Final Report

for

BTown Properties

Prepared by Group 13

Gaurav Ranade

Gouri Netravali

Jay Mercer

Ritesh Kavungal

Suresh Kanagala

Enrique Areyan

8 December 2011

Table of Contents

1. Introduction

1.1 Project Scope

BTown Properties is a web-based system where members and visitors can search the system's database for other members' properties. Only a member is allowed to contact another member to set meeting times to see a property. The system will allow members to upload information regarding their properties, maintain watch lists of properties and property features, and write reviews about other members' properties.

1.2 Purpose

In this document, we describe the conceptual features of *BTown Properties*, as well as its main design components and implementation methodology. In particular, the document contains important definitions used throughout the project, the systems' functional requirements, data modeling, software architecture, optimization on the database and the testing methodology employed to ensure the quality of the application.

The data modeling design contains both the ER modeling design and the design of the relational schema. Also, *BTown Properties* business logic will illustrate the flow of data/control between the front-end and back-end, and the transition among web pages. Moreover, in the section on implementation details, we point out how the Zend Framework was leveraged as an architectural template and library repository to enhance the application's development. Finally, we describe important optimization performed on the database, making emphasis on the creation of relevant indexes.

Note: we will use the male pronoun to refer to both male and female users.

1.3 Definitions

The following terms will be used throughout the document.

Property: physical unit with a unique address in the real world and whose owner has voluntarily agreed to upload its information into the system.

Users' accounts: there are two types of *users' accounts*:

1. Administrative account: a user with this type of account will have permission to access and perform operations on all the data in the system's database. This user can enable and disable other users' accounts. This user is simply called an *administrator*.
2. Member account: a user with this type of account can enjoy all the benefits of the system. A user with a member account is simply called a *member*. A member cannot modify or erase data of another member.

Visitor: person that visits *BTown Properties* but does not have any type of user account. A visitor can only search the property's database but will need to sign up for a member account to enjoy the system's other benefits.

Watch list: collection of properties defined by a member according to his preferences. There are two types of watch lists:

1. Properties watch list: a member can add a specific property to this list.
2. Features watch list: a member can add specific features to be watched into this list, e.g. watch all properties with two bedrooms and 1.5 bathrooms; the watch list will contain all properties that match such features.

2. System Features (SF)

In this section, we present the functional requirements or major services provided by the system. The section is organized by detailing each major feature along with its priority. A priority is a natural number from 1 to 3, with 1 being a high priority feature (core functionality), 2 a medium priority, and 3 a low priority (nice-to-have feature).

2.1 Visitor Sign-Up. Priority 1.

2.1.1 Description: a visitor should be able to sign-up and become a member of *BTown Properties*. A member is uniquely identified by his primary email address, which is the main point of contact between the member and the system.

2.1.2 Functional Requirements: the user must input the following data. (*) means the data is obligatory (this same notation is used throughout the system features):

- First and Last Name (*)
- Primary Email Address (*)
- Date of Birth
- Nickname
- Password (*)
- Phone Number

Once the data is validated, the account is activated and he can login into the system (SF 2.2).

2.2 Member Login/Logout. Priority 1.

2.2.1 Description: members can login into the system where they can access their data (SF 2.3), change their personal information, review their watch lists (SF 2.6), and review any contacts that might have been received or sent (SF 2.5). Also, members can logout of the system.

2.2.2 Functional Requirements: the member inputs his primary email address (SF 2.1) and password to login in the system. If the credentials are valid, a session will be created. If not, an error message will be displayed. Finally, a link will be provided for the member to logout. The logout function will destroy the session previously created.

2.3 Add, modify and delete properties. Priority 1.

2.3.1 Description: a member can add, modify and delete only his own properties. Once a property is in the database, it will be accessible to both users and visitors.

2.3.2 Functional Requirements: the member inputs the data of a property through a form with the following fields:

- Address (*), e.g., 980 W. Basswood St. Apt. C, Bloomington, IN, 47403.
- Description.
- Photo.
- Features, e.g., number of bedrooms, bathrooms; utilities are included or not, etc.
- Rental/Sale price

The member may, at any time, modify or delete this information.

