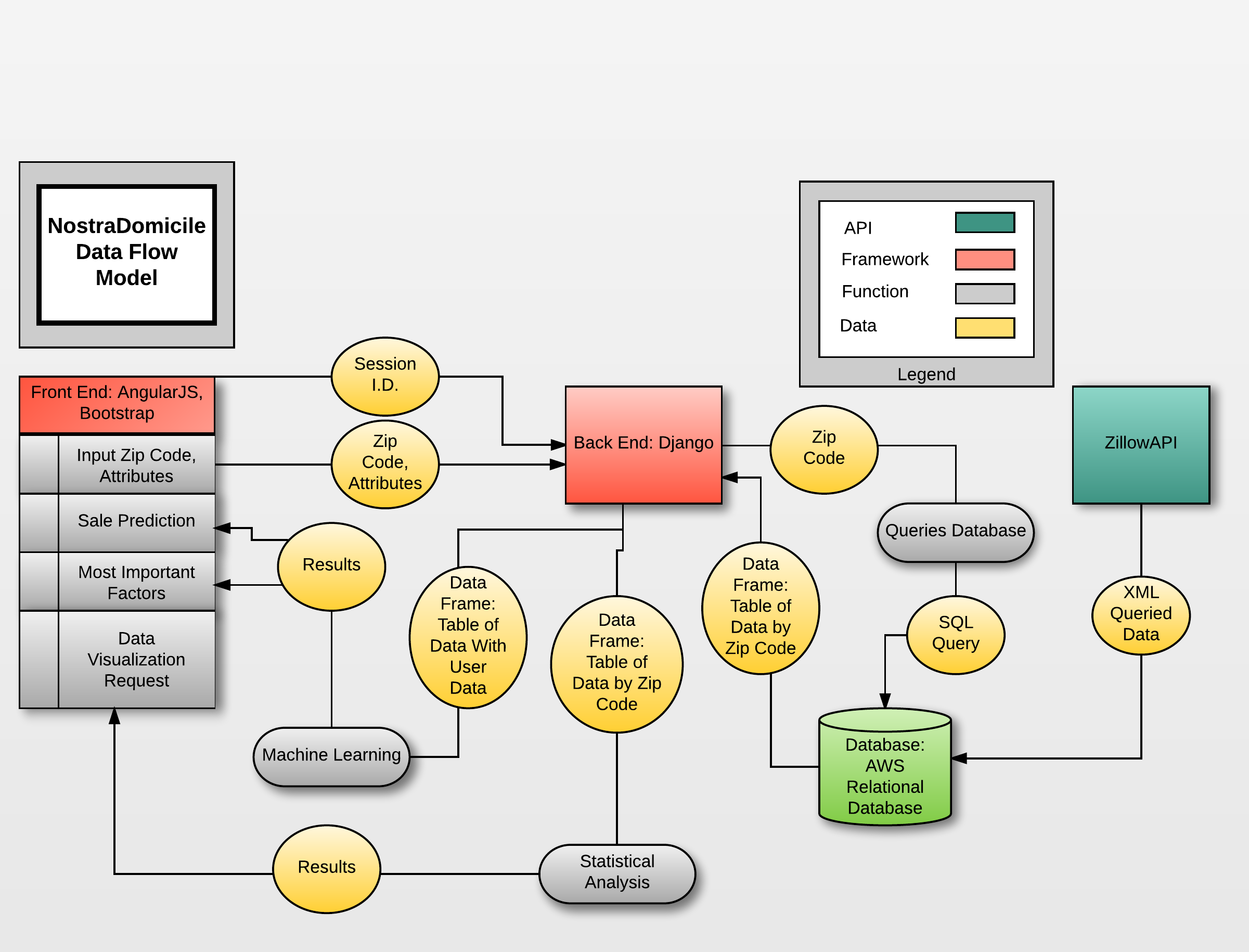
* Table with variables and their descriptions used in this project:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute Name** | **Optional/Required** | **Attribute Type** | **Description** | **Database Table** |
| street\_address | Required | Text | Address of individual property for lookup. Required - used as primary key for home in database and required for Zillow API call. | PyZillow\_Data.home\_data |
| zip | Required | Integer | Zip code value for the given property, used as secondary index attribute. Required for Zillow API call and for machine learning predictions. | PyZillow\_Data.home\_data |
| city | Optional | Text | City where the property is located. Not required in case a null value is received and for homes outside municipalities. | PyZillow\_Data.home\_data |
| state | Optional | Text | State where the property is located. Not required in case null value returned. | PyZillow\_Data.home\_data |
| home\_type | Optional | Text | Description of the type of property at the given address (e.g., single-family, multi-family, raw land). | PyZillow\_Data.home\_data |
| bedrooms | Optional | Integer | Number of bedrooms for the given property. | PyZillow\_Data.home\_data |
| bathrooms | Optional | Integer | Number of bathrooms for the given property. | PyZillow\_Data.home\_data |
| finished\_sq\_footage | Optional | Integer | Total amount of finished square footage for the given property. | PyZillow\_Data.home\_data |
| lot\_size\_sq\_footage | Optional | Integer | Size of the lot on which the given domicile is located, in square feet. | PyZillow\_Data.home\_data |
| year\_build | Optional | Integer | Year a dwelling or improvement on the property was built, if any. | PyZillow\_Data.home\_data |
| year\_updated | Optional | Integer | Year a dwelling or improvement on the property was updated, if any. | PyZillow\_Data.home\_data |
| number\_of\_floors | Optional | Integer | Number of floors of any dwelling on the property. | PyZillow\_Data.home\_data |
| parking\_type | Optional | Text | Description of available vehicle parking or storage structures on the property. | PyZillow\_Data.home\_data |
| heating\_sources | Optional | Text | Description of the heating source used by the heating system in the dwelling on the property. | PyZillow\_Data.home\_data |
| heating\_system | Optional | Text | Description of the heating system used to heat the dwelling on the property. | PyZillow\_Data.home\_data |
| floor\_covering | Optional | Text | Description of the types of floor coverings in the dwelling, all types used in the dwelling are included in the description. | PyZillow\_Data.home\_data |
