**CS 350 – DATABASE SYSTEMS**

**TERM PROJECT**

**PART 4: Implementation**

1. **Motivation and Requirements:**

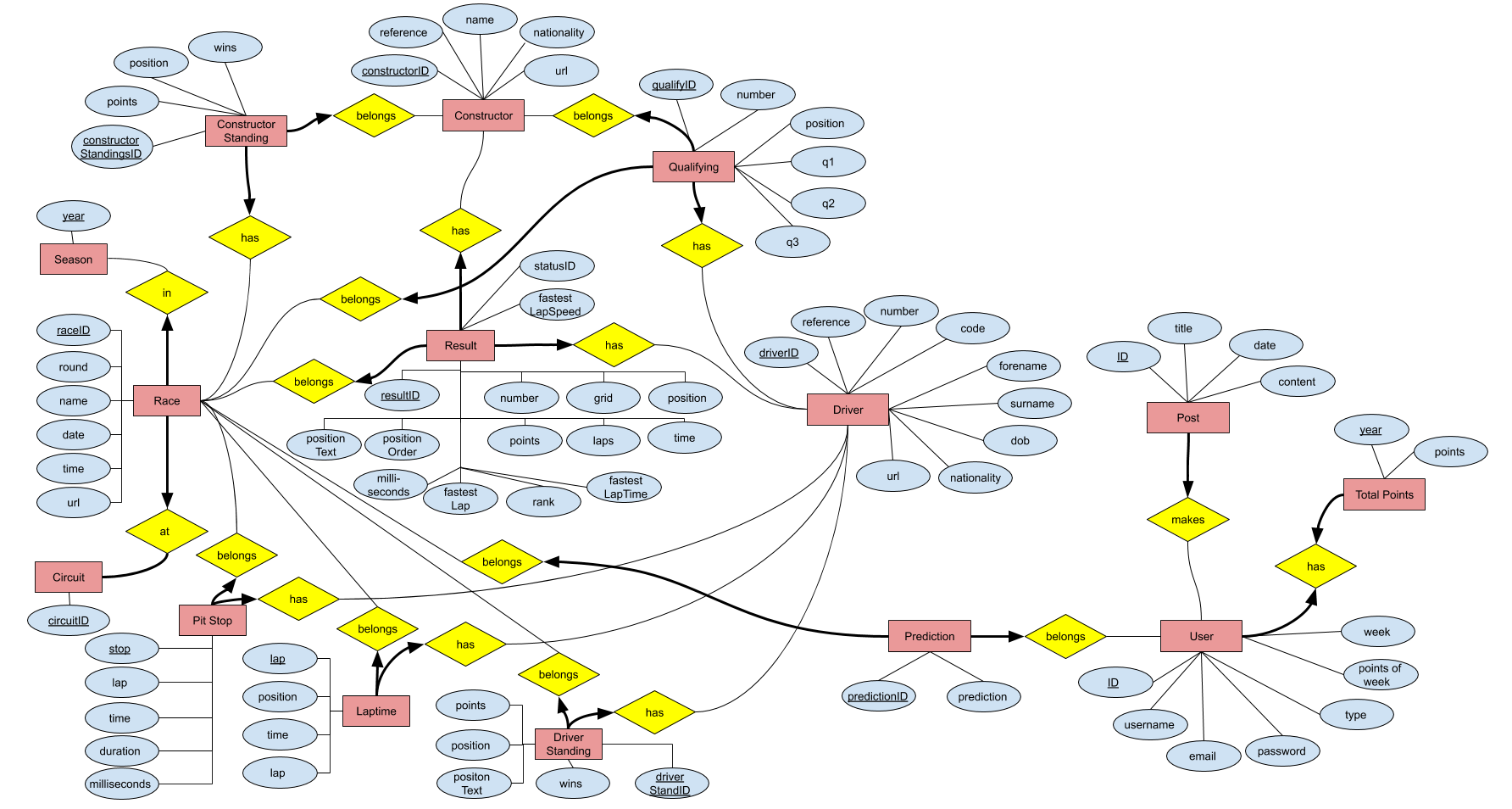
My Project is a tool for overviewing former and current Formula 1 race results, driver statistics, and race track statistics. All the data is available on but going through lots of pages to see specific results in relation to other results is really time consuming and there is no direct comparison available. The platform will also allow users to have more interaction with each other, which is always something fans are looking for, giving them a platform where they can post their ideas about upcoming and completed races and everything else correlated with Formula 1. They will also be able to make predictions about the upcoming race where they will be ranked afterwards to see how they compare to other Formula 1 fans around the world.