2.4 Property view. Priority 1.

2.4.1 Description: members and visitors can view the details of any single property.

2.4.2 Functional Requirements: an interface will show all the data of a single property in a centralized manner. The property view will display the property's information and any review that it may have. A user can access this view directly if they know the property's id in the system or they may reach it via the search function (SF 2.5). A Google Map will be shown pointing at the property's location.

2.5 Search and list properties. Priority 1.

2.5.1 Description: visitors and members can search the properties database to display a list of properties that match criteria selected from a list of property attributes. Example search attributes would include as min/max rent, number of bedrooms/bathrooms, pool, etc. A basic, "quick," search would result in a list of properties that match the given criteria. Due to time constraints, a more advanced "skyline query" search feature was not implemented to filter and rank the results so that the user can select the most promising ones.

2.5.2 Functional Requirements: the basic search would consist of a simple SQL SELECT query. After inputting the desired search criteria and submitting the form, the user would be presented with a list of properties matching selected criteria. Clicking on a search result would allow them to view the single property. (SF 2.4)

2.6 Contact a member. Priority 1.

2.6.1 Description: a contact function will stand for a "rent" function given that before the actual physical rent of a property takes place, the parties involved must undergo other processes in the real world and outside the scope of the system. Instead, a member can contact another member and set a meeting to see a property**.**

2.6.2 Functional Requirements: a member viewing the details of a single property (SF 2.4) may click a *Contact* link, which will display a form with fields:

- Property (fixed with the current property)
- Owner Member Name (fixed with the current owner)
- Contacting Member Name (fixed with the current member)
- Commentary
- Proposed Date of Visit

Once the member submits the form, a copy will be sent to the owner member via email.

2.7 Watch Lists. Priority 2.

2.7.1 Description: members may maintain two types of watch lists: properties watch list and features watch list.

2.7.2 Functional Requirements: the system will indicate to a member who is viewing (SF 2.4) a single property that is in either his properties or features watch list, that he is "watching" this property. Otherwise it will allow him to add the property only to the properties' watch list.

The features watch list depends on specific features of a property and thus may not be added one at the time. For instance, a member may define a features watch list to watch all properties with 2 bedrooms and a pool. In the future, if a property with such characteristics is uploaded into the system, it will be displayed on his property watch list view.

2.8 Review properties. Priority 2.

2.8.1 Description: a review is a commentary made by one member about a property. A member may review another member's properties but he may not review his own properties. A message will be display to members about *BTown Properties'* non-offensive language policy. Moreover, the system will check and disable any potential offensive language in a review. The reviews will be visible to both members and visitors in the view of a property (SF 2.4).

2.8.2 Functional Requirements: a member may click on a *Review* link inside the view of a single property (SF 2.4), and fill out a review form with the following fields:

