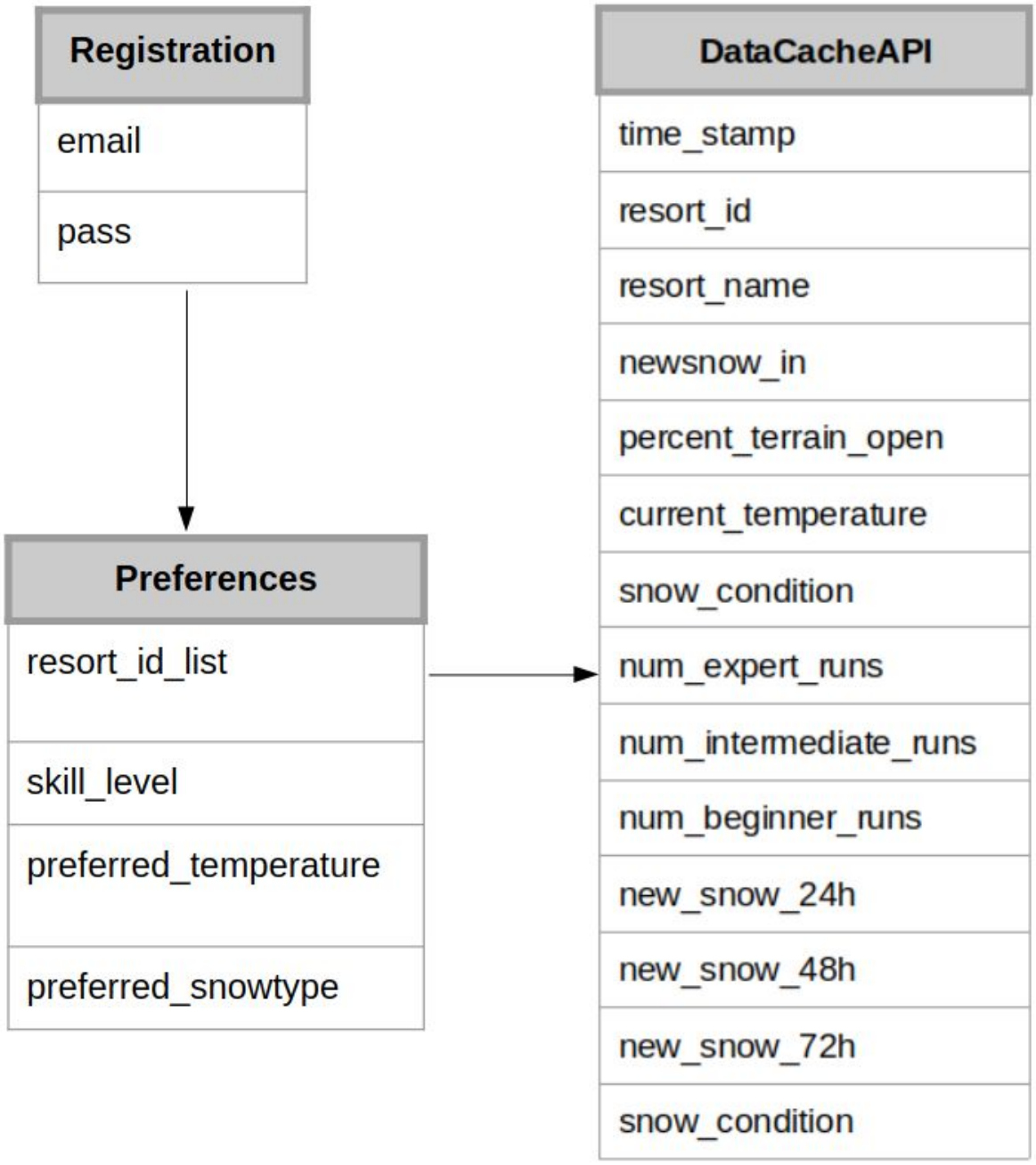
MileStone 4 - Database Design

Database Collections Overview

We have three collections in our database:

1) To store user login information.
2) To store preferences for a given user.
3) To store resort information pulled from the APIs.