1. **Conceptual Database Design:**

**Data Requirements**

* Each circuit has an ID, reference, name, location, country, latitude, longitude, altitude and URL.
* Each circuit hosts **at least one** race.
* Each constructor has an ID, reference, name, nationality and URL.
* Each constructor has **0 or more** constructor standings, **0 or more** qualifying’s and **0 or more** results.
* Each constructor standing has an ID, points, position and number of wins.
* Each constructor standing has **exactly one** constructor and **exactly one** race.
* Each driver has an ID, reference, number, code, forename, surname, date of birth, nationality and URL.
* Each driver has **0 or more** pit stops, **0 or more** lap times, **0 or more** driver standings, **0 or more** qualifying’s and **0 or more** results.
* Each lap time has a lap number, position, time in h:mm:ss and millisecond format.
* Each lap time has **exactly one** race and **exactly one** driver.
* Each pit stop has a stop number, lap number, time, duration and time in milliseconds.
* Each pit stop has **exactly one** race and **exactly one** driver.
* Each qualifying has an ID, number, position and times for Q1, Q2 and Q3.
* Each qualifying has **exactly one** race, **exactly one** driver and **exactly one** constructor.
* Each race has an ID, round (x’th race), name, date, time, URL.
* Each race has **exactly one** season, **exactly one** constructor standing, **exactly one** circuit, **0 or more** pit stops, **0 or more** driver standings, **0 or more** results, **0 or more** qualifying’s, **0 or more** lap times and **0 or more** predictions.
* Each result has an ID, number, grid, position, points, laps count, time in hours, time in milliseconds, fastest lap, rank, fastest lap time, fastest lap speed and a status.
* Each result has **exactly one** race, **exactly one** constructor and **exactly one** driver.
* Each season has a year and URL.
* Each prediction has an ID and a prediction.
* Each prediction has **exactly one** race and **exactly one** user.
* Each user has an ID, a name, a surname, a password, a nick, a type, points of week score and a week counter.
* Each user has **0 or more** predictions, **0 or more** posts and **exactly one** total points score.
* Each total points score has a year and a points counter.
* Each total points score has **exactly one** user.

**ER Model**

****

1. **Logical Database Design:**
2. CREATE TABLE IF NOT EXISTS `f1db`.`circuits` (

`circuitId` INT NOT NULL AUTO\_INCREMENT,

`circuitRef` VARCHAR(255) NOT NULL DEFAULT '',

`name` VARCHAR(255) NOT NULL DEFAULT '',

`location` VARCHAR(255) NULL DEFAULT NULL,

`country` VARCHAR(255) NULL DEFAULT NULL,

`lat` FLOAT NULL DEFAULT NULL,

`lng` FLOAT NULL DEFAULT NULL,

`alt` INT NULL DEFAULT NULL,

`url` VARCHAR(255) NOT NULL DEFAULT '',

PRIMARY KEY (`circuitId`),

UNIQUE INDEX `url` (`url`))

ENGINE = MyISAM

AUTO\_INCREMENT = 75

DEFAULT CHARACTER SET = utf8;

1. CREATE TABLE IF NOT EXISTS `f1db`.`constructors` (

`constructorId` INT NOT NULL AUTO\_INCREMENT,

`constructorRef` VARCHAR(255) NOT NULL DEFAULT '',

`name` VARCHAR(255) NOT NULL DEFAULT '',

`nationality` VARCHAR(255) NULL DEFAULT NULL,

`url` VARCHAR(255) NOT NULL DEFAULT '',

PRIMARY KEY (`constructorId`),

UNIQUE INDEX `name` (`name`))

ENGINE = MyISAM

AUTO\_INCREMENT = 213

DEFAULT CHARACTER SET = utf8;

