# CMPT 307.01 Databases

## GrifSpot Database

**Alex Brown**
**Tony Garcia**
**Charles Bailey**
**Jameson Reynolds**

## Summary

***GriftSpot (http://grifspot.cloudg.io)*** is a website that allows Westminster users to explore the area surrounding the college roughly within a 5 mile radius. By implementing a simple graphic-driven UI, students can search from seven categories and subsequently view photos, public ratings, and view reviews posted by other users on the many possible destinations. GrifSpot can be utilized on both computers, tablets, and mobile devices.

## Table of Contents



## Project Summary

***GrifSpot*** is a website in which users have the ability to chose what they want to explore around Westminster College and the local hangouts around Sugar House. This site, both mobile and computer supported, is both user focused and experience driven. The site is

supported by a database developed on a Play™ framework by our group. The database is currently hosted on an OS X server at: http://grifspot.cloudg.io Taking queues from apps like ***Scout*** and ***FieldTrip***, ***GrifSpot*** offers students and faculty of Westminster the opportunity to learn more about the big city Salt Lake while sticking around school. The application is primarily designed to promote exploration and a fully ubiquitous college experience.

***GrifSpot's*** database was designed using **MySQL WorkBench** in combination with **Sequel Pro.** This was necessary as to allow a connection to the MySQL server and then hard code POIs (point of interest) and other related information. They come together flawlessly to generate a useful and effective user experience.

One of the bigger challenges our team initially faced was attempting to create enough data tables. As most new projects do, this project steered us to eventually expand the number of tables to account for unique attributes. The team also experienced difficulty deciding what attributes would be most effective to use for each category, and example being *Education*, however we found attributes that bridged those gaps for a sound database design waiting to be implemented.

## Workflow

We began this project by first deciding what Westminster students would find useful out of our database. We had several meetings to discuss the many possible database tables and attributes to use. As the database tables were discussed, the group realized we could create additional tables linked through the use of foreign keys. After we created our EER Diagram through MySQL Workbench, the next logical step was to create the database and implement our discussed framework.

## Mission Statement

**To provide Westminster students with a free portal to experience and enjoy local life around the college.**

## Required tasks

### Must haves

- Our database must contain entries for each category
- A user can select/deselect every category

- Each Point of Interest has basic information based on the category
    - Example: Restaurant will have the type of food, specialties, price 1-5, as well as fast food, takeout, sit-down, etc.
- Point of Interests entries must have a picture

## Could haves (if time allows)

- Users can create individual accounts for the purpose of rating and writing comments on a Point of Interest
- Users can login/logout of the website
- Each Point of interest will have an average user rating as well as a comment section
- More information on recreational places would include if the activities are inside/outside, and if any specific equipment is needed for the activity.
- Badges are to be awarded to users for rating a point of interest, providing feedback through comments, as well as suggesting a point of interest
- Users would be allowed to suggest POIs and upload a new entries

# Data Modeling

The data modeling process went through several stages. After having an initial list of tasks, we created a list of Entities and their Attributes, giving an easy to use overview of the entire project.

# List of Entities and Attributes

## Comments

- Comment ID
- POI ID
- User ID
- Number of Stars
- Comment Text

## User

- User ID
- First Name
- Last Name
- User Name

## Restaurant

- Restaurant ID
- POI ID
- Menu
- Rest Type
- Specialty
- Zagats Rating
- Style
- Price
- Local
- POI ID
- Name
- Description
- Address
- Phone Number
- URL

## Transportation

- PT ID
- POI ID
- PT Type
- Fares

## Retail

- Retail ID
- POI ID
- Ret Type
- Local Business
- Used

## Education

- Education ID
- POI ID
- Edu Type
- Curriculum
- Calendar

## Social

- Social ID
- POI ID

- Minimum Age
- Smoking
- LGBT Friendly
- Occupency

## Architecture

- Architecture ID
- POI ID
- Architecture Type
- Purpose

## Recreational

- Recreational ID
- POI ID
- Services

## Vote

- Vote ID
- Comments ID
- User ID
- Thumbs Up

## Feedback

- Feedback ID
- Question Num
- Yes/Maybe/No

## POI_has_feedback

- POI ID
- Feedback ID

## Feedback_has_User

- Feedback ID
- User ID

## FeelingGriffy

- POI ID
- Category

## Badges

- Badges ID
- Badges JPG

## Badges_has_POI

- Badges ID
- POI ID

## Questions

- Questions ID

- Question

### Activities

- Activities ID
- Education ID
- Name

### IndoorOutdoor

- IndoorOutdoor ID
- Type

### Equipment

- Equipment ID
- Type

### Season

- Season ID
- Season

### Recreational_has_season

- Recreational ID
- Season ID

### Recreational_has_IndoorOutdoor

- Recreational ID
- IndoorOutdoor ID

### Recreational_has_Equipment

- Recreational ID
- Equipment ID

After creating the list of entities and attributes, we decided to create a logical schema. This allowed us to think about the data types we needed.