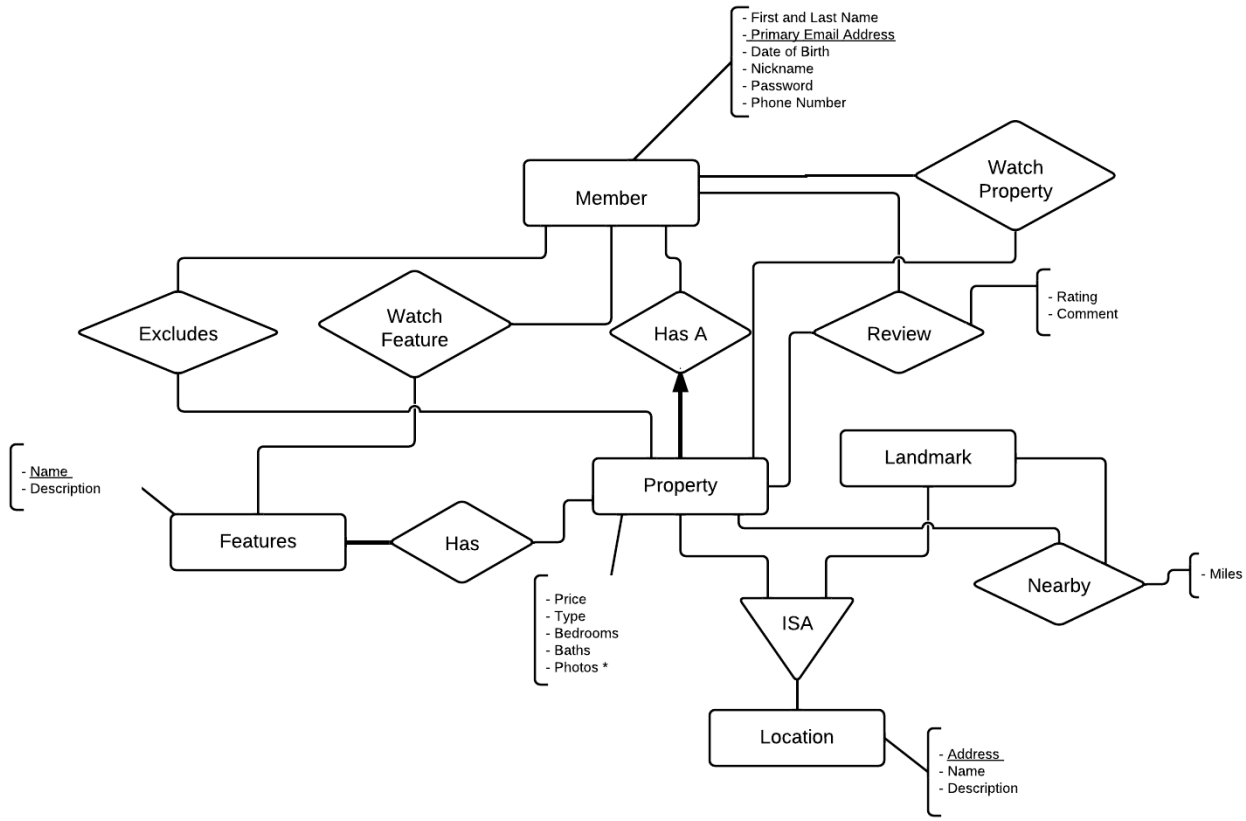
- Commentary (*)
- Ranking (1-10) (*)

Once a commentary is submitted, it will go through a "bad word filter" that will check each word of the message and delete any potential bad word that are matched against a "bad words" repository. If a message surpasses a certain number of "bad" words, it will be placed under quarantine until an administrator reviews it.

Appendix A shows a schematic of the system's features.

3. Data Modeling Design

3.1 ER Modeling Design



Notes on ER Diagram: Only one member owns a property. A member may or may not have one or many properties. A member may or may not review any number of properties. Likewise, a property may or may not have one or more reviews. A property and a landmark are kinds of

locations. A Property may be near a Landmark. A member may watch one or many properties or features. A member may like to exclude properties so that they do not show on search results.

3.2 Relational Schema

Members (**Member ID**, First Name, Last Name, Email Address, Date of Birth, Nickname, Password, Phone Number)

Artificial key Member ID is used for implementation convenience.

Features (**Feature ID**, Name, Description)

Artificial key Feature ID is used for implementation convenience.

The Feature ID represents amenities and is the primary key. For example, a Feature ID with a value of 1 can represent an amenity such as swimming pool, a Feature ID with a value of 2 can represent parking facility etc. The description attribute gives more information about the features.

Has_Feature(**Property ID, Feature ID**)

Property ID is a foreign key to Property ID in Properties entity. Feature ID is a foreign key to Feature ID in Properties entity.

Properties (**Property ID**, Name, Description, Address, Price, Type, Bedrooms, Bathrooms, Member_ID, photo)

Artificial key Property ID is used for implementation convenience. The 'type' attribute indicates whether this property is a rental or sublet. Bedrooms and bathrooms are natural numbers indicating the property's number of bedrooms and bathrooms respectively. Member_ID is a foreign key to Member_ID in Members entity.

Artificial key Photo ID is used for implementation convenience. Property ID is a foreign key to Property ID in Properties entity.

Landmark (**Landmark ID**, Name, Description, Address)

Artificial key Landmark ID is used for implementation convenience.

Nearby (**Property ID, Landmark ID**, Miles)

The 'Nearby' relation would give us the distance between a property and a nearby landmark. Property ID is a foreign key to Property ID in Property entity. Landmark ID is a foreign key to Landmark ID in Landmark entity.

Review (**Member ID, Property ID**, Rating, Comment)

Member ID is a foreign key to Member ID in Members entity. Property ID is a foreign key to Property ID in Properties entity. The rating would be scaled from 1 to 5.