1. CREATE TABLE IF NOT EXISTS `f1db`.`constructorstandings` (

`constructorStandingsId` INT NOT NULL AUTO\_INCREMENT,

`raceId` INT NOT NULL DEFAULT '0',

`constructorId` INT NOT NULL DEFAULT '0',

`points` FLOAT NOT NULL DEFAULT '0',

`position` INT NULL DEFAULT NULL,

`positionText` VARCHAR(255) NULL DEFAULT NULL,

`wins` INT NOT NULL DEFAULT '0',

PRIMARY KEY (`constructorStandingsId`),

FOREIGN KEY (`raceId`) REFERENCES `f1db`.`races` (`raceId`),

FOREIGN KEY (`constructorId`) REFERENCES `f1db`.`constructors` (`constructorId`))

ENGINE = MyISAM

AUTO\_INCREMENT = 27473

DEFAULT CHARACTER SET = utf8;

1. CREATE TABLE IF NOT EXISTS `f1db`.`drivers` (

`driverId` INT NOT NULL AUTO\_INCREMENT,

`driverRef` VARCHAR(255) NOT NULL DEFAULT '',

`number` INT NULL DEFAULT NULL,

`code` VARCHAR(3) NULL DEFAULT NULL,

`forename` VARCHAR(255) NOT NULL DEFAULT '',

`surname` VARCHAR(255) NOT NULL DEFAULT '',

`dob` DATE NULL DEFAULT NULL,

`nationality` VARCHAR(255) NULL DEFAULT NULL,

`url` VARCHAR(255) NOT NULL DEFAULT '',

PRIMARY KEY (`driverId`),

UNIQUE INDEX `url` (`url`))

ENGINE = MyISAM

AUTO\_INCREMENT = 850

DEFAULT CHARACTER SET = utf8;

1. CREATE TABLE IF NOT EXISTS `f1db`.`driverstandings` (

`driverStandingsId` INT NOT NULL AUTO\_INCREMENT,

`raceId` INT NOT NULL DEFAULT '0',

`driverId` INT NOT NULL DEFAULT '0',

`points` FLOAT NOT NULL DEFAULT '0',

`position` INT NULL DEFAULT NULL,

`positionText` VARCHAR(255) NULL DEFAULT NULL,

`wins` INT NOT NULL DEFAULT '0',

PRIMARY KEY (`driverStandingsId`),

FOREIGN KEY (`raceId`) REFERENCES `f1db`.`races` (`raceId`),

FOREIGN KEY (`driverId`) REFERENCES `f1db`.`drivers` (`driverId`))

ENGINE = MyISAM

AUTO\_INCREMENT = 69789

DEFAULT CHARACTER SET = utf8;

1. CREATE TABLE IF NOT EXISTS `f1db`.`laptimes` (

`raceId` INT NOT NULL,

`driverId` INT NOT NULL,

`lap` INT NOT NULL,

`position` INT NULL DEFAULT NULL,

`time` VARCHAR(255) NULL DEFAULT NULL,

`milliseconds` INT NULL DEFAULT NULL,

PRIMARY KEY (`raceId`, `driverId`, `lap`),

FOREIGN KEY (`raceId`) REFERENCES `f1db`.`races` (`raceId`),

FOREIGN KEY (`driverId`) REFERENCES `f1db`.`drivers` (`driverId`))

ENGINE = MyISAM

DEFAULT CHARACTER SET = utf8;

1. CREATE TABLE IF NOT EXISTS `f1db`.`pitstops` (

`raceId` INT NOT NULL,

`driverId` INT NOT NULL,

`stop` INT NOT NULL,

`lap` INT NOT NULL,

`time` TIME NOT NULL,

`duration` VARCHAR(255) NULL DEFAULT NULL,

`milliseconds` INT NULL DEFAULT NULL,

PRIMARY KEY (`raceId`, `driverId`, `stop`),

FOREIGN KEY (`raceId`) REFERENCES `f1db`.`races` (`raceId`),

FOREIGN KEY (`driverId`) REFERENCES `f1db`.`drivers` (`driverId`))

ENGINE = MyISAM

DEFAULT CHARACTER SET = utf8;

