Florida International University
School of Computing and Information Sciences

Software Engineering Focus

Final Deliverable

Path of Least Resistance (PoLR): Commute Time Property Searching for BreazeHome

Team Members: Jose Sobalvarro, Juan Garcia-Torres, Robert J. Fortunato Jr., Xu Li

Product Owner(s): Dr. Masoud Sadjadi

Mentor(s): Aaron Feng

Instructor: Dr. Masoud Sadjadi

Abstract

This document presents the information necessary to gain a good understanding of how the Path of Least Resistance (PoLR) commute time search feature operates in the BreazeHome system. PoLR is a novel search filter and algorithm for merging classic property search criteria with commute time search criteria into one search via BreazeHome. The document includes a discussion of the business case, feature significance, technical implementation information, project execution logistics, and system documentation. The aim of the PoLR project was to create a software feature for the existing BreazeHome platform that would bring novel, value add functionality that allows users to find properties by filtering the data based on commute times to a given location. The feature has been implemented into the main BreazeHome desktop user interface and supporting backend API. It leaves for future integration into the mobile user interface and partner systems by virtue of design and implementation.

Table of Contents

INTRODUCTION

Current property purchase and rental web applications, including BreazeHome, include search criteria for finding candidate properties based on price, floor plan layout, housing type, and general neighborhood vicinity. They do not include results based on time and location specific and variant commute times. This makes the commute time dimension of a property search an afterthought of the search process, if considered at all. According to an analysis by the U.S. Census Bureau, the average one-way commute times for U.S. residents is 25.5 minutes with metropolitan area commute times being substantially higher than the average [1]. Moreover, strong and substantial correlations have been established between happiness and shorter commute times [2]. This makes commute time to reach a destination a consideration that should be taken into account when a user is going to buy or rent a property. As a result, this also makes having the feature as part of a set of property search filters a business requirement for BreazeHome.

Current System

Prior to the onset of the PoLR project, BreazeHome suffered from the same shortcomings that current competing web applications for property purchase and rental searching suffer from: a search without consideration for the relationship between the property and commute times to frequent destinations. Zillow and Trulia are not as lacking in functionality as BreazeHome and others. They offer a feature providing average commute times from a location or property to surrounding areas; however, the estimated averages are not time variant and are subject to large variances in actual commute time based on the time of the day and day of the week a commute occurs. Google Maps provides for very accurate commute times between destinations for a give time of the day and day of the week; however, Google Maps is not juxtaposed against property searches and systems. This leaves the commute time data collection and analysis as a series of time consuming iterative research steps encumbered upon the buyer following the identification of a candidate property.

Purpose of New System

The PoLR commute time search allows any BreazeHome user to shop for properties based on commute time from the property to an arbitrary destination given a commute time, commute day, commute duration, and supported mode of transport. PoLR is not a mutually exclusive search function for BreazeHome given BreazeHome's current capabilities. Rather, it is a smart search feature that supplements current search filters as well as planned future search features. In the context of the new system feature, the PoLR end user experience is delivered the classic desktop

UI. The desktop UI is supported by a reusable BreazeHome API that can furnish the same, consistent PoLR services to the mobile UI and/or future business partner systems wishing to leverage the capabilities that differentiate BreazeHome from its competitors.

USER STORIES

Implemented User Stories

User Story Name: Search Properties By Commute Time – Desktop UI (#682)

Description:

As a user of BreazeHome and the Desktop UI, I would like to find properties that are within a given commute time by car from an address I specify at a time of the day during a day of the week so that I can narrow down my candidate property choices to those that will not encumber me with excessive commute times pursuant to the maximum time I am willing to spend driving.

Acceptance Criteria:

1. The system must take a city, landmark, or address search string as input and use it as the commute destination.
2. The system must allow for filtering on based on all of the following, together:
    a. A day of the week.
    b. A time of the day.
    c. A maximum allowable commute time.
3. The search output of properties reachable in a given time/day in less than or equal to the commute time must be displayed as the results of the search.
4. For incorrect or incomplete search filters/criteria, the system must display an error message on the UI to indicate the specific actions a user should take to remediate the problem.
5. The system must clear the error message upon successful application of corrected filters and search criteria.
6. The implementation must comply with BreazeHome security, performance, usability, and stylistic requirements as defined by the project.

Related Tasks:

Number	Name	Status
702	Integrate, test, and validate all component implementations; fix bugs as discovered.	Done
701	Implement backend API functionality for isochrone polygon search filters via GIS queries to the database.	Done
700	Implement API/Web Service wrapper to avoid Cross Site Scripting Errors from CORS.	Done
699	Implement isochrone generation algorithm.	Done
698	Implement future date/time calculations.	Done
697	Implement address geocoding service wrapper.	Done
696	Write front end code to collect and validate PoLR search input parameters.	Done
695	Design the frontend GUI for gathering the input parameters.	Done
694	Analyze geospatial geometry points in BH database and establish database work needed.	Done
693	Set up front and back end services on BreazeHome development server.	Done

User Story Name: Search Properties by Commute Time via Centralized API (#692)

Description:

As a Developer of BreazeHome and the various UI's, I would like to have the commute time search centralized into a shareable API such that users can find properties from an address at a time of the day during a day of the week via multiple different UI's without having to maintain multiple code bases for the functionality.

Acceptance Criteria:

1. The current BreazeHome Desktop UI commute time search functionality must continue to work as is.
2. The current BreazeHome Mobile UI must display the exact same results the Desktop UI does.
3. A partner/client non-BreazeHome UI should display the exact same results.

4. A direct API call must produce the data output used by various UIs.
5. The API must be usable by other current and future subsystems in the Breaze ecosystem, both internal and external.
6. The implementation must comply with BreazeHome security, performance, usability, and stylistic requirements as defined by the project.

Related Tasks:

Number	Name	Status
719	Integrate, test, and validate all component implementations; fix bugs as discovered.	Done
717	Document API Methods	Done
716	Implement isochrone generation algorithm.	Done
714	Implement address geocoding service wrapper.	Done
713	Remove current frontend/backend PoLR implementation code and reroute frontend calls to new backend.	Done

User Story Name: Non-Vehicular Commute Time Searching - Desktop UI

As a user of BreazeHome and the Desktop UI, I would like to use the commute time search in conjunction with non-vehicular travel modes to find properties from an address I specify at a time of the day during a day of the week so that I can narrow down my candidate property choices to those that will not encumber me with excessive commute times using travel modes that aren't vehicle based.

Acceptance Criteria

1. The current commute time search functionality must continue to work as is.
2. New options for selecting modes of travel must be added. The new modes of travel must be as follows:

1. Bicycling
2. Walking

3. The search output of properties reachable in a given time/day in less than or equal to the commute time must be displayed as the results of the search.
4. The Mobile UI should display the exact same results the Desktop UI does.
5. The implementation must comply with BreazeHome security, performance, usability, and stylistic requirements as defined by the project.

Use Case

- Name: Transit type commute search
- Actor: User
- Precondition:
    1. User has navigated to results page.
- Description <Flow of Events>:
    1. Use case starts when the user has navigated to the results page and pulls down more filters
    2. User can then fill out the commute search requirements and submit
    3. The system returns properties on results page according to specified search criteria.
- Alternate Flow of Events
    1. User does not select a transit type
    2. User gets a message to fill in missing attribute

User Story Name: Dynamic Filter Stacking over Commute Type

As a user of BreazeHome and the Desktop UI, I would like to use the commute time search in conjunction with any other available filter to find properties from an address I specify at a time of the day during a day of the week so that I can narrow down my candidate property choices to those that will not encumber me with excessive commute times and meets other expected criteria.

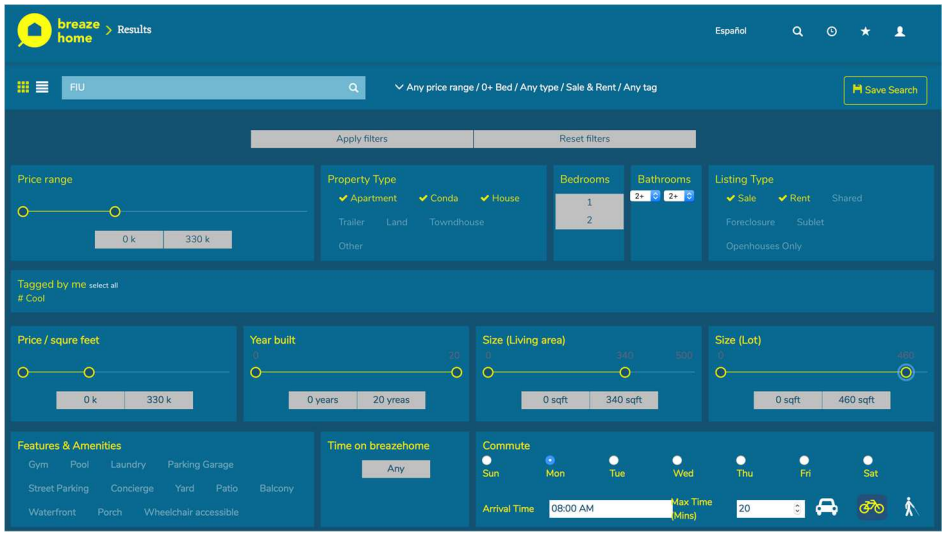
Acceptance Criteria

1. The current commute time search functionality must continue to work as is.
2. Any other filter property available to the user must be able to be combined with commute time search
3. The search output of properties reachable in a given time/day in less than or equal to the commute time must be displayed as the results of the search.
4. The Mobile UI should display the exact same results the Desktop UI does.
5. The implementation must comply with BreazeHome security, performance, usability, and stylistic requirements as defined by the project.