## Logical Schema

**Points_of_Interest(PointID, Name, Description, Address, PhoneNumber, URL, Hours)**

**Feedback(FeedbackID, Num_Stars, Comment, PointID)**

**Comments(CommentID, PointID, UserID, NumStars, CommentText)**

**User(UserID, FirstName, LastName, UserName, FeedbackID)**

**Restaurant(RestaurantID, PointID, Menu, Rest_Type, Phone, Specialty, ZagatRating, Style, Price, Local)**

**Transportation(PTID, PointID, PT_Type, Fares)**

**Social(SocialID, PointID, MinAge, 21+, Smoking, LGBT_Friendly, Occupancy)**

**Retail(RetailID, PointID, Ret_Type, LocalBusiness, Prodcuts_Used)**

**Education(EducationID, PointID, Edu_Type, Curriculum, Activites, Calendar, Sports, Performances)**

**Architecture(ArchitectureID, PointID, Arch_Type, Method, Purpose)**

**Recreational(RecreationalID, PointID, Services, Indoor, Outdoor, Equipment, Season)**

**Vote(VoteID, CommentsID, UserID,Thumbs Up)**

**Feedback(FeedbackID, QuestionNum, YMN)**

**POI_has_feedback(PointID, FeedbackID)**

**Feedback_has_User(FeedbackID, UserID)**

**FeelingGriffy(PointID, Category)**

**Badges(BadgesID, BadgesJPG)**

**Badges_has_POI(BadgesID, PointID)**

**Questions(QuestionsID, Question)**

**Activities(ActivitiesID, EducationID, Name)**

**IndoorOutdoor(IndoorOutdoorID, Type)**

**Equipment(EquipmentID, Type)**

**Season(SeasonID, Season)**

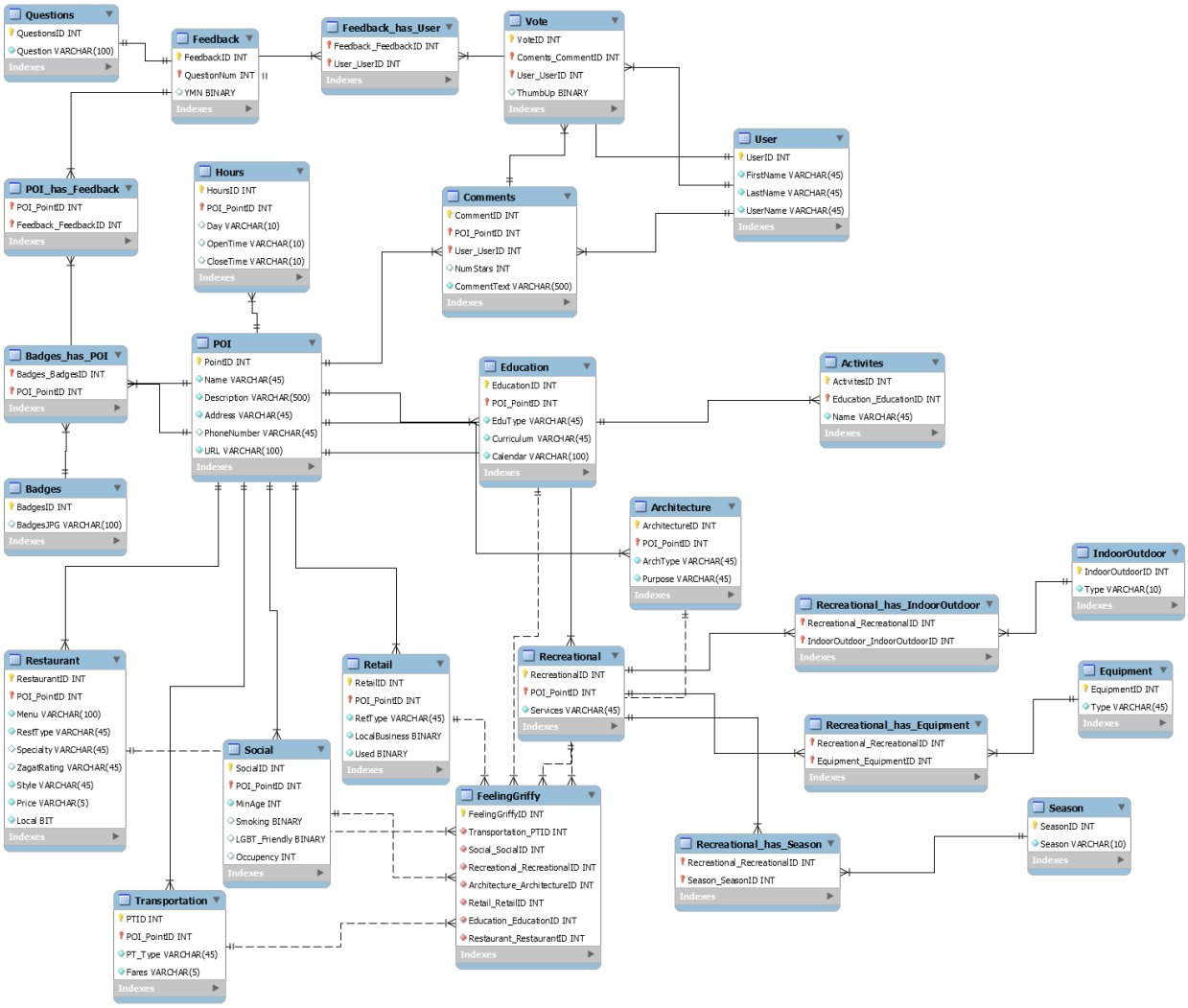
**Recreational_has_season(RecreationalID, SeasonID)**