| number\_of\_rooms | Optional | Integer | Total number of rooms of any dwelling on the property. | PyZillow\_Data.home\_data |
| neighborhood | Optional | Text | Neighborhood in which the property is located, if any (NOT street). | PyZillow\_Data.home\_data |
| school\_district | Optional | Text | Public school district in which the property is located, if any. | PyZillow\_Data.home\_data |
| sold\_binary | Required | Integer | Required binary value indicating whether the property has been sold in the past year. | PyZillow\_Data.home\_data |
| last\_sold\_date | Optional | Text | Last date on which the property was sold, if any (as maintained by Zillow). | PyZillow\_Data.home\_data |
| last\_sale\_price | Optional | Decimal(10,2) | Last price at which the property was sold, if any (as maintained by Zillow). | PyZillow\_Data.home\_data |
| appliances | Optional | Text | A description/list of appliances included with the property for the purpose of promoting its sale. | PyZillow\_Data.home\_data |
| roof\_type | Optional | Text | The type of roofing material covering the dwelling. | PyZillow\_Data.home\_data |
| room\_types | Optional | Text | A description of rooms/areas in the dwelling for the purposes of promoting sale. | PyZillow\_Data.home\_data |
| updated\_properties | Required | Integer | Required binary value used to indicate whether certain additional property details were available from the Zillow API. | PyZillow\_Data.home\_data |
| session\_id | Required | Integer | Required value to ensure user data is kept throughout the session. | N/A |
| targ\_zip\_code | Required | Integer | Required value to select zip code for comparison/data visualization | N/A |
| usr\_sq\_foot | Optional | Integer | User input square footage for attribute based predictions about home sale. | N/A |
| usr\_lot\_size | Optional | Float | User input square footage for lot for predictions. | N/A |
| usr\_bedrooms | Optional | Integer | User input number of bedrooms for predictions. | N/A |
| usr\_bathrooms | Optional | Integer | User input number of bathrooms for predictions | N/A |
| usr\_year\_built | Optional | Integer | User input build year for dwelling for predictions. | N/A |
| usr\_stories | Optional | Integer | User input number of stories for predictions. | N/A |
| usr\_neighborhood | Optional | Text | User input neighborhood for predictions. | N/A |
| usr\_school\_district | Optional | Text | User input school district for predictions. | N/A |
| usr\_home\_type | Optional | Text | User input home type for predictions. | N/A |
| usr\_parking\_type | Optional | Text | User input parking type for predictions. | N/A |