Use Case

- Name: Transit type commute search with multiple filters
- Actor: User
- Precondition:
    1. Use case starts when the user has navigated to the results page and pulls down more filters
    2. User can then fill out the commute search requirements with multiple filters and submit
    3. The system returns properties on results page according to all specified search criteria.

- Alternate Flow of Events
    1. User does not select a transit type
    2. User gets a message to fill in missing attribute



User Story Name: Generate a layer on the map based on the commute time search (#666)

Description:

· As a user, I would like to visualize the whole area reachable given a specific time and destination, so that I can select the desired property easily.

Acceptance Criteria:

1. Show the destination point as a pointer in the map

2. Display an area layer that shows the distance that the user can reach in the filtered time

Use Case

· Name: Generate a layer on the map based on the commute time search

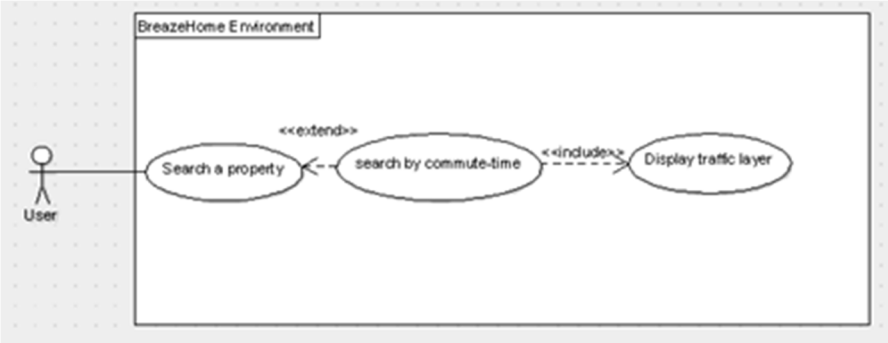
· Actor: User

· Preconditions: The user is on the map page, he has already inserted the parameters and click on apply filters.

· Description

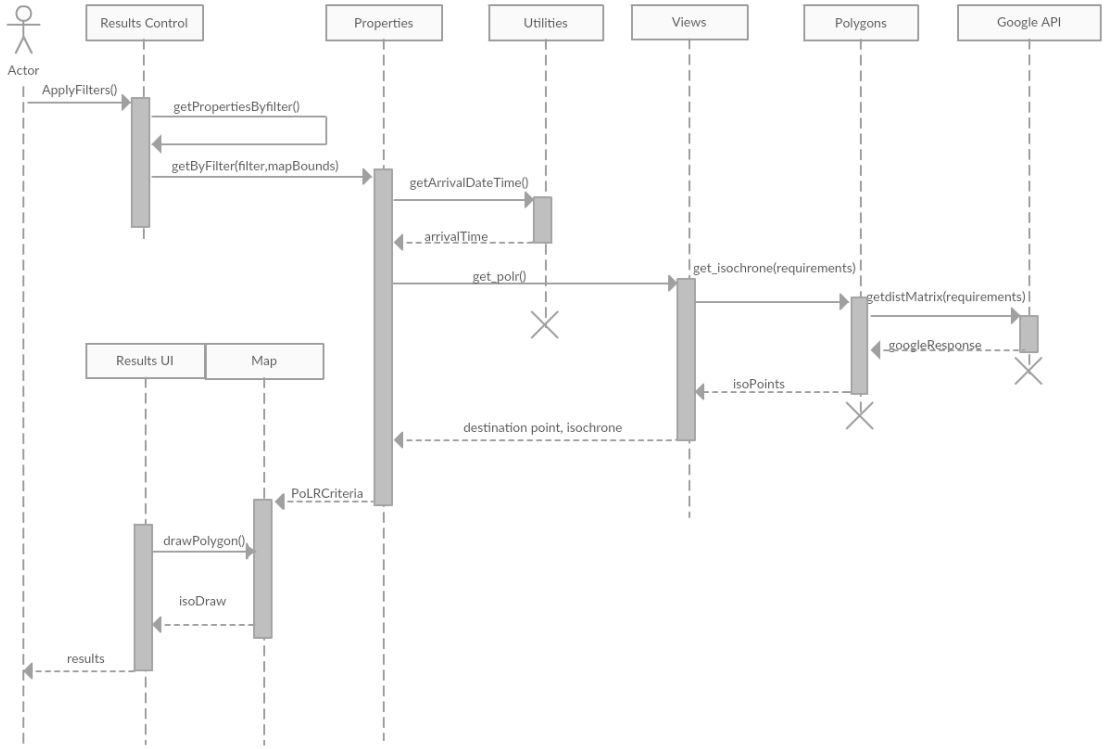
1. Use case starts when the user has navigated to the results page and pulls down more filters
2. User can then fill out the commute search requirements and submit
3. System retrieves the results of the commute time search as tiles and also it shows the map layer with the parameters specified.

Use Case Diagram



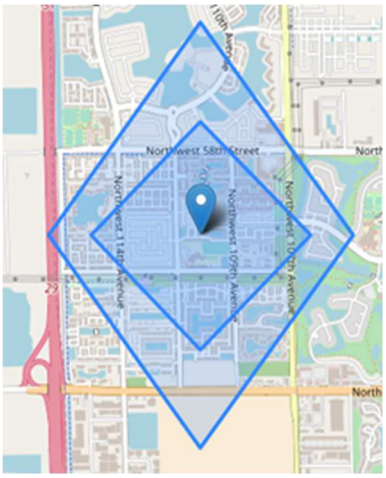
Use case for story #666

Sequence Diagram



Sequence Diagram for user story #666

Visual user deliverable



User visual deliverable for story #666

User Story Name: Generate a multi-point layer on the map based on the commute time search (#685)

Description:

- As a user, I would like to reflect a multiple-point layer on the map so that I can visualize the area where I can select properties easily

Acceptance Criteria:

1. Show the destination point on the map
2. Generate a multiple-point polygon on the map that shows the distance that the user can reach given the commute-time input parameters

3. The polygon has to have more than four points (based on the feedback received in the show and tell)

4. The traffic layer operation must be adapted in the BreazeHome platform

Use Case

· Name: Generate a multi-point layer on the map based on the commute time search

· Actor: User

· Preconditions: The user is on the map page, he has already inserted the parameters and click on apply filters.