Scripts Overview

We also have four scripts.

1) To pull data from the API and store it in the database
This script can be executed from either a get or a post request. Currently, it is only functional with a get request from a page on the website that must be manually visited. However, once the front end functionality is improved, this script will be able to execute in the background depending on a user's last login.

2) **To generate unique user accounts.** This is executed from the index.pug page, which takes user input to login.

3) **To input user preferences into the database.** This is executed from the cave.js file.

4) **To determine when a new API pull is needed** (based on user preferences and time since last pull.

The Documents are:

Database Structure: https://github.com/mskogen/APCS-BlizzardKicker/tree/dev/models

- Registration.js -- The database structure for storing user registrations
- Preferences.js -- The database structure for storing user preferences
- DataCacheAPI.js -- The database structure for storing resort data

Scripts: https://github.com/mskogen/APCS-BlizzardKicker/tree/dev/routes

- datapull.js -- a script to pull API data into the database
- register.js -- a script to add usernames and passwords to the correct collection
- userPrefs.js: a script to where the user can fill in their preferences to be saved to the database. Our plan is to use this script to call datapull.js when needed

DataBase Structures:

Preferences.js:

```
const mongoose = require('mongoose');

const userPrefsSchema = new mongoose.Schema({
  resort_id_list: {
    type: List<Number>,
  },
  skill_level : {
    type: Number, // scale 1-10?
  }
  preferred_temperature: {
    type: Number, // also a number.. calc as num degrees off from ideal
  },
  preferred_snowtype : {
    type: String, // string must be from a set only. Could turn string into list of ints?
  },
});

module.exports = mongoose.model('Preferences', userPrefsSchema);
```

Preferences	
Parameter	**Type**
resort_id_list	List<Number>
skill_level	Number
preferred_temperature	Number
preferred_snowtype	String

DataCacheAPI.js:

Note: In all mongodb structures, the unique key is the datamember _id. Normally, this key is a BSON format object that is generated by the mongoDB to be a unique identifier that incorporates information about the JSON object being stored. In our structure, the unique key has been overwritten to be an integer corresponding to the unique ski resort ID as defined by the API source. This has two important consequences: First, each report can be referenced directly by its _id (like an object in an array) rather than via the mongoDB .find() function. Second, because _id must be unique across an entire mongodb Collection, adding a new object with the same _id will overwrite the previous object. In this way, our code updates resorts with each pull rather than adding a new entry for each API call.

```
const mongoose = require('mongoose');

const dataCacheSchema = new mongoose.Schema({
// Do NOT store the entire JSON object
  _id: Number, // _id is a pre-defined Mongo field that MUST be a unique key.
  // Type cast it as a number rather than defaut BSON
  time_stamp: {
    type: Date,
  },
  resort_id: {
    type: Number,
  },
  resort_name: {
    type: String,
  },
  newsnow_in: {
    type: Number,
  },
  percent_terrain_open: {
    type: Number,
  },
  current_temperature: {
    type: Number,
  },
  num_expert_runs: {
    type: Number,
  },
  num_intermediate_runs: {
    type: Number,
  },
  num_beginner_runs: {
    type: Number,
```

```
  },
  new_snow_24h: {
    type: Number,
  },
  new_snow_48h: {
    type: Number,
  },
  new_snow_72h: {
    type: Number,
  },
  snow_condition: {
    type: String,
  },
});

module.exports = mongoose.model('DataCacheAPI', dataCacheSchema);
```

DataCacheAPI	
Parameter	**Type**
time_stamp	Date
resort_id	Number
resort_name	String
newsnow_in	Number
percent_terrain_open	Number
current_temperature	Number
snow_condition	Number
num_expert_runs	Number
num_intermediate_runs	Number
num_beginner_runs	Number

new_snow_24h	Number
new_snow_48h	Number
new_snow_72h	Number
snow_condition	String

Registration.js

```
const mongoose = require('mongoose');

const registrationSchema = new mongoose.Schema({
  email: {
    type: String,
    trim: true,
  },
  pass: {
    type: String,
    trim: true,
  },
});

module.exports = mongoose.model('Registration', registrationSchema);
```