WatchListFeatures (**Member ID, Feature ID**)

Member ID is a foreign key to Member ID in Members entity. Feature ID is a foreign key to Feature ID in Features entity.

WatchListProperties (**Member ID, Property ID**)

Member ID is a foreign key to Member ID in Members entity. Property ID is a foreign key to Property ID in Properties entity.

ExcludeProperties (**Member ID, Property ID**)

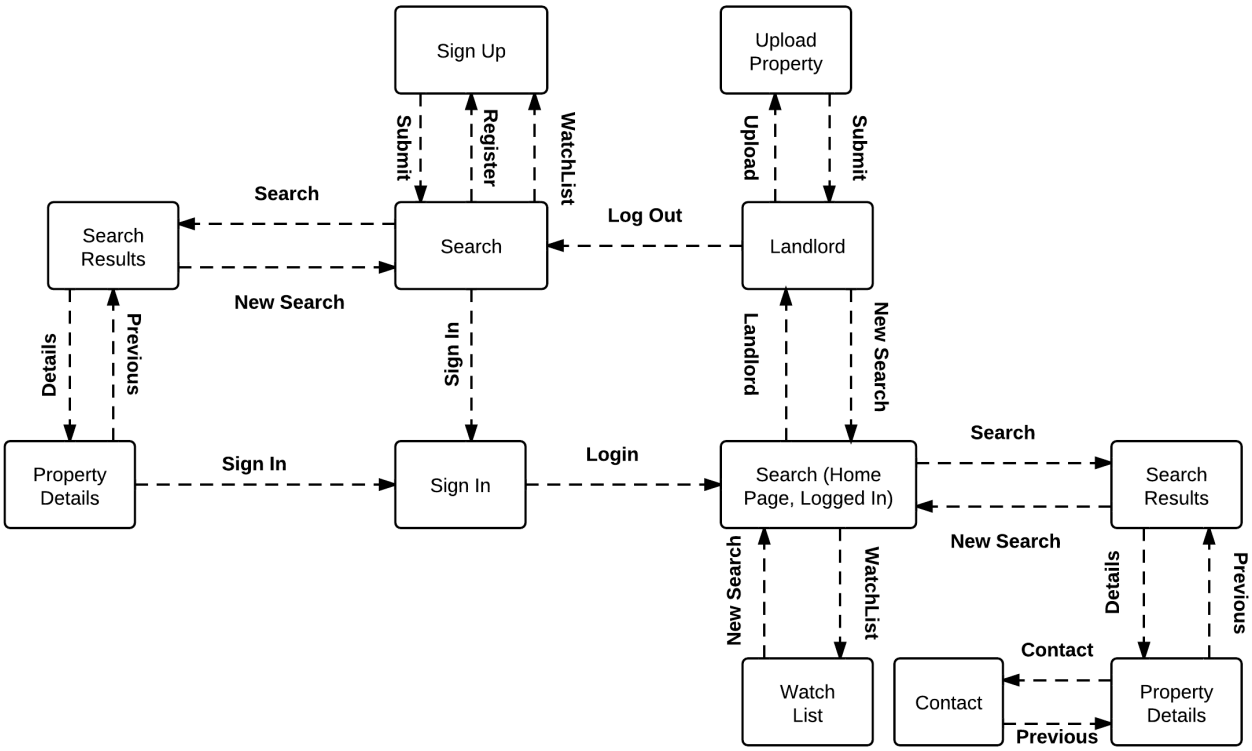
Member ID is a foreign key to Member ID in Members entity. Property ID is a foreign key to Property ID in Properties entity. Web Interface Design (WD)

3.3 Transition among the Web Pages

The navigation among the web pages is indicated by the schematic below. Each box represent a Web Page, and each is named for convenience of the reader. A dashed arrow leaving a box represents a click made by a visitor or a member in a page. The arrow points towards the destination reached by a member or visitor.

Note that whenever a visitor clicks on any of the system's restricted features, he is redirected to the sign up page since those features are only accessible to members.

Like discussed in the ER Diagram, visitors can still access all the search facilities, hence a second home page has been created (for visual purposes only) to indicate what a visitor can or cannot do.