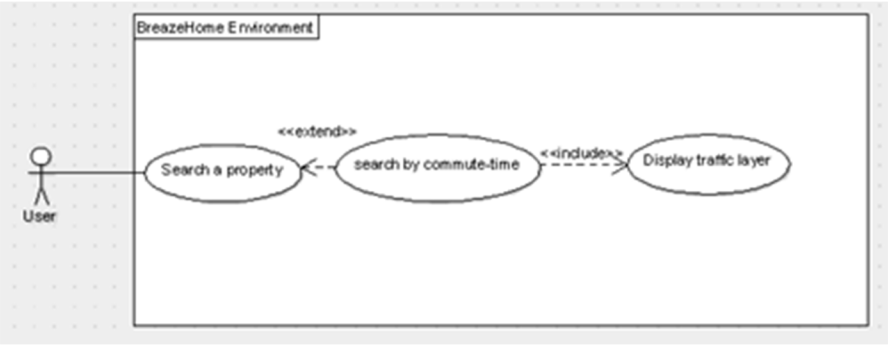
· Description

1. Use case starts when the user has navigated to the results page and pulls down more filters

2. User can then fill out the commute search requirements and submit

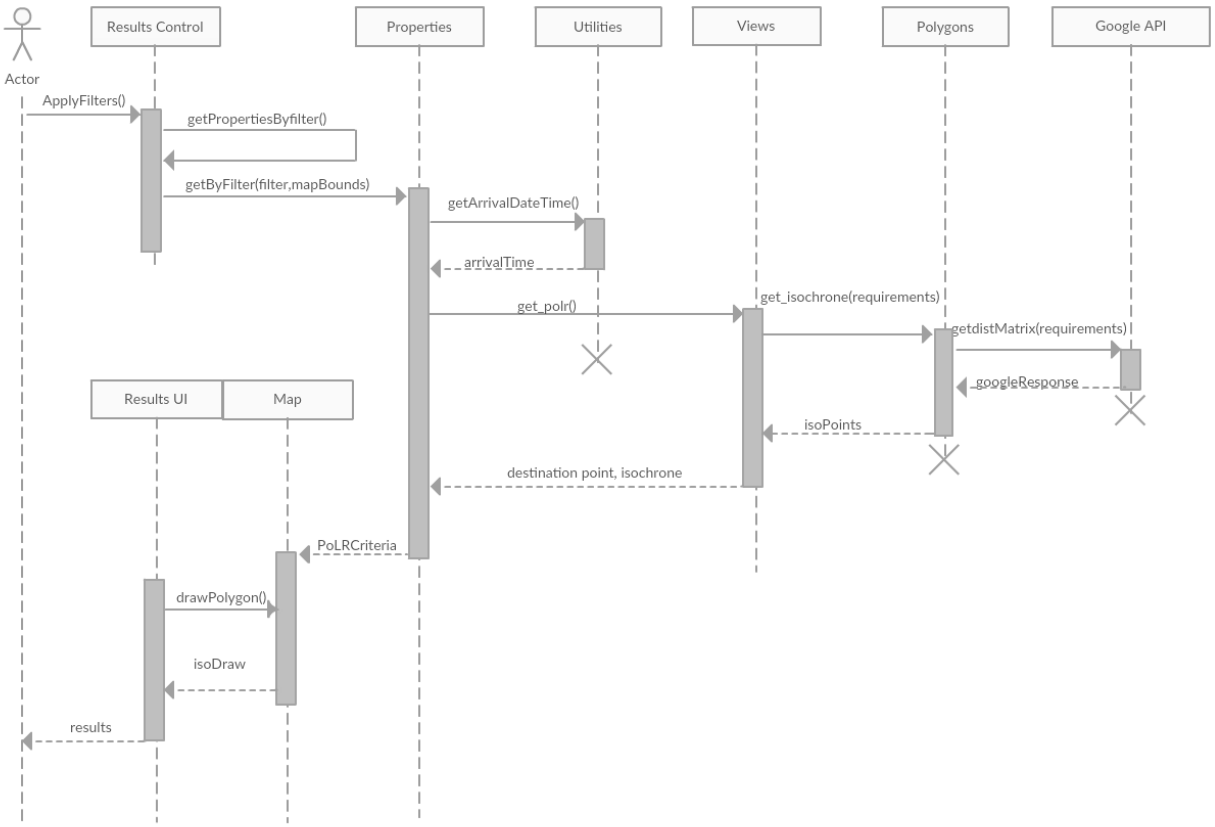
3. System retrieves the results of the commute time search as tiles and also it shows the map layer with the parameters specified.

Use Case Diagram



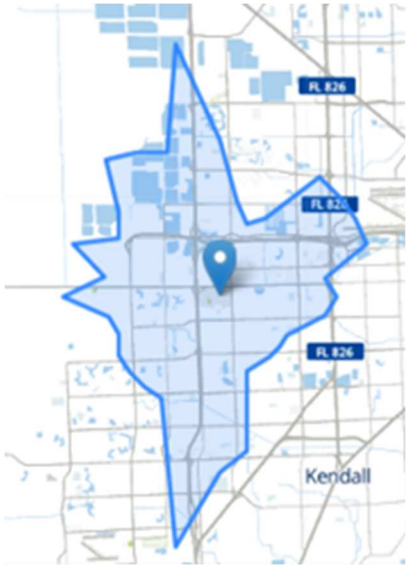
Use case for story #685

Sequence Diagram



Sequence diagram for user story #685

Visual user deliverable



User visual deliverable for story #685

User Story Name: Show the results of the commute time filter on the map (#712)

Description:

As a user, I would like to visualize the results of the commute time filter on the map so that I can select the desired property according to my requirements

Acceptance Criteria:

1. Show the destination point on the map
2. Show the commute time layer on the map
3. Show the properties obtained by the commute-time filter on the map

Use Case Diagram

· Name: Show the results of the commute time filter on the map

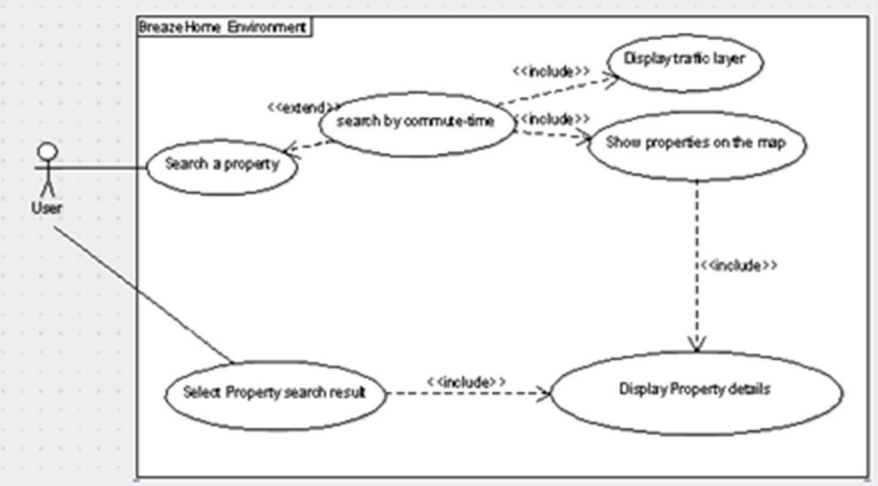
· Actor: User

· Preconditions: The user is on the map page, he has already inserted the parameters and click on apply filters.

· Description

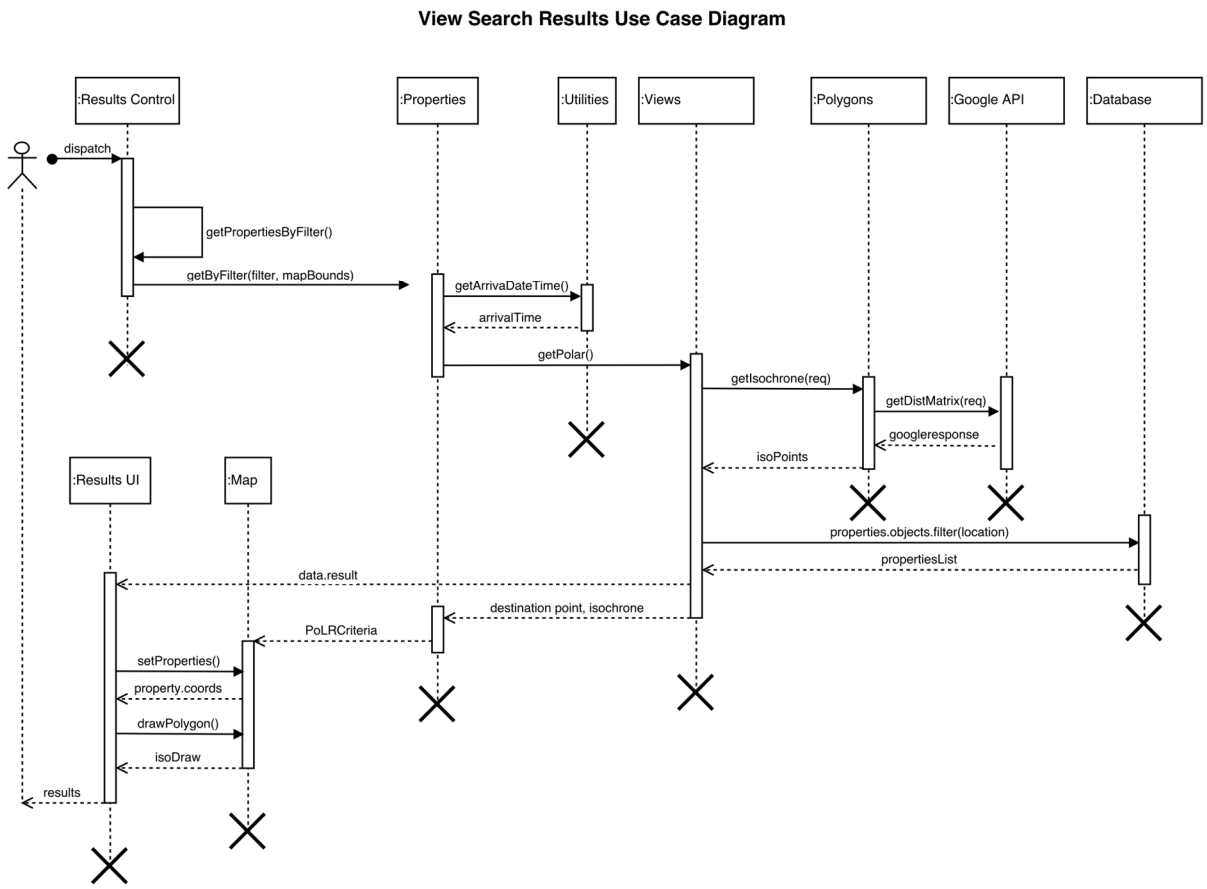
1. Use case starts when the user has navigated to the results page and pulls down more filters
2. User can then fill out the commute search requirements and submit
3. System retrieves the results of the commute time search, showing the map layer with the properties inside and the tiles from the properties aside the map.