**Recreational_has_IndoorOutdoor(RecreationalID, IndoorOutdoorID)**

**Recreational_has_Equipment(RecreationalID, EquipmentID)**

## EER Diagram

While reviewing the data that was still needed to implement appropriate datatypes, it was decided the group needed to create an EER Diagram. This EER diagram served as the third and final stage of database modeling. Using the EER Diagram proved helpful in seeing a visual overview of the relationships between the Entities.




***EER-Diagram shows all entities, their attributes and their relationships to each other.***

## Tables

The database for ***GrifSpot*** has 27 tables:

- Activities
- Architecture
- Badges
- Badges_has_POI
- Category
- Comments
- Education
- Equipment
- Feedback
- Feedback_has_User
- FeelingGriffy
- Hours
- IndoorOutdoor
- POI
- POI_has_Feedback
- Questions
- Recreational
- Recreational_has_Equipment
- Recreational_has_IndoorOutdoor
- Recreational_has_Season
- Restaurant
- Retail
- Season
- Social
- Transportation
- User
- Vote

On average, each table has roughly 20 entries, currently, ranging anywhere from 10 to 25 entries. This shall increase over the lifetime of the site, as users and admins will continue to add new data.

## Key Tables

The key tables in the database are POI, Restaurant, Architecture, Transportation, Retail, Education, Social, Recreational. They hold the most critical data. The generated content on

the website uses select statements based on these tables. The select statement easily access and retrieve the necessary information in the shortest amount of time.

## Database Interface/Application

We originally thought that the Sails.js framework would be the best fit for our project. As we brought up an instance of Sails.js we hit a brick wall when it came to creating the database models. We were under the impression that given a MySQL schema we would be able to generate models; this was not the case.

After some further research we found that the Play framework supports javaJpa plugins such as Hibernate which provide JDBC and JPA api dependencies. Using Hibernate as our ORM we were able to feed it our MySQL schema and have it generate models for each of our tables. To do this we had to add the Hibernate dependency to our project:

```
libraryDependencies ++= Seq(
  javaJpa,
  "org.hibernate" % "hibernate-entitymanager" % "3.6.9.Final" // replace by your
jpa implementation
)
```

JPA requires the datasource to be accessible via JNDI. To support Play-managed datasource via JNDI by adding this configuration in `conf/application.conf`:

```
db.default.driver=org.h2.Driver
db.default.url="jdbc:h2:mem:play"
db.default.jndiName=DefaultDS
```

Next, we had to create a proper `persistence.xml` JPA configuration file. Put it into the `conf/META-INF` directory, so it will be properly added to our classpath.