3.4 Business Logic

3.4.1 General Rules

A visitor cannot contact a member; only members can interact. When a visitor tries to access restricted resources, the system will redirect him to a sign-up page. The system's restricted resources are: properties upload, contact a member and maintain watch lists.

For privacy reasons, a landlord will not be able to see the members who added his property into their watch lists but will see the count, i.e. the number of users, who added his property into their watch lists.

3.4.2 Specific Rules

The following rules are specific to a web page. We will denote the webpage to which we are referring by its code number; e.g., (WD 4.1.2) refers to sign-in on page 7.

When a user enters a location name (WD 4.1.1), he will be able to see properties listed in that location only. Properties close to the location will not be shown. 'Rent Min' and 'Rent Max' are not mandatory fields, which imply that a user can enter only a 'Rent Min' value or only a 'Rent Max' value to attain preferred results. If visitor or member enters only half of the property name, for example if he enters 'park', the system should be able to display all properties having 'park' in their names such as Park Doral, Fountain Park, etc. The same functionality would apply for any such related searches.

In (WD 4.1.2), if a visitor leaves a field blank or enters an invalid email address, the system has to direct him to correct the information. If the credentials do not match in the system, the visitor will be redirected to (WD 4.1.2), and two messages will be shown: 1) "forgot password?" which will provide functionality to recover a password and 2) "become a member" which will redirect to (WD 4.1.2) Sign Up page. As soon as the visitor registers and becomes a member, a registration email notification would be sent to him to confirm his credentials.

In (WD 4.1.5), in the property description, a "Contact" button will only be shown to members. In the place of the "Contact" button, a visitor will see the message: "Do you want to contact the owner of this property? Click here and become a member to be able to do it"

In (WD 4.1.6) when a member tries to add the same property that he had already added into his watch list, the system will notify him. There can be overlapping between the features and properties watch list.

4. Implementation

In this section, we present the system's most important implementation details.

4.1 Zend Framework and MVC Design Pattern

BTown Properties was built on top of **Zend Framework** (http://framework.zend.com/) for PHP. The Framework offers an implementation of the Model-View-Controller (MVC) object-oriented design pattern, which provides an adequate structure for the application from a software architectural point of view. In a nutshell, an MVC application is divided into Models (objects to access and operate on the data layer) View (objects to work with the presentation layer) and Controllers (objects in which to implement the business logic). The MVC architecture fits nicely into the three-tier application development paradigm discussed in class.

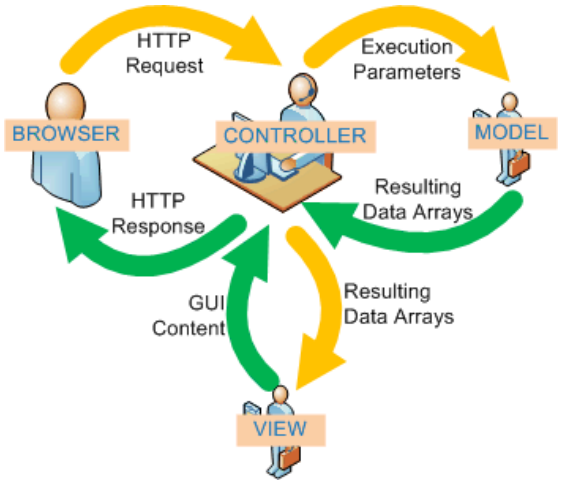


Figure source: http://www.cyblance.com/development/mvc-development

Apart from using the framework as an architectural template, we adapted some of the pre-packed objects to fit the particular needs of *BTown properties*. In concrete, we used objects to handle login and session (Zend_Session http://framework.zend.com/manual/en/zend.session.html) and HTML Form rendering and validating (Zend_Form http://framework.zend.com/manual/en/zend.form.html). Other objects used were: Zend_Registry to keep a singleton access to important data, such as user session; and Zend_Paginator to easily handle pagination of large amounts of data, such as the list of properties in the database.