Use Case Diagram



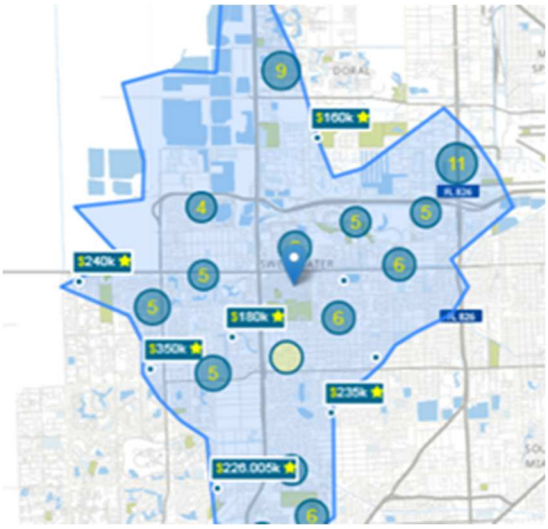
Use case for story #712

Sequence Diagram



Sequence diagram for user story #712

Visual user deliverable



User visual deliverable for story #712

User Story Name: Manipulate commute time search filter settings on admin console (#718)

Description:

As the administrator of BreazeHome, I would like to manipulate the variables settings related to the commute time search filter, so that I can adjust the outcome of the function of the commute time search

Acceptance Criteria:

1. Create text inputs for duration tolerance, search angles, API key, geocode URL, search iterations, maximum allowable miles per minute and matrix elements per second and store them.

2. The PoLR functionality must read and use the settings specified in the admin UI

3. Get the values of the inputs and send them to settings.py as variables.

4. The implementation must comply with BreazeHome security, performance, usability, and stylistic requirements as defined by the project.

Use Case Diagram

· Name: Manipulate commute time search filter properties on admin console

· Actor: User

· Preconditions: The user is on the results page, he is already logged in and he clicked on the manage button

· Description

1. Use case starts when the admin logs in BreazeHome environment

2. Admin goes to the result page and pulls down the user options

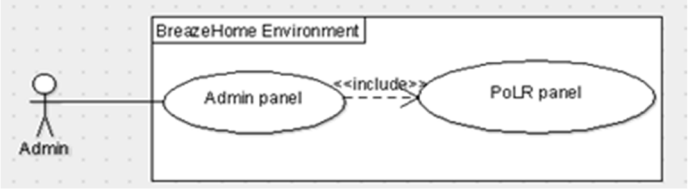
3. Admin clicks on manage

4. Admin changes PoLR variables on his panel

Alternative flow of events:

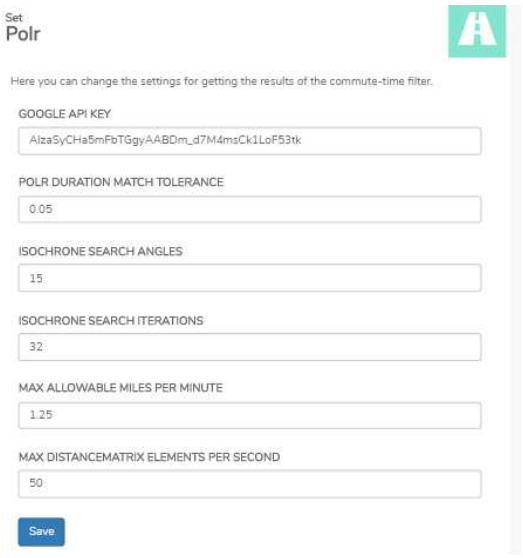
1b. Admin goes to the results page and logs in from there.

Use Case Diagram



Use case for story #718

Visual user deliverable



User visual deliverable for story #718

Pending User Stories

User Story Name: Search Properties By Commute Time – Mobile UI (#689)

Description:

As a user of BreazeHome and the Mobile UI, I would like to find properties that are within a given commute time by car from an address I specify at a time of the day during a day of the week so that I can narrow down my candidate property choices to those that will not encumber me with excessive commute times pursuant to the maximum time I am willing to spend driving.

Acceptance Criteria:

1. The system must take a city, landmark, or address search string as input and use it as the commute destination.
2. The system must allow for filtering on based on all of the following, together:
    a. A day of the week.
    b. A time of the day.
    c. A maximum allowable commute time.
3. The search output of properties reachable in a given time/day in less than or equal to the commute time must be displayed as the results of the search.
4. The Mobile UI should display the exact same results the Desktop UI does.
5. The implementation must comply with BreazeHome security, performance, usability, and stylistic requirements as defined by the project.

User Story Name: Transit Commute Time Searching – Desktop UI (#691)

Description:

As a user of BreazeHome and the Desktop UI, I would like to use the commute time search in conjunction with transit based travel modes to find properties from an address I specify at a time of the day during a day of the week so that I can narrow down my candidate property choices to those that will not encumber me with excessive commute times using travel modes that are mass transit based.

Acceptance Criteria:

1. The current commute time search functionality must continue to work as is.
2. A new option for selecting modes of travel must be added. The new modes of travel must be as follows:
    a. Transit

3. New options for selecting modes of transit must be added. The new modes of transit must be as follows:
    a. Bus
    b. Rail
    c. Subway
    d. Train
    e. Tram
4. The search output of properties reachable in a given time/day in less than or equal to the commute time must be displayed as the results of the search.
5. The Mobile UI should display the exact same results the Desktop UI does.
6. The implementation must comply with BreazeHome security, performance, usability, and stylistic requirements as defined by the project.

PROJECT PLAN

This next section describes the Agile methodology and framework Team PoLR used to implement the feature set in the v1 release. A brief overview of the hardware and software stack that went into the PoLR implementation is also detailed.

Team PoLR adhered to a Scrum framework and methodology during the execution of the project. Team PoLR met virtually on a daily basis for a Scrum stand up. Intermeeting communication took place via WhatsApp to keep a reference of the historical conversation taking place between the team members when they were not all online and working in unison. As expected with a Scrum methodology, the team executed regularly cadenced sprints of two-week durations. Sprints were preceded by retrospective meetings for the previous sprint which then applied the lessons learned to the planning for the next sprint. At the conclusion of each sprint the team would present to the product owner in a sprint review meeting.

Early sprint execution was informal in the context of Agile and Scrum methodology. The team met on a bi-weekly basis after scheduled class times; however, formal recording of the planning minutes from those sessions was not aligned to Scrum methodology and emulated more a free form brainstorming and working session type meeting. Pictures, notes, and follow up email organizing the team effort were the output artifacts.

Hardware and Software Resources

The following is a list of all hardware and software resources that were used in this project:

Platform: BreazeHome
Development technology:
Frontend: JavaScript, AngularJS, NodeJS, SSAS/CSS, and HTML 5
Backend: Python 3.6, Django REST Framework
3rd Party APIs: Google Maps Geocode and Distance Matrix APIs
Hardware Environment: 4 Ubuntu 17.04 Linux Servers. 2 CPUs, 8 GB of RAM, each.
Database: FIU SCIS's PostgreSQL BreazeHome Data Back End

Sprint Plans

Sprint 4

Attendees: Jose, Juan, Robert, & Xu
Start time: 7:45pm
End time: 9:30pm

After discussion, the velocity of the team were estimated to be <Enter the estimated team velocity>.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.
1.Jose : 20 hours
2. Juan:20 hours
3. Robert: 20 hours
4. Xu: 20 hours

The team members indicated their willingness to work on the following user stories.
Jose - #672 - Search by commute time & price
Juan - #666-Generate a layer on the map based on commute search.
Robert - #682-Search properties by commute time desktop UI.
Xu - #678-Search properties using an exact address.

Sprint 5

Attendees: Jose, Juan Robert, & Xu
Start time: 6:45
End time: 9:45

After discussion, the velocity of the team were estimated to be <Enter the estimated team velocity>.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.
1.Jose : 20 hours
2. Juan:20 hours
3. Robert: 20 hours
4. Xu: 20 hours

The team members indicated their willingness to work on the following user stories.
Jose - #672-Search by commute time and price.
Juan - #685-Generate a multipoint layer on the map based on commute time search.
Robert - #682-Search properties by commute time desktop UI.
Xu - #684-Search properties using a valid address.

Sprint 6

Attendees: Jose, Juan, Robert, & Xu
Start time: 6:45pm
End time: 9:35pm

After discussion, the velocity of the team were estimated to be 80 story points.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.
1. Jose : 20 hours
2. Juan:20 hours
3. Robert: 20 hours
4. Xu: 20 hours

The team members indicated their willingness to work on the following user stories.
Jose - #690-Non Vehicular commute time searching - Desktop UI
Robert - #692-Search properties by commute time - centralized API
Xu - #667-Search properties by address
Juan - #712- Show the results of the commute time filter on the map
Juan - #718 - Manipulate commute time search filter properties on admin console

System Design

This section contains information on the design decisions that went into this project. The architecture patterns are outlined and explained. The entire system is shown in a package diagram and the subsystems are explained. Finally, the design patterns used in the project are discussed.

Architectural Patterns

PoLR Commute time feature is based on a MVC architecture managed by Django REST. From the WEB GUI Server, the required parameters (departure time, arrival time, destination and commute type) are introduced by the client and the results of the process are displayed in the web browser. This process is executed by the view-controller services. Commute-time filter calls Google Distance Matrix API to process the parameters obtained from the view-controller section and generates the isochrone with the points that are going to be used as a filter in the property model. This filter retrieves the desired properties from PostgreSQL database and sends this information back to the controller and the view as the final response to the client.

System and Subsystem Decomposition

PoLR commute time feature can be decomposed in its frontend and backend.

Frontend system is composed in services, templates and controllers.