1. CREATE TABLE IF NOT EXISTS `f1db`.`qualifying` (

`qualifyId` INT NOT NULL AUTO\_INCREMENT,

`raceId` INT NOT NULL DEFAULT '0',

`driverId` INT NOT NULL DEFAULT '0',

`constructorId` INT NOT NULL DEFAULT '0',

`number` INT NOT NULL DEFAULT '0',

`position` INT NULL DEFAULT NULL,

`q1` VARCHAR(255) NULL DEFAULT NULL,

`q2` VARCHAR(255) NULL DEFAULT NULL,

`q3` VARCHAR(255) NULL DEFAULT NULL,

PRIMARY KEY (`qualifyId`),

FOREIGN KEY (`raceId`) REFERENCES `f1db`.`races` (`raceId`),

FOREIGN KEY (`driverId`) REFERENCES `f1db`.`drivers` (`driverId`),

FOREIGN KEY (`constructorId`) REFERENCES `f1db`.`constructors` (`constructorId`))

ENGINE = MyISAM

AUTO\_INCREMENT = 8378

DEFAULT CHARACTER SET = utf8;

1. CREATE TABLE IF NOT EXISTS `f1db`.`races` (

`raceId` INT NOT NULL AUTO\_INCREMENT,

`year` INT NOT NULL DEFAULT '0',

`round` INT NOT NULL DEFAULT '0',

`circuitId` INT NOT NULL DEFAULT '0',

`name` VARCHAR(255) NOT NULL DEFAULT '',

`date` DATE NOT NULL DEFAULT '0000-00-00',

`time` TIME NULL DEFAULT NULL,

`url` VARCHAR(255) NULL DEFAULT NULL,

PRIMARY KEY (`raceId`),

FOREIGN KEY (`circuitId`) REFERENCES `f1db`.`circuits` (`circuitId`),

FOREIGN KEY (`year`) REFERENCES `f1db`.`seasons` (`year`),

UNIQUE INDEX `url` (`url`))

ENGINE = MyISAM

AUTO\_INCREMENT = 1053

DEFAULT CHARACTER SET = utf8;

1. CREATE TABLE IF NOT EXISTS `f1db`.`seasons` (

`year` INT NOT NULL DEFAULT '0',

`url` VARCHAR(255) NOT NULL DEFAULT '',

PRIMARY KEY (`year`),

UNIQUE INDEX `url` (`url`))

ENGINE = MyISAM

DEFAULT CHARACTER SET = utf8;

1. CREATE TABLE IF NOT EXISTS `f1db`.`results` (

`resultId` INT NOT NULL AUTO\_INCREMENT,

`raceId` INT NOT NULL DEFAULT '0',

`driverId` INT NOT NULL DEFAULT '0',

`constructorId` INT NOT NULL DEFAULT '0',

`number` INT NULL DEFAULT NULL,

`grid` INT NOT NULL DEFAULT '0',

`position` INT NULL DEFAULT NULL,

`positionText` VARCHAR(255) NOT NULL DEFAULT '',

`positionOrder` INT NOT NULL DEFAULT '0',

`points` FLOAT NOT NULL DEFAULT '0',

`laps` INT NOT NULL DEFAULT '0',

`time` VARCHAR(255) NULL DEFAULT NULL,

`milliseconds` INT NULL DEFAULT NULL,

`fastestLap` INT NULL DEFAULT NULL,

`rank` INT NULL DEFAULT '0',

`fastestLapTime` VARCHAR(255) NULL DEFAULT NULL,

`fastestLapSpeed` VARCHAR(255) NULL DEFAULT NULL,

`statusId` INT NOT NULL DEFAULT '0',

PRIMARY KEY (`resultId`),

FOREIGN KEY (`raceId`) REFERENCES `f1db`.`races` (`raceId`),

FOREIGN KEY (`driverId`) REFERENCES `f1db`.`drivers` (`driverId`),

FOREIGN KEY (`constructorId`) REFERENCES `f1db`.`constructors` (`constructorId`))

ENGINE = MyISAM

AUTO\_INCREMENT = 24626

DEFAULT CHARACTER SET = utf8;