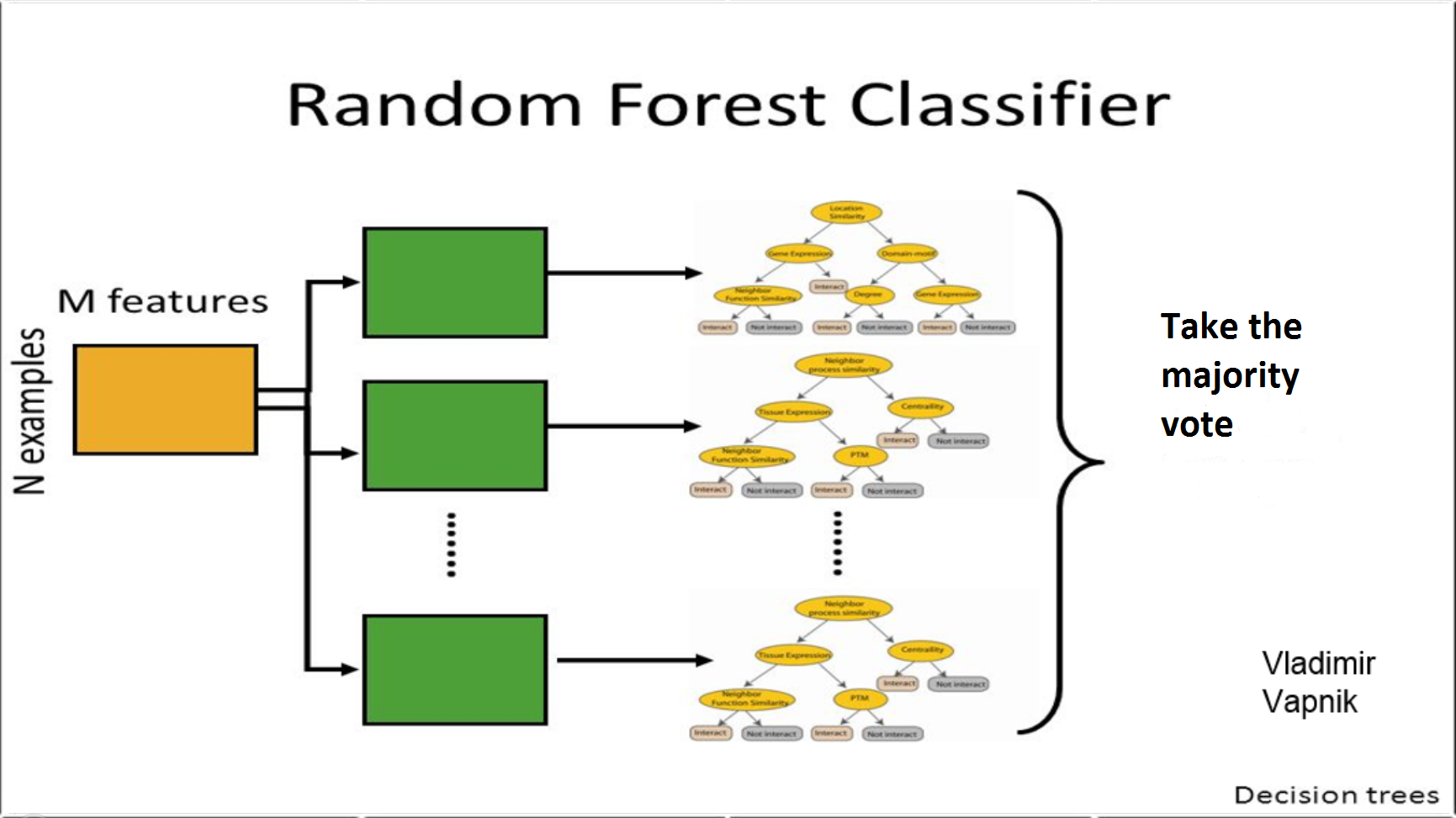
* Home data from Zillow API stored in database table home\_data in PyZillow\_Data schema. Includes:
  + Home attributes, for example, address (PK), zip code (secondary index attribute), last sale date, last sale price, and number of bed/bathrooms.
  + Values showing additional details were available and if a house "sold" for use in sale predictions.
* Data from user sessions:
  + Session ID
  + Attribute filters for sale predictions and data visualizations.