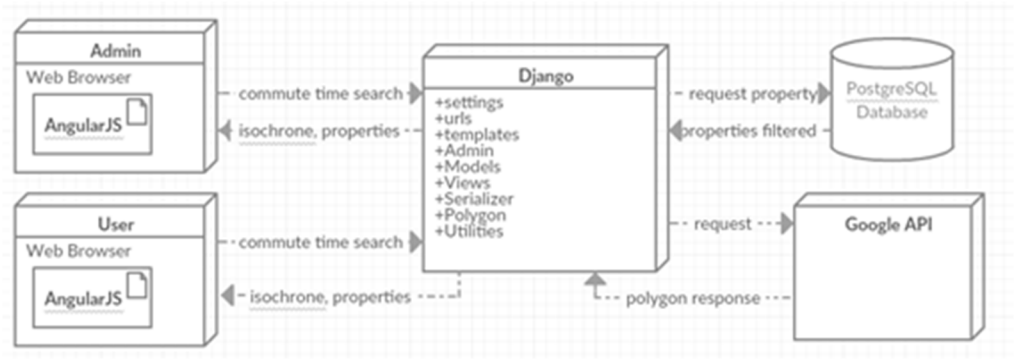
- Services: PoLR feature uses services to connection between the frontend and the backend, and also its bring support for the commute time search in order to calculate the selected day of the week and convert the coordinates of the isochrone in string values to be used in the results view
- Templates: commute time feature uses as base the same templates from Breazehome but it includes the creation of the commute time filter in the HTML Results
- Controller: this project uses BreazeHome controllers as base but the results are affected depending on the decision of the client to use this filter.

Backend system is composed in views, models and serializer. Also, two support components were created (polygons and utilities)

- Polygons and utilities allows to generate the isochrone using the Google API, creates the isochrone and apply a filter with its points
- Views: views are in charge of taking the requests and return the response. For the project, views returns the properties that were filtered.
- Model defines the classes which contains the rows and the columns that are linked with the database. PoLR manipulates the existing property model to retrieve the properties with the parameters desired.
- Serializer converts the data in python format and converts it in JSON. The projects uses propertySerializer in order to get a valid format of the property data, PolygonSerializer that allows to serialize locations as GeoJSON compatible data

Deployment Diagram

The deployment diagram shows the nodes and the artifacts that affects the process. The project uses two new services called polygons and utilities and they call the external system (Google API) in order to generate the polygon filter and send it back to the database artifact.

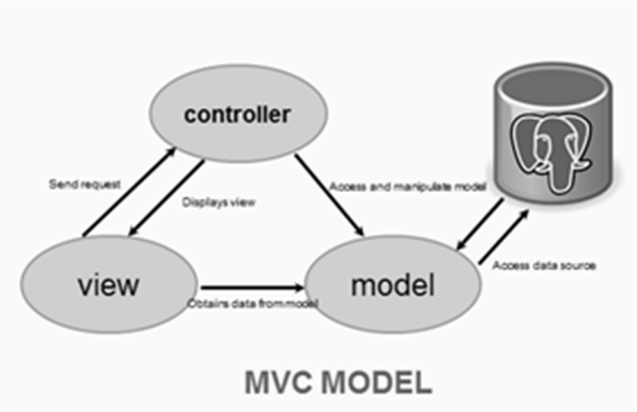


Deployment diagram

Design Patterns

Because the main project was built in AngularJS, PoLR also uses AngularJS implement a MVC design pattern. Its model is considered a singleton defined by service which maintains the information; it updates itself and responds from the view. View is the presentation of the data to the user. In this case, the project uses HTML and SCSS. Controllers in PoLR pull the views and

the model to show the final process to the user. Results controller is the main controller that integrates functions such as properties, map, lists, ModalServices, users. Other existing controllers that PoLR doesn't manipulate are landing, forgotpassword, among others. Backend implements an MVC design pattern, where views, models and controllers interact among them to retrieve the data and send it back to the frontend. Other services such as polygons and utilities bring support to the MVC model and given the desired result.



MVC model

SYSTEM VALIDATION

- **Test Case ID**: commute_type_0
- Description/Summary of Test: User does not select a commute type when invoking the commute time search
- Precondition: The user has invoked a commute time search and has selected a destination point, commute day, time of day, and commute time.
- Expected Results: The user should receive an alert indicating the missing requirement for a proper search. The search should not be invoked with missing parameters.
- Actual Results: The user received an alert indicating the missing requirement for a proper search. The search is not invoked with missing parameters.
- Status (Fail/Pass): Pass

- **Test Case ID**: commute_type_1

- Description/Summary of Test: User is on the results page and invokes a commute time search with driving as a selected commute type.
- Precondition: User has navigated to the results page and pulls down more filters. Commute time search is invoked with driving selected and all other required fields.
- Expected Results: User should be shown a map with an isochrone and filtered properties according to the specified commute type and all other required commute time fields
- Actual Results: User is shown a map with an isochrone and filtered properties according to the specified commute type and all other required commute time fields
- Status (Fail/Pass): Pass

- **Test Case ID**: commute_type_2
- Description/Summary of Test: User is on the results page and invokes a commute time search with cycling as a selected commute type.
- Precondition: User has navigated to the results page and pulls down more filters. Commute time search is invoked with cycling selected and all other required fields.
- Expected Results: User should be shown a map with an isochrone and filtered properties according to the specified commute type and all other required commute time fields
- Actual Results: User is shown a map with an isochrone and filtered properties according to the specified commute type and all other required commute time fields
- Status (Fail/Pass): Pass

- **Test Case ID**: commute_type_3
- Description/Summary of Test: User is on the results page and invokes a commute time search with walking as a selected commute type.
- Precondition: User has navigated to the results page and pulls down more filters. Commute time search is invoked with walking selected and all other required fields.
- Expected Results: User should be shown a map with an isochrone and filtered properties according to the specified commute type and all other required commute time fields
- Actual Results: User is shown a map with an isochrone and filtered properties according to the specified commute type and all other required commute time fields
- Status (Fail/Pass): Pass

- **Test Case ID:** map_destination_1
- Description/Summary of Test: Given the user is on the results page, when the user extends the filters, inserts the parameters of day, arrival time, departure time, commute type and clicks on apply filters, then the map should display a marker showing the destination point.

- Pre-condition: The user is on the map page, he has already inserted the parameters and click on apply filters.
- Expected Results: Marker showing destination point should be displayed on the map.
- Actual Result: the marker of destination point is shown on the map
- Status (Fail/Pass): Pass.

- **Test Case ID:** map_layer_1
- Description/Summary of Test: Given the user is on the results page, when the user extends the filters, inserts the parameters of day, arrival time, departure time, commute type and clicks on apply filters, then the map should display a blue layer showing the reachable distance from the given point.
- Pre-condition: The user is on the results page, he has already inserted the parameters and click on apply filters.
- Expected Results: the distance layer should be displayed on the map
- Actual Result: the distance layer is displayed on the map
- Status (Fail/Pass): Pass.

- **Test Case ID:** map_marker_property_1
- Description/Summary of Test: Given the user is on the results page, when the user extends the filters, inserts the parameters of day, arrival time, departure time, commute type and clicks on apply filters, then the map should display the markers pertaining to the properties
- Pre-condition: The user is on the results page, he has already inserted the parameters and click on apply filters.
- Expected Results: Marker showing properties should be displayed on the map.
- Actual Result: property markers are shown on the map
- Status (Fail/Pass): Pass.

- **Test Case ID:** map_cluster_1
- Description/Summary of Test: Given the user is on the results page, when the user extends the filters, inserts the parameters of day, arrival time, departure time, commute type and clicks on apply filters, then the map should display clusters of properties based on how near the properties are.
- Pre-condition: The user is on the results page, he has already inserted the parameters and click on apply filters.
- Expected Results: map should display clusters of properties based on its proximity
- Actual Result: clusters are made based on the proximity among properties
- Status (Fail/Pass): Pass.

- **Test Case ID:** map_marker_onclick_1
- Description/Summary of Test: Given the user is on the results page, when the user extends the filters, inserts the parameters of day, arrival time, departure time, commute type, clicks on apply filters and retrieved the properties pertaining the commute time filter, then the property should pop-up a windows with its detailed information when it's clicked.
- Pre-condition: The user is on the results page and he has already made a commute time search.
- Expected Results: a new window should open with the detailed information of the property when its marker is clicked.
- Actual Result: a new window opens with the detailed information of the property when its marker is clicked
- Status (Fail/Pass): Pass.

- **Test Case ID:** map_commute_type_1
- Description/Summary of Test: Given the user is on the results page, when the user extends the filters, inserts the parameters of day, arrival time, departure time, selects "driving" as commute type and clicks on apply filters, then the map should display the distance layer and the properties of the "driving" commute type.
- Pre-condition: The user is on the results page, he has already inserted the parameters and click on apply filters.
- Expected Results: distance layer and properties inside the layer should appear in relation with the commute type selected.
- Actual Result: driving commute-time filter is show on the map and the properties are inside the layer
- Status (Fail/Pass): Pass.

- **Test Case ID:** map_commute_type_2
- Description/Summary of Test: Given the user is on the results page, when the user extends the filters, inserts the parameters of day, arrival time, departure time, selects "bicycling" as commute type and clicks on apply filters, then the map should display the distance layer and the properties of the "bicycling" commute type.
- Pre-condition: The user is on the results page, he has already inserted the parameters and click on apply filters.
- Expected Results: distance layer and properties inside the layer should appear in relation with the commute type selected.