1. CREATE TABLE IF NOT EXISTS `f1db`.`users` (

`user\_id` INT NOT NULL AUTO\_INCREMENT,

`user\_name` VARCHAR(255) NOT NULL,

`user\_email` VARCHAR(255) NOT NULL,

`user\_password` VARCHAR(255) NOT NULL,

`user\_type` VARCHAR(255) NOT NULL DEFAULT 'normal',

`points\_of\_week` INT NOT NULL DEFAULT '0',

`week` INT NOT NULL DEFAULT '0',

`user\_img` VARCHAR(255) NOT NULL DEFAULT 'default.jpg',

PRIMARY KEY (`user\_id`),

UNIQUE INDEX `user\_email\_UNIQUE` (`user\_email`),

UNIQUE INDEX `user\_name\_UNIQUE` (`user\_name`))

ENGINE = InnoDB

AUTO\_INCREMENT = 15

DEFAULT CHARACTER SET = utf8;

1. CREATE TABLE IF NOT EXISTS `f1db`.`total\_points` (

`user\_id` INT NOT NULL,

`year` INT NOT NULL,

`points` VARCHAR(255) NULL DEFAULT NULL,

PRIMARY KEY (`user\_id`, `year`),

FOREIGN KEY (`user\_id`) REFERENCES `f1db`.`users` (`user\_id`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8;

1. CREATE TABLE IF NOT EXISTS `f1db`.`user\_predictions` (

`user\_id` INT NOT NULL,

`prediction\_id` INT NOT NULL,

`race\_id` INT NOT NULL,

`prediction` VARCHAR(255) NULL DEFAULT NULL,

PRIMARY KEY (`prediction\_id`, `user\_id`),

FOREIGN KEY (`raceId`) REFERENCES `f1db`.`races` (`raceId`),

FOREIGN KEY (`user\_id`) REFERENCES `f1db`.`users` (`user\_id`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8;

1. CREATE TABLE IF NOT EXISTS `f1db`.`posts` (

`post\_id` INT NOT NULL AUTO\_INCREMENT,

`post\_title` VARCHAR(45) NOT NULL,

`post\_date` DATETIME NOT NULL,

`post\_content` VARCHAR(255) NOT NULL,

`user\_id` INT NOT NULL,

PRIMARY KEY (`post\_id`),

FOREIGN KEY (`user\_id`) REFERENCES `f1db`.`users` (`user\_id`))

ENGINE = InnoDB

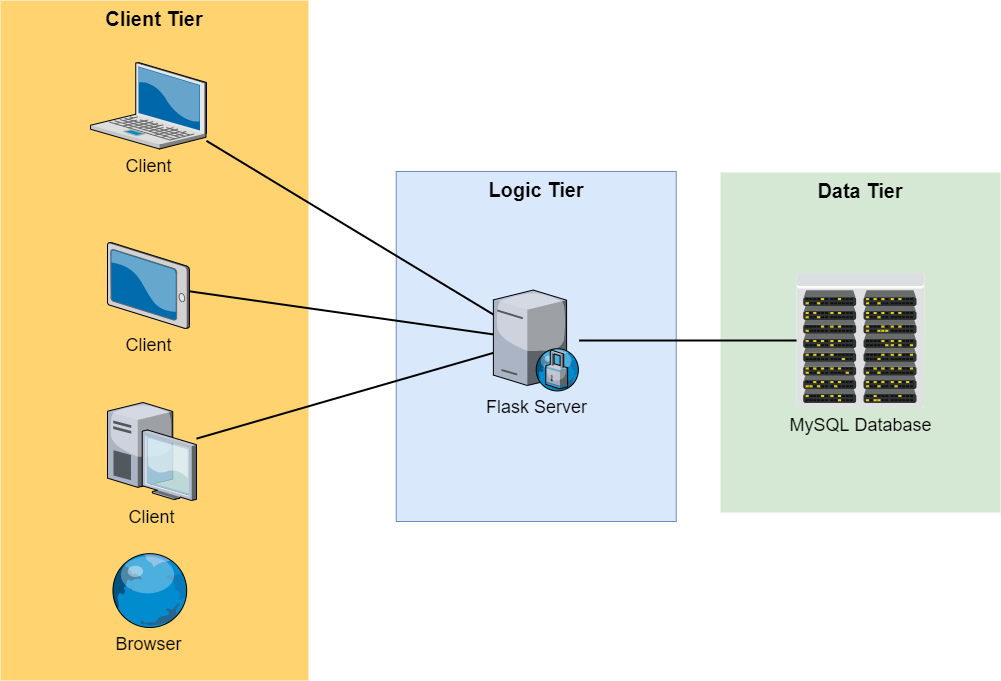
AUTO\_INCREMENT = 16

DEFAULT CHARACTER SET = utf8;

1. **Application Design and Implementation:**

# First of all I want to quickly state that I followed the YouTube tutorial series of Corey Schafer: Python Flask Tutorial: Full-Featured Web App, which may or may not cause in similarities to other students and or friends that I recommended this series to.