Here's what we used to use Hibernate:

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
          version="2.0">

   <persistence-unit name="defaultPersistenceUnit"
transaction-type="RESOURCE_LOCAL">
      <provider>org.hibernate.ejb.HibernatePersistence</provider>
      <non-jta-data-source>DefaultDS</non-jta-data-source>
      <properties>
```

```
        <property name="hibernate.dialect"
value="org.hibernate.dialect.H2Dialect"/>
      </properties>
   </persistence-unit>

</persistence>
```

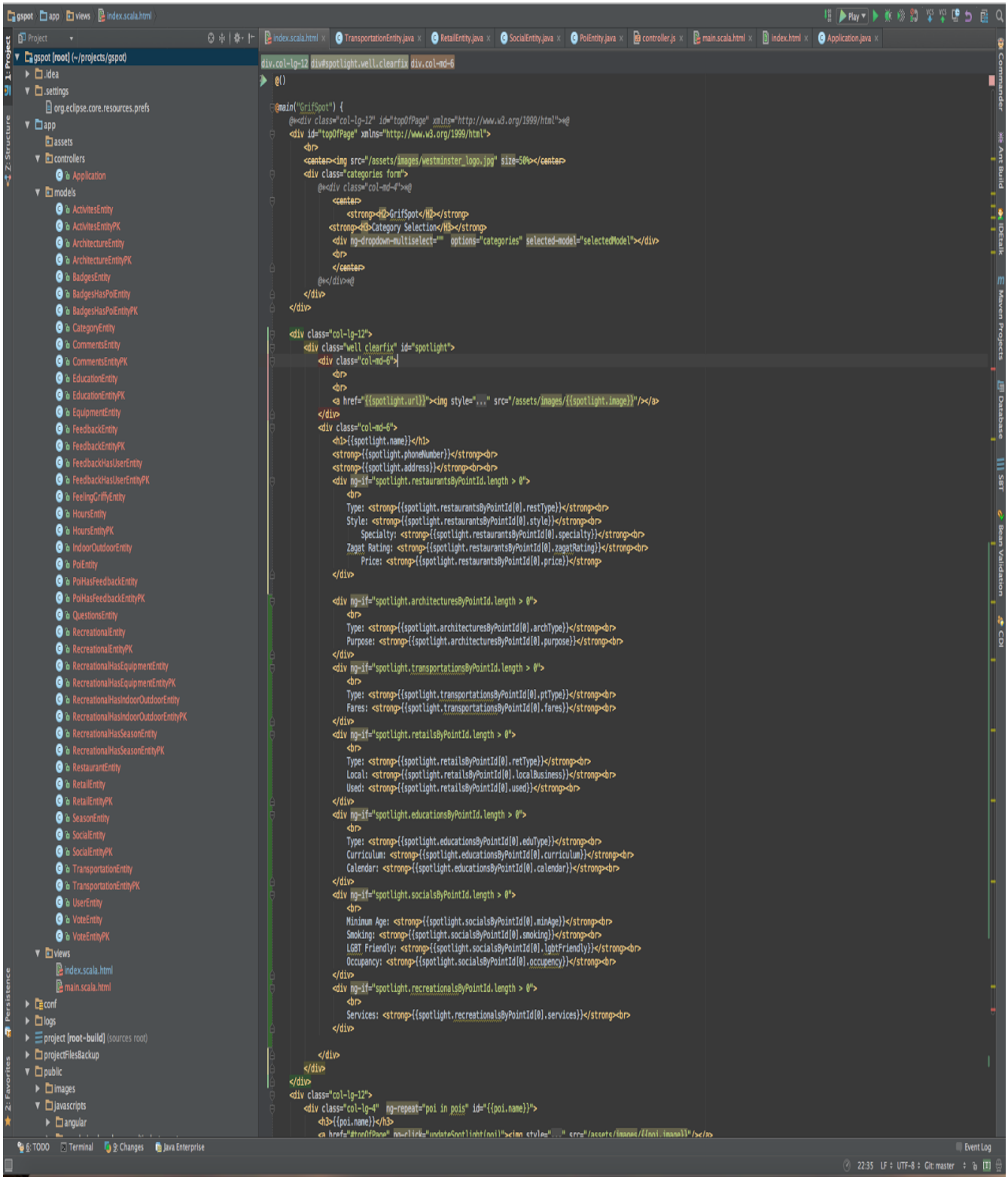
Finally, we had to tell Play which persistent unit should be used by your JPA provider. This was done by adding the `jpa.default` property in our `application.conf`.