Registration	
Parameter	**Type**
email	String
pass	String

This Is An Example of the JSON Object We Pull:


```
{
   "id": 482,
   "report": {
      "openflag": 1,
      "openflagname": "Open",
      "snowQuality": {
         "offSlope": {
            "surfaceBottom": "Machine Made",
            "surfaceBottomId": 3,
            "surfaceTop": "Packed Powder",
            "surfaceTopId": 2,
            "lowerDepth": 0.8,
            "upperDepth": 2.4,
            "middleDepth": 2.4
         },
         "onSlope": {
            "surfaceBottom": "Hard Packed",
            "surfaceBottomId": 5,
            "surfaceTop": "Hard Packed",
            "surfaceTopId": 5,
            "lowerDepth": 0.4,
            "upperDepth": 1.2,
            "middleDepth": 0.8
         },
         "snowCondition": "Good"
      },
      "terrainReport": {
         "trailsOpen": 5,
         "numExpertRuns": 5,
         "numAdvancedRuns": 5,
         "numIntermediateRuns": 5,
         "numBeginnerRuns": 5,
         "acresOpen": 5,
         "openLenBeginner": 5,
         "openLenIntermediate": 5,
         "openLenAdvanced": 5,
         "openLenExpert": 5,
         "openLenWeekend": 5,
         "openPerBeginner": 5,
         "openPerIntermediate": 5,
         "openPerAdvanced": 5,
         "openPerExpert": 5,
```

```
    "openPerWeekend": 5,
    "numKmOpen": 5
},
"snowParkReport": {
    "snowparksOpen": 5,
    "pipesOpen": 5,
    "jumpsOpen": 5,
    "parkReshapedDate": "2013-07-09",
    "parkReshapedEpoch": 1373328000,
    "pipesRecutDate": "2013-07-09",
    "pipesRecutEpoch": 1373328000
},
"nordicReport": {
    "xCtryKilometers": 5,
    "xcTrackset": 5,
    "xcSkateGroomed": 5
},
"liftsReport": {
    "liftsOpen": 5,
    "liftsOpenWeekend": 5,
    "returnBySki": "5",
    "returnBySkiText": "N/A",
    "perLiftsOpen": 0
},
"resortReportedWeather": {
    "topWeather": "5",
    "baseWeather": "3",
    "visibilityTop": 5,
    "visibilityTopText": "N/A",
    "visibilityBottom": 5,
    "visibilityBottomText": "N/A",
    "tempTop": 42,
    "tempBottom": 37,
    "windDirectionTop": 5,
    "windDirectionTopText": "N/A",
    "windForceTop": 5,
    "windDirectionBottom": 5,
    "windDirectionBottomText": "N/A",
    "windForceBottom": 5,
    "baseWeatherText": "N/A",
    "topWeatherText": "N/A",
    "avalancheScale": 5,
    "avalancheScaleText": "Very high"
```

```
        },
        "snowfall": {
            "snowReportedDate": "2013-07-16",
            "snow24h": 0.4,
            "snow48h": 0.8,
            "snow72h": 1.2,
            "callAheadPhone": "970-754-4888"
        },
        "openflagName": "Open"
    },
    "currentWeather": {
    "direction": "SE",
    "is_reported": false,
    "is_forecast": false,
    "tempTopMax": 43.8,
    "tempTopMin": 35.1,
    "tempBottomMax": 50.1,
    "tempBottomMin": 41.4,
    "tempTop": 38.2,
    "tempBottom": 44.6,
    "windspeed": 4.7,
    "weatherSymbol": "SUN",
    "bottomWinddirectionName": "SE",
    "bottomWindspeed": 4.7,
    "bottomWeatherSymbol": "SUN",
    "date": 1374019200,
    "date_str": "2013.07.17 00:00:00 UTC",
    "date_local_str": "2013.07.17 00:00:00 UTC",
    "date_local": 1374019200
    },
    "resortShortName": "Vail",
    "resortName": "Vail",
    "activeDate": "2013-07-04"

}
```