**Data Model (New Section):**

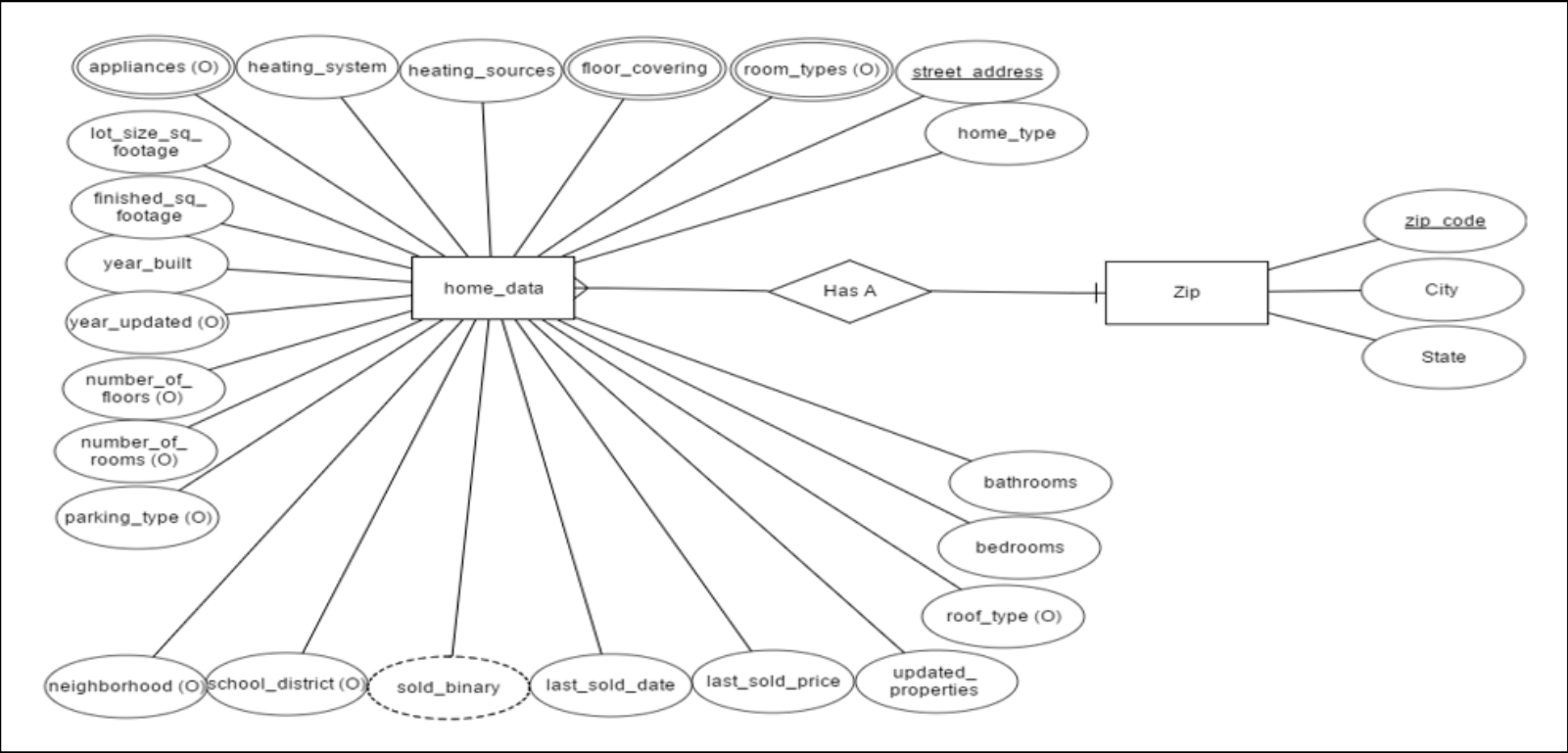
1. For our tables and data structures, we will mainly be using one table (home\_data). This table will be used to store all the information that we retrieved from the Zillow API. This is also the table that we will access to achieve our features. As far as data structures, we will only be using data frames for the machine learning (and perhaps the statistical analysis for analyzing results, if we get to that feature).
2. The data I/O based on the system and sub-system models have the following look:
   * The overall system:



* + The Random Forest machine learning subsystem:



1. Entity-Relationship model:



**System Analysis (New Section):**

1. Analysis of system space and time complexity (O() notation):
   1. Database
      1. Uses a b+-tree index on primary key "address", log(n) to search. [1]
      2. Insertions:
         1. Time - ~~Lookup to avoid error from existing PK (log(n)) + API call (constant "C") + insertion into DB (log(n) + C) = 2(log(n) + C) = O(log(n)).~~ It takes constant time to get data from the API, so the complexity ends up being O(log(n) + nC) = o(n).
         2. Space - Only current index node keys are stored in memory (say, "m" keys per node) + record for insertion (C ) = m + C = O(m).
      3. Lookups:
         1. Time - ~~For a single record, index search (log(n)) + transfer to memory (C ) = log(n) + C = O(log(n)). For multiple records, still O(log(n)) since C\*(log(n)) = O(log(n)).~~ Using a index search has an O(log(n)) complexity, there is no transfer to memory.
         2. Space - Current index keys (m) + PK of record sought (1) = m + 1 = O(m).
   2. Random Forest
      1. Algorithms:
         1. Supervised learning as opposed to online learning.
         2. Built with a multitude of Decision Trees.
         3. Bootstrap sampling.
         4. Averaging tree results to reach a predicted classification.
      2. Time and Space Complexity of Algorithms:
         1. Time - O(M(mn log n))
         2. Space - O(mn log n)
         3. Where M=number of trees, m=number of features, and n=number of elements in the dataset. [2]
         4. This is an estimate, but our algorithm will likely be faster due to processing trees in parallel.
   3. Overall System
      1. Time complexity of database query: O(log n)
      2. Time complexity of Random Forest: O(M(mn log(n)))
      3. O(M(mn log n)) + O(log(n))
      4. Random Forest dominates
      5. Time complexity of system: O(M(mn log(n)))
2. Indexing Attributes: Primary key on street address. There is also a secondary index on zip code.

**References (New Section):**

1. Silberschatz, Abraham, et al. *Database Systems Concepts*, 6th Ed. P.490.
2. Vens, Costa, *Random Forest Based Feature Induction, Data Minin*g, 2011.