- Actual Result: bicycling commute-time filter is show on the map and the properties are inside the layer
- Status (Fail/Pass): Pass.

- **Test Case ID:** map_commute_type_3
- Description/Summary of Test: Given the user is on the results page, when the user extends the filters, inserts the parameters of day, arrival time, departure time, selects “walking” as commute type and clicks on apply filters, then the map should display the distance layer and the properties of the “walking” commute type.
- Pre-condition: The user is on the results page, he has already inserted the parameters and click on apply filters.
- Expected Results: distance layer and properties inside the layer should appear in relation with the commute type selected.
- Actual Result: walking commute-time filter is show on the map and the properties are inside the layer
- Status (Fail/Pass): Pass.

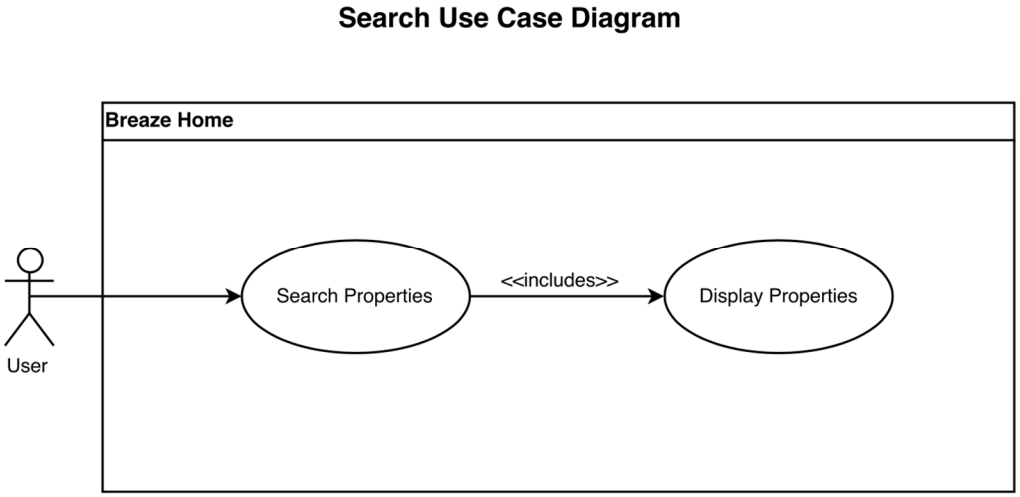
- **Test Case ID:** map_redo_polr_1
- Description/Summary of Test: Given the user is on the results page, when the user has already made a commute time search and wants to make another commute time search, then the map display the results of the commute time search.
- Pre-condition: The user is on the results page, he has already made a commute time search.
- Expected Results: a new layer and its respective property markers should appear on the map, the previous layer should disappear.
- Actual Result: the new layer and the markers of the properties pertaining to the current commute time search appear on the map and the previous search disappears.
- Status (Fail/Pass): Pass.

GLOSSARY

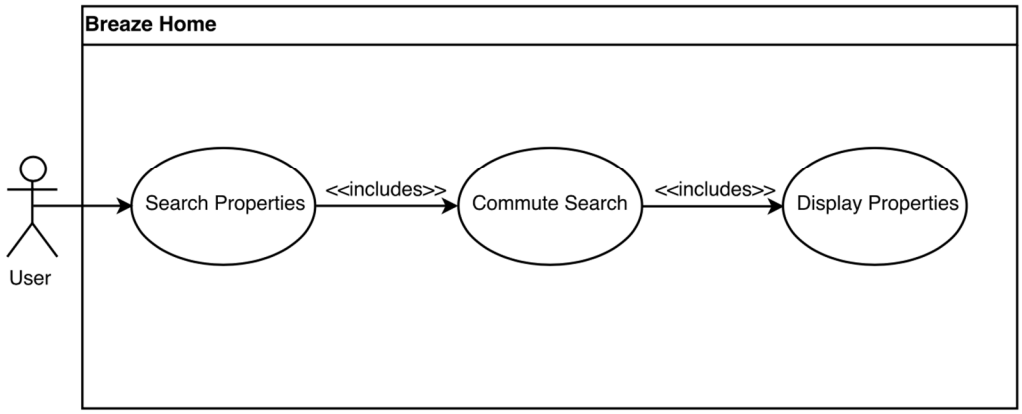
PoLR – Path of Least Resistance, pronounced Polar or PōLR.
Isoline – A line segment from an origin point to a distal radial point the represents the distance traversable from the distal point to the origin in a give amount of time.
Isochrone – A polygon consisting of all of the distal radial points in each radial isoline for a common origin. The area contained within the polygon represents the set of points from which the origin is reachable in less than or equal to the time for each distal radial point.