* 1. **Architecture**



My application has a three tier architecture consisting of client, logic and data tiers. The client tier is responsible for displaying the information of the application and provide options to jump to other websites, provide fields for input and notify the user of any changes. In my case this could be showing the posts of the users, showing the login and registration pages where the user can insert data, etc.

The logic tier handles the requests coming from the client tier such as requests to go to another page or reacting to data that wants to be submitted by the user. E.g., if the user wants to login the server has to send the request to the database for validation and then has to process the answer coming from the database in order to give the user feedback about his action.  
 The data tier is responsible to execute the queries that have been send by the server and then return the answers to these queries back to the server in order for the server to process them. A applicable example for my application could be the user wants to log in and the server sends the database a query to check if the user is already existing. If yes then the database returns the user with the corresponding data to the server who checks if all the provided input is valid e.g. the password for this user.

* 1. **Queries**
     + select user\_name, user\_img   
       from users   
       where user\_id={user\_id}  
       ***Semantics:*** Return the user with the user\_id that has been requested.
     + select \*   
       from users  
       ***Semantics:*** Return all users with all attributes from the users table.
     + select year   
       from seasons  
       ***Semantics:*** Return all years from the seasons.
     + select forename, surname   
       from drivers  
       ***Semantics:*** Return the forename and surname of all drivers from the drivers table.
     + select name from constructors  
       ***Semantics:*** Return the names of all the constructors.
     + select name   
       from circuits  
       ***Semantics:*** Return the names of all circuits.
     + select status from status  
       ***Semantics:*** Return all status names.
     + select \*   
       from posts  
       ***Semantics:*** Return all posts with all columns.
     + select round   
       from races   
       where year={int(season)}  
       ***Semantics:*** Return the round number for the races from the specified season.
     + insert into users(user\_name, user\_email, user\_password)  
        values(%s,%s,%s), (user\_name, user\_email, user\_password)  
       ***Semantics:*** Insert the user data that was provided in the registration form to the database.
     + select \*   
       from users   
       where user\_email = '{form.email.data}'  
       ***Semantics:*** Return the user that has the corresponding email that was given.
     + update users   
       set user\_img='{picture\_fn}'   
       where user\_id='{current\_user.id}'  
       ***Semantics:*** Update the user profile picture of the current user.
     + insert into posts(post\_title, post\_date, post\_content, user\_id)"   
        "values(%s,%s,%s,%s)", (form.title.data, utc\_to\_local(datetime.utcnow()), form.content.data, current\_user.id)  
       ***Semantics:*** Insert the new post that was created into the database with all corresponding details.
     + select \* from posts   
       where post\_id='{post\_id}' ***Semantics:*** Return the post with the provided post id with all corresponding data.
     + update posts   
       set post\_title='{form.title.data}', post\_content='{form.content.data}'  
       where post\_id='{p[0]}'  
       ***Semantics:*** Update the post with the give post id with the corresponding data.
     + delete   
       from posts   
       where post\_id='{p[0]}'  
       ***Semantics:*** Delete the post with the given post id from the database.

* + - 1. **Dynamic Queries**

The Following queries are dynamic, i.e. they are dependent on filters. If a filter is chosen the where clauses at the end of this section are applied. Some of the join clauses are already being added due to the fact that that some of the data is required for the results that are going to be displayed. For certain queries the is also an order by clause that is added to the query at the end. For one of the queries there is also a group by clause that is applied.

* + - select distinct circuits.name, circuits.location, circuits.country, circuits.url   
      from circuits  
        
      **Join Clause Options**join races on circuits.circuitId = races.circuitId  
      join results on races.raceId = results.raceId  
      join drivers on results.driverId = drivers.driverId  
      join constructors on results.constructorId = constructors.constructorId  
      join status on results.statusId = status.statusId  
        
      ***Semantics:*** Return all circuits with their corresponding names, locations, countries and url according to the applied filters.
    - select distinct constructors.name, constructors.nationality, constructors.url   
      from constructors   
       **Join Clause Options**join results on constructors.constructorId = results.constructorId  
      join races on races.raceId = results.raceId  
      join drivers on results.driverId = drivers.driverId  
      join status on results.statusId = status.statusId  
      join circuits on races.circuitId = circuits.circuitId  
      ***Semantics:*** Return all constructors with their corresponding names, nationalities and url according to the applied filter.
    - select distinct concat\_ws(' ', drivers.forename, drivers.surname) as name, drivers.number, drivers.nationality, drivers.dob, drivers.url   
      from drivers  
        
      **Join Clause Options**join results on drivers.driverId = results.driverId  
      join races on races.raceId = results.raceId  
      join constructors on results.constructorId = constructors.constructorId  
      join status on results.statusId = status.statusId  
      join circuits on races.circuitId = circuits.circuitId  
        
      ***Semantics:*** Return all drivers with their corresponding forenames, surnames, permanent driver numbers, nationalities and url according to the applied filters.
    - select distinct concat\_ws(' ', races.year, races.name) as race, qualifying.position, concat\_ws(' ', drivers.forename, drivers.surname) as name, constructors.name, qualifying.q1, qualifying.q2, qualifying.q3   
      from qualifying  
        
      **Join Clause Options**join results on drivers.driverId = results.driverId   
      join races on races.raceId = qualifying.raceId   
      join constructors on qualifying.constructorId = constructors.constructorId   
      join status on results.statusId = status.statusId   
      join circuits on races.circuitId = circuits.circuitId   
      join drivers on qualifying.driverId = drivers.driverId   
        
      ***Semantics:*** Return all qualifying results with their corresponding races, qualifying positions, drivers, construcors and Q1, Q2, Q3 times according to the applied filters.

* + - select distinct concat\_ws(' ', races.year, races.name) as race, results.positionText, concat\_ws(' ', drivers.forename, drivers.surname) as name, constructors.name, results.laps, results.grid, results.time, status.status, results.points   
      from results  
        
      **Join Clause Options**join races on races.raceId = results.raceId  
      join constructors on results.constructorId = constructors.constructorId  
      join status on results.statusId = status.statusId  
      join drivers on results.driverId = drivers.driverId  
      join circuits on races.circuitId = circuits.circuitId  
        
      **Order By Clause**order by races.year, races.round, results.positionOrder  
        
      ***Semantics:*** Return all Race Results with their corresponding final positon, driver names, constructor names, amount of laps completed, race grid position, final racing time, final status and points won according to the applied filters.
    - select distinct races.year, races.round, races.name, races.date, circuits.name, circuits.location, circuits.country, races.url   
      from races  
        
      **Join Clause Options**join circuits on races.circuitId = circuits.circuitId  
      join results on races.raceId = results.raceId  
      join constructors on results.constructorId = constructors.constructorId  
      join drivers on results.driverId = drivers.driverId  
      join status on results.statusId = status.statusId  
        
      **Order By Clause**order by races.round

***Semantics:*** Return all races with their corresponding year, round, circuit name, circuit location, and url according to the applied filters.

* + - select distinct seasons.year, seasons.url   
      from seasons  
        
      **Join Clause Options**join races on seasons.year = races.year  
      join results on races.raceId = results.raceId  
      join drivers on results.driverId = drivers.driverId  
      join constructors on results.constructorId = constructors.constructorId  
      join circuits on races.circuitId = circuits.circuitId  
      join status on results.statusId = status.statusId

**Order By Clause**order by seasons.year

***Semantics:*** Return all seasons with their corresponding year and url according to the applied filters.

* + - select distinct status.status, count(results.statusId) as count   
      from results  
        
      **Join Clause Options**join status on results.statusId = status.statusId  
      join races on results.raceId = races.raceId  
      join constructors on results.constructorId = constructors.constructorId  
      join drivers on results.driverId = drivers.driverId  
      join circuits on races.circuitId = circuits.circuitId  
        
      **Order By Clause**order by status.statusId

**Group By Clause**group by status.status  
  
***Semantics:*** Return all drivers with their corresponding forenames, surnames, permanent driver numbers, nationalities and url according to the applied filters.