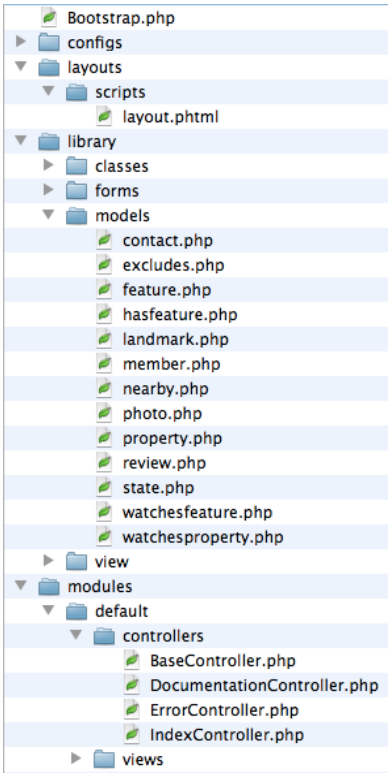
Lastly, custom objects were coded to handle *BTown Properties* specific-business logic. All of these objects are grouped in the library/ folder of the application's files. The most important objects are: LatLng, which encapsulates all the code needed to communicate with Google's API and display a Google map positioned at an arbitrary location. SearchProperties, which encapsulates all the logic related with the search of a property in the database.

One of the main advantages of the MVC architecture is the ability to easily divide responsibilities among developers in an efficient manner. This follows from the fact that each layer of the MVC model communicates with others in a well defined API and thus, one developer may be working on one layer without affecting other portions of the system. The keywords are encapsulation and abstraction.

4.2 BTown MVC Arquitecture

The following figure contains *BTown Properties'* MVC files and folders structure.

Application Structure



We divided *BTown Properties* into modules in order to make the application's development, test and maintenance easier.

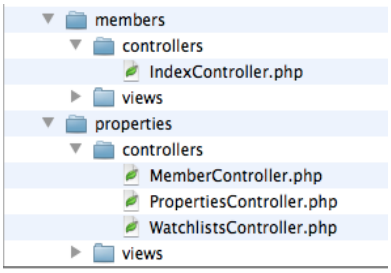
There are three modules: default, members and properties. As one can see in the figure on the left, there is a folder for each module in the modules/ folder. Each module has two folders, i.e., controllers/ and views/. As can be inferred from their names, the controller folder contains all the controllers for a given module as so the view folder. A module can have many controllers, which are PHP classes that individually implement part of the application's business logic and that combined constitute all of the application's business logic.

The default module contain the application Index, Error handling and Base Controller, all of which are special purpose controllers. The application index is the main page of the project. The error controller gets invoked each time an error occurs in the application that cannot be fix within a given controller. The base controller is extended by every controller in the system and thus constitute a centralized code in which common operations can be implemented just once, e.g., check whether a user is logged in or not.

The members module contains a single controller in which all the logic related to a member is implemented, e.g., sign up.

The properties module groups the logic from the point of view of a property. This module contains three controllers: member, properties and watch lists.

The library/ folder contains folders named: classes, forms, models and view. The classes folder contains custom objects such as LatLng and SearchProperties mentioned before. The forms folder contains objects for the HTML forms used to interface with the user. The models folder contains all the models used to interface with the data



layer, e.g, properties, member, etc.

The layout/ folder contains a sub-folder scripts/ with a single file, i.e., layout.phtml. This single file contains the base html markup used throughout the application and thus, serves as a centralized point in which to quickly add/remove new presentation details without having to modify many files.

Specific views for each action inside a controller are stored in the views/ folder of each module. These views are referred to as 'partial views', because these implement a portion of the presentation layer corresponding to specific actions. Partial views are rendered together with the layout to produce final results for the end-user.

4.3 Search functionality

The search functionality is encapsulated in two files: library/classes/SearchProperties.php and library/view/helper/DisplayProperties.php. The first file is an object we coded to perform the search query in http://www.cs.indiana.edu/cgi-pub/mercerjd/properties/properties/search. The second file implements an object that handles the presentation logic of a list of properties, i.e., it displays an HTML table with the name, description, address, price, type, bedrooms, bathrooms and arrows to sort results by all of these fields. The output obtained by this object can be parameterized to handle different cases, e.g., if a member is logged in, it can show an option to add a property to the watch list. Because this file is properly encapsulated and abstracted from the data and business logic layers, it is used to display list of properties throughout the site. In concrete, it is used in the home, search, list my properties, property watch list and features watch list pages.