APPENDIX

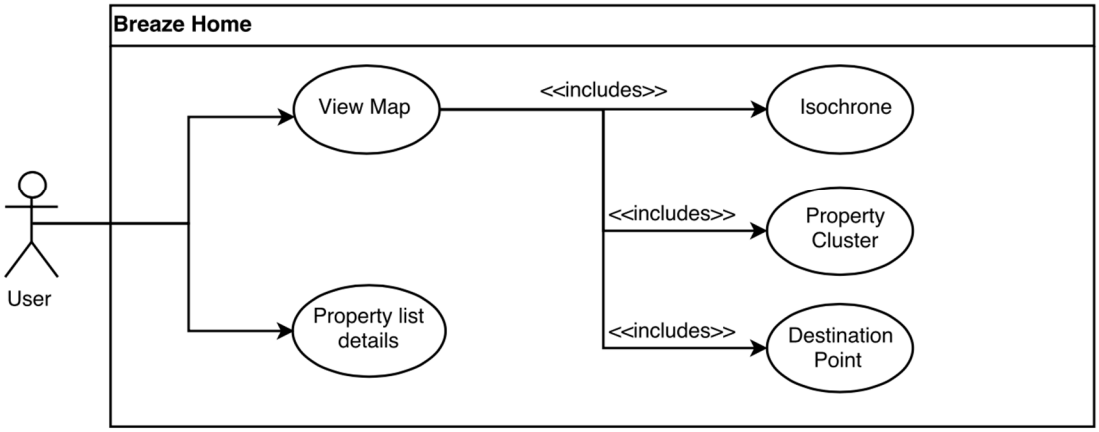
Appendix A - UML Diagrams

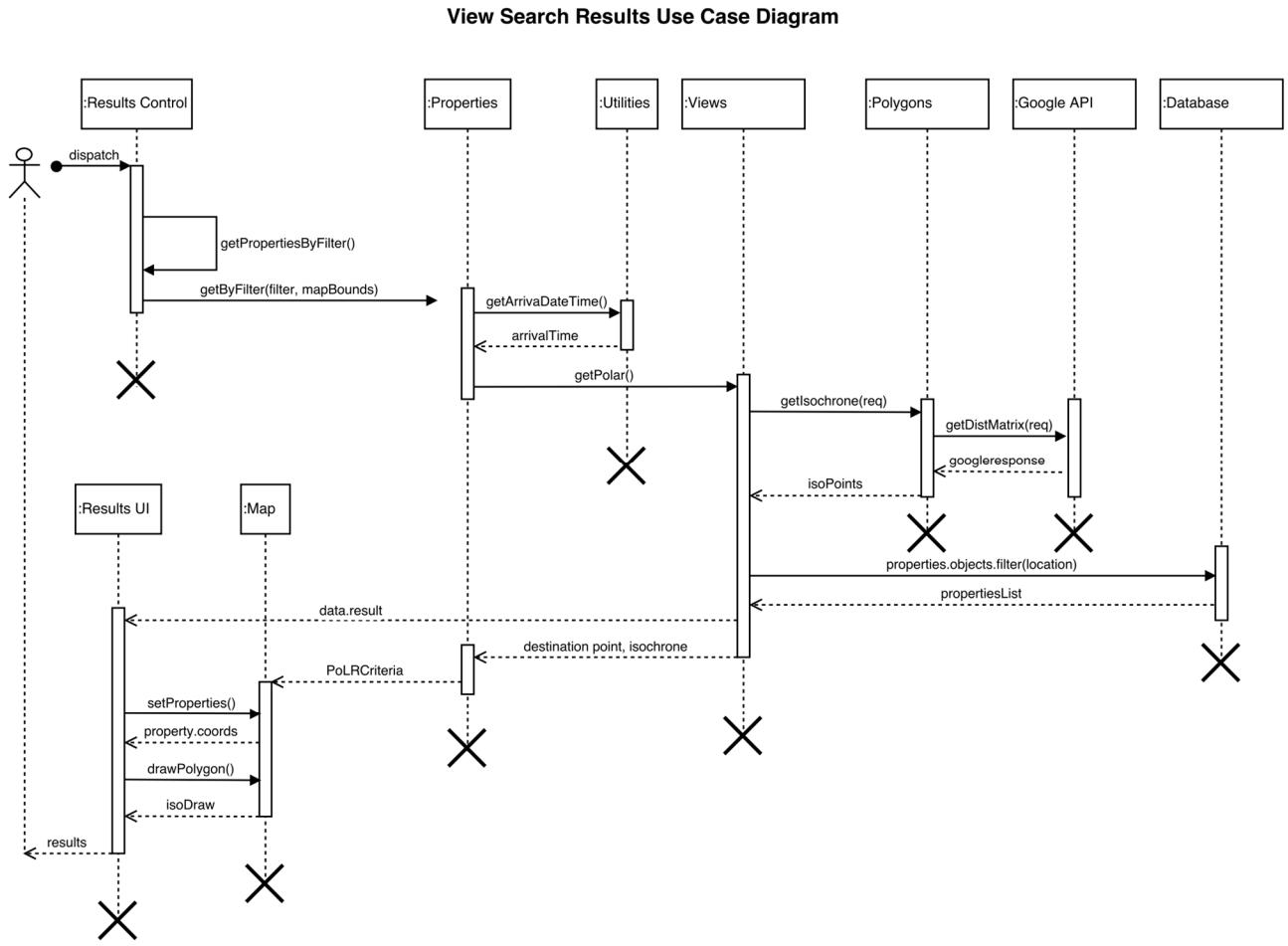


Commute Search Use Case Diagram



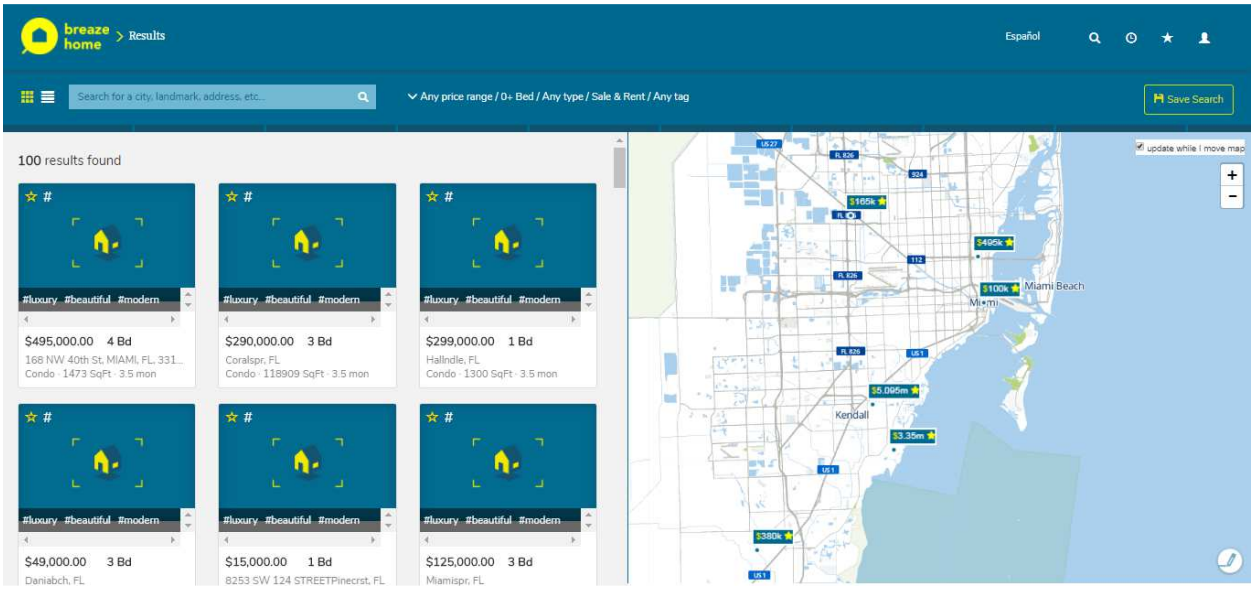
View Search Results Use Case Diagram





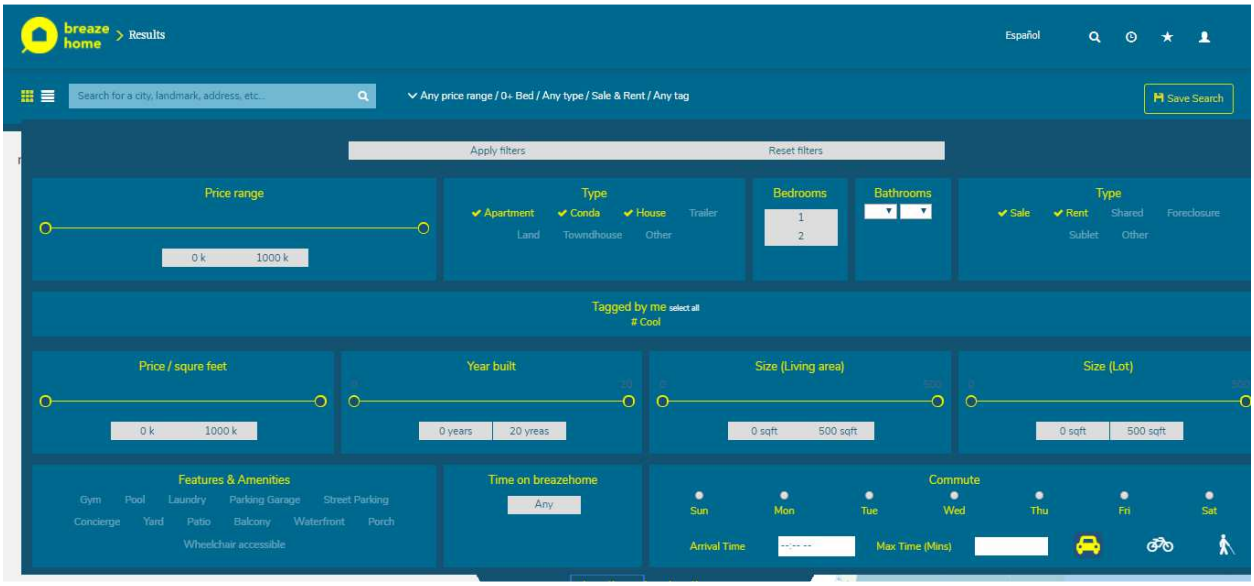
Appendix B - User Interface Design

BreazeHome results page

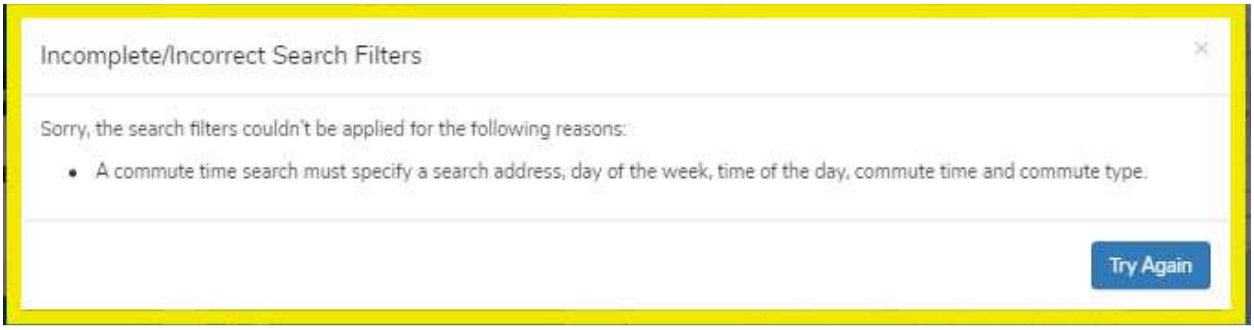


Commute time filter

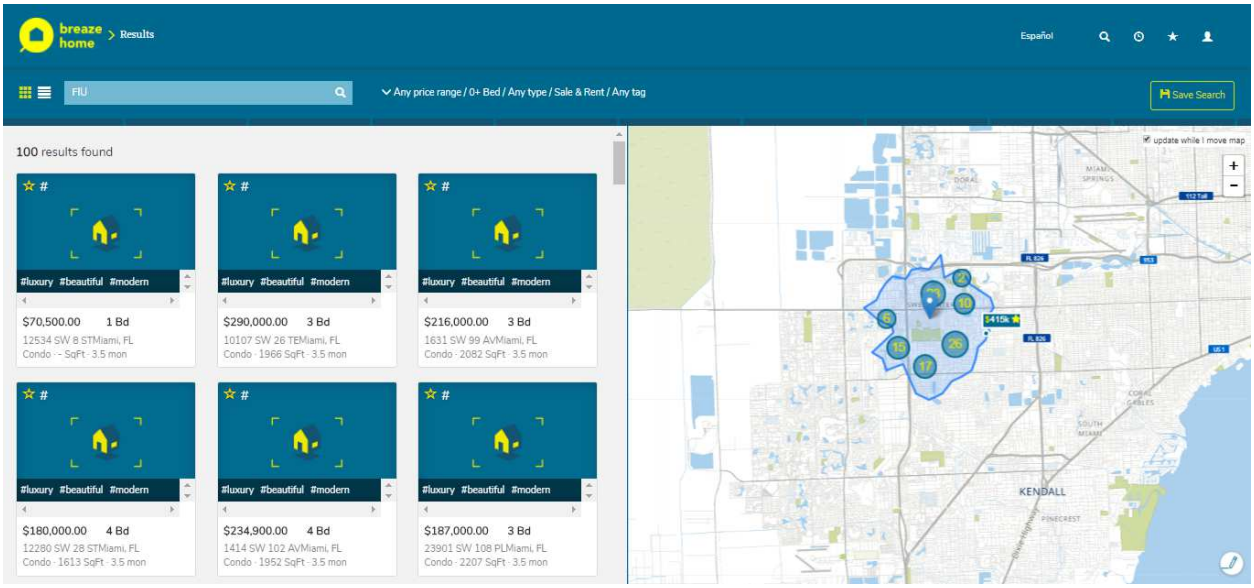
Location of the commute time filter in BreazeHome