```
jpa.default=defaultPersistenceUnit
```

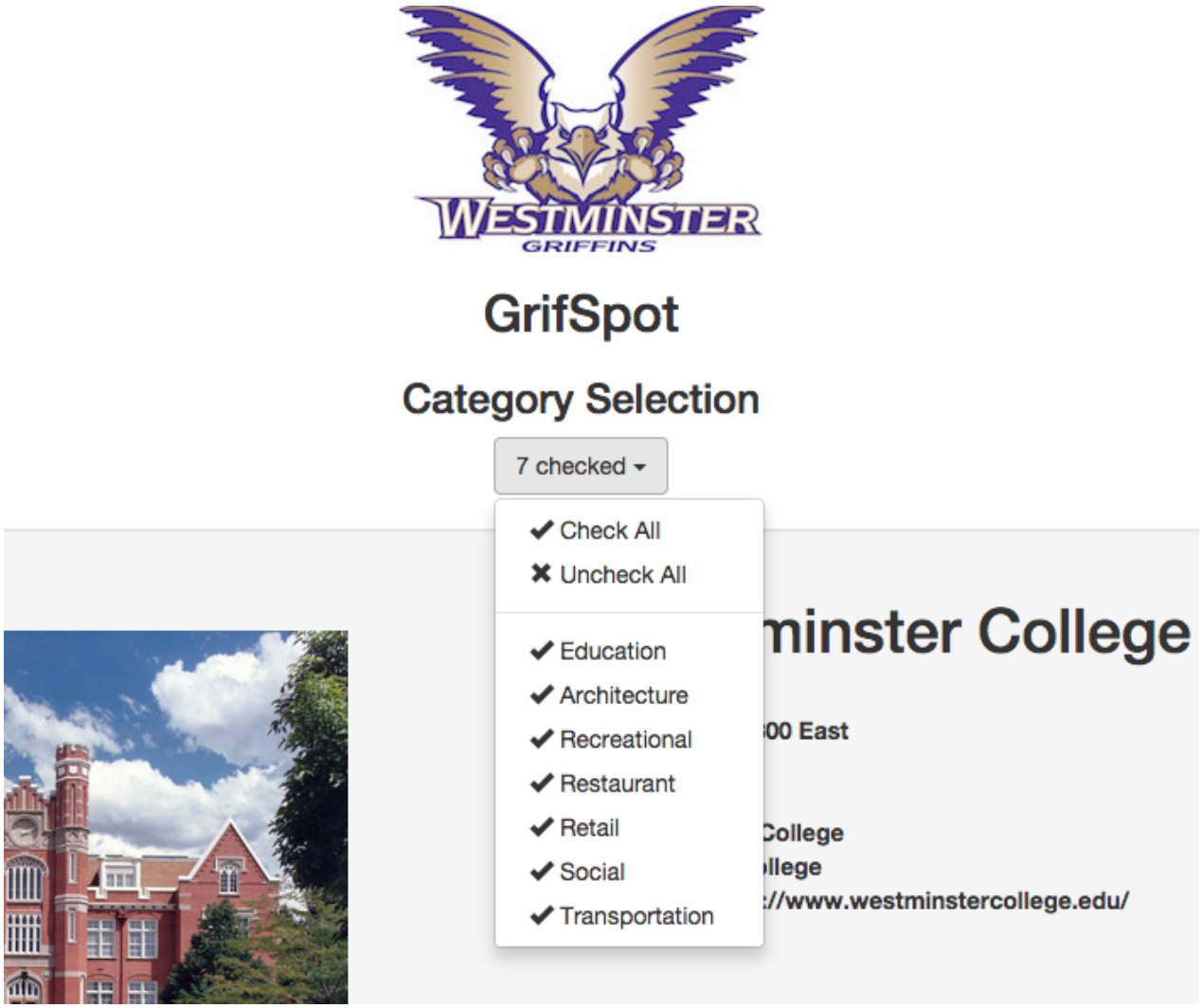
Since we were using an OS X server we had to install Brew to then install Play. Another challenge was that the OS X server has Apache running on port 80. Since you can only have one service bound to the same port, we had to add proxy forwarding to the httpd.conf file for GrifSpot.

```
<Proxy *>
        Order deny,allow
        Allow from all
        </Proxy>
        <Location />
        ProxyPass http://localhost:9000/
        ProxyPassReverse http://localhost:9000/
        </Location>
```

Play is based on Java and we used IntelliJ IDEA for our editor. Using the AngularJS libraries were were able to create one dynamic page that allowed for a fluid user experience.



The AngularJS framework also allowed us to incorporate a drop-down multi-selection menu, making it easy for our users to quickly choose multiple categories at once.



The end result was a dynamic, vibrant website that works on any device of your choosing.

http://GrifSpot.CloudG.io