4.4 Features Watch list

In the features watch list link, a member is provided with an HTML form containing solely a checkbox per feature stored in the database. The user can check any combination of features, meaning that he wants to search those properties containing any of such features. The system automatically saves this configuration of features in the database. If the users navigate away from the feature watch list or logouts into the system and returns at a later point in time, the previously saved features will be automatically shown in the feature's form and the corresponding properties will be displayed below.

4.5 Admin module

If a member has administrative permissions, then he will see a manage option in addition to all other options provided to members on the head menu. In the manage option an admin user can manage features, landmarks and members. In the manage features option, an admin can add or delete features. If a feature is added, it will be automatically added in the search and upload properties form and thus, accessible to all visitors and members. In the same manner, a deleted option will be removed from these forms. A similar functionality is provided for both landmarks and users but with appropriate changes reflected throughout the system. For instance, if a landmark is removed, it will no longer be displayed in the search form and on any of the system's Google map.

4.6 URL

The application can be accessed and tested at **http://www.cs.indiana.edu/cgi-pub/mercerjd/**

For admin capabilities, please login as user: **mercerjd**, password: **b56113**

5. Database Optimization

In most data-intensive web applications, one of the first performance bottlenecks usually faced by developers involves the access to the data layer. This often means that the data stored in the database has increased up to the point where some operations, or bottleneck points, are taking longer than usual. Many of these bottlenecks points occur when searching for particular data in large databases. In order to avoid such issues and optimize the database performance, one may want to create indexes to aid in searching for data, especially in complicated queries.

5.1 Tables Indexing

For BTown Properties, we created the following indexes:

- Table Property, B+ Tree index on price. This index will help speed up point and range queries over the properties' price. We believe this is a useful index because search on property's price is a very common operation when looking for properties.
- Table WatchesFeature, B+ Tree index on memberId and B+ Tree index on featureId. These indexes help speeds up the process of finding the features that are being watched by a member.
- Table WatchesProperty, Hash index on memberId and Hash index on propertyId. These indexes help speeds up the process of finding the properties that are being watched by a member. These are hash indexes because most queries on this table are point queries.
- Table Nearby, B+ Tree index on miles. This index will help speed up point and range queries over the distance of a Landmark. We believe this is a useful index because searches on property's distance are a common operation when looking for properties.
- Table Landmark, B+ Tree index on lat and B+ Tree index on lng. These indexes help speeds up the process of finding distances over landmarks.
- In general, a primary key column in MySQL is automatically indexed for efficiency. Therefore, all searches over primary keys, e.g., view property number n, will be fast.

6. Test

In this section, we present the methodology used to test the system.

6.1 Test-driven development cycle

The choice of Zend as a framework greatly simplified testing and the development cycle. We were able to construct the application navigation as a shell without any functionality, and the flow-control as a separate entity. Then, as each section was implemented, that section could be developed and tested in isolation from the others, without fear of disruption of other sections. Each section could then be tested on a use case basis, implementing the constraints as defined by the business logic.

Additionally, with foresight, we installed the framework on the designation host from the start of development, thus avoiding the issues associated with migration and deployment from a local host to an external one. With the use of an IDE(Integrated Development Environment) and ssh public/private keys, we were able to develop and test directly on the destination host, instead of developing the application locally, deploying the site on the destination server, then testing for functionality.

Furthermore, the Zend framework has integrated constraint checking for form fields, making implementation of data constraints not only straightforward, but, visible and easy to understand in the code.

6.2 Data point testing and calculation

Testing of the DIST stored procedure, which calculates the distance of a property to a landmark based on the latitude and longitude of the two locations, was done by generating 100,000 random points and iterating through them making the DIST calculation. The input domains of the two locations were varied to ensure the distance procedure handled as many cases as possible. Domains varied from a rectangle the size of Monroe County to a rectangle the size of the United States. It must be noted that this rigorous testing was highly costly

6.3 End User Testing

Finally, substantial end user testing was performed. Each team member thoroughly simulated the end user scenarios, as public, member, and admin users. This included the sign up and login process, the manipulation of properties, addition/ deletion of features, searching through the entire range of search combinations.

7. Appendices

7.1 Appendix A: System's Features Schematic

The interactions among the systems' features are depicted on the *system features schematic design* below. The dashed lines represent operations that can only be performed by members.