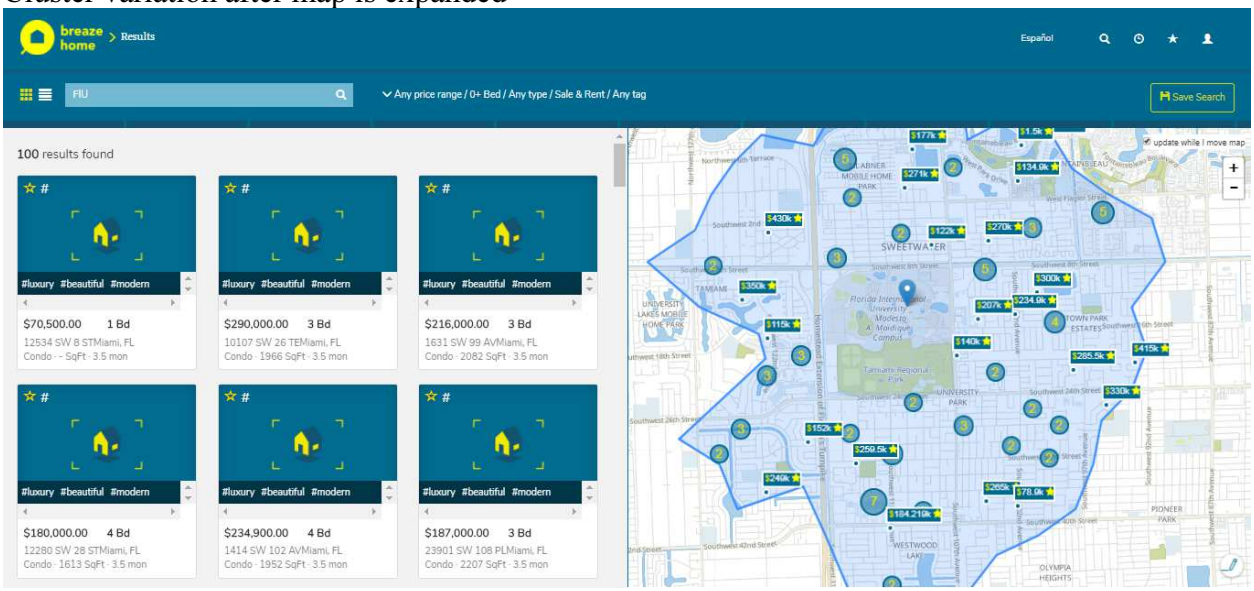
Missing parameters notification message for client



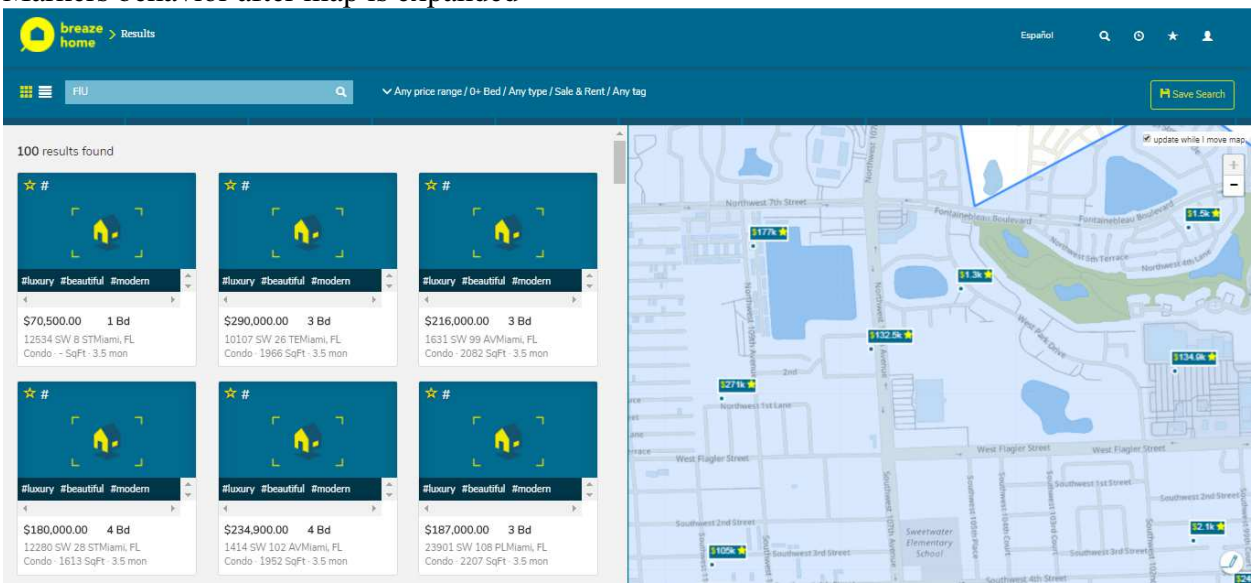
Results page after processing commute time filter



Cluster variation after map is expanded



Markers behavior after map is expanded



Appendix C - Sprint Review Reports

Sprint 3

Release #: 1
Sprint #: 3
Date:10/11/2017
Attendees: Xu, Robert, Jose, Juan
Start time: 5:30pm
End time:

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.
Jose
672 Search by commute time and price
Juan
#666 Generate a layer on the map based on the commute time search
Robert
#682-Search properties by commute time desktop UI.
Xu
#677 feasibility study about the address search

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Spring Planning meeting.
Jose
none
Juan
(Corrected user story #666)
Robert
Xu
None

Sprint 4

Release #: 1

Sprint #: 4
Date:11/08/2017
Attendees: Xu, Robert, Jose, Juan
Start time: 5:30pm
End time:

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.
Jose
672-Search by commute time and price
Juan
#685-Generate a multipoint layer on the map based on commute time search
Robert
#682-Search properties by commute time desktop UI.
Xu
#684-Search properties using a valid address

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Spring Planning meeting.
Jose
none
Juan
none
Robert
none
Xu
none

Sprint 5

Attendees: Jose, Juan, Robert, Xu, Dr. Sadjadi
Start time: 5:30pm
End time: 6:00pm

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.
#682-Search Properties By Commute Time – Desktop UI

UI Feedback was received. The UI should be pushed down into the “More Filters” section
#684-Search properties using a valid address
#672-Search by commute and price
Implementation feedback was received, the story should encompass the implementation for searching by commute and all other filters.
#685-Generate a multi-point layer on the map based on the commute time search

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Spring Planning meeting.
Jose
none
Juan
none
Robert
none
Xu
none

Sprint 6

Attendees: Xu, Robert, Jose, Juan

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.
#690-Non Vehicular commute time searching - Desktop UI
#692-Search properties by commute time - centralized API
#667-Search properties by address
#712- Show the results of the commute time filter on the map
#718- Manipulate commute time search filter settings on admin console

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Spring Planning meeting.
The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Spring Planning meeting.
Jose
none
Juan

none
Robert
none
Xu
none

Appendix D - User Manuals, Installation/Maintenance Document, Shortcomings/Wishlist Document and other documents

Feature Introduction and User Manual Video

https://youtu.be/_gE5F_miKJ4
https://youtu.be/8f-sMPp_ipM

Installation and Maintenance Video

https://youtu.be/yLhhSO1UaCQ

Shortcomings and Wishlist Video

https://youtu.be/LgLnHRXw6H0

Release Notes for Integration into Production

Modifications of existing code

breaze-desktop

All substantive changes to existing BreazeHome code are wrapped with “BEGIN PoLR” and “END PoLR” comments. The easiest way to find any changes is to search for the “BEGIN PoLR” string.
All console logging for debugging is enabled/disabled via the 'VERBOSE_CONSOLE_DEBUG': true setting settings.js. It is disabled by default in our checked in release (set to false).

breaze-api-django

All substantive changes to existing BreazeHome code are wrapped with “BEGIN PoLR” and “END PoLR” comments. The easiest way to find any changes is to search for the “BEGIN PoLR” string.
All console logging for debugging is enabled/disabled via the POLR_DEBUG_LEVEL setting settings.py. It is disabled by default in our checked in release (set to 0).

Additions of new code files

breaze-desktop
app/views/modals/search-incorrect.html

breaze-api-django
real_estate/polygons.py
real_estate/utilities.py

Commenting of code

All code is intuitively written, clearly inline commented for logic and intent purposes, and documented appropriately with code documentation headers for publicly facing code.

References

[1] U.S. Census Bureau, "A Look at Commuting Patterns in the United States from the American Community Survey," 8 March 2013. [Online]. Available: https://www.census.gov/content/dam/Census/newsroom/c-span/2013/20130308_cspan_commuting.pdf. [Accessed 31 August 2017].

[2] M. Hilbrecht, B. Smale and S. E. Mock, "Highway to health? Commute time and well-being among Canadian adults," *World Leisure Journal,* vol. 56, no. 2, pp. 151-163, 08 Apr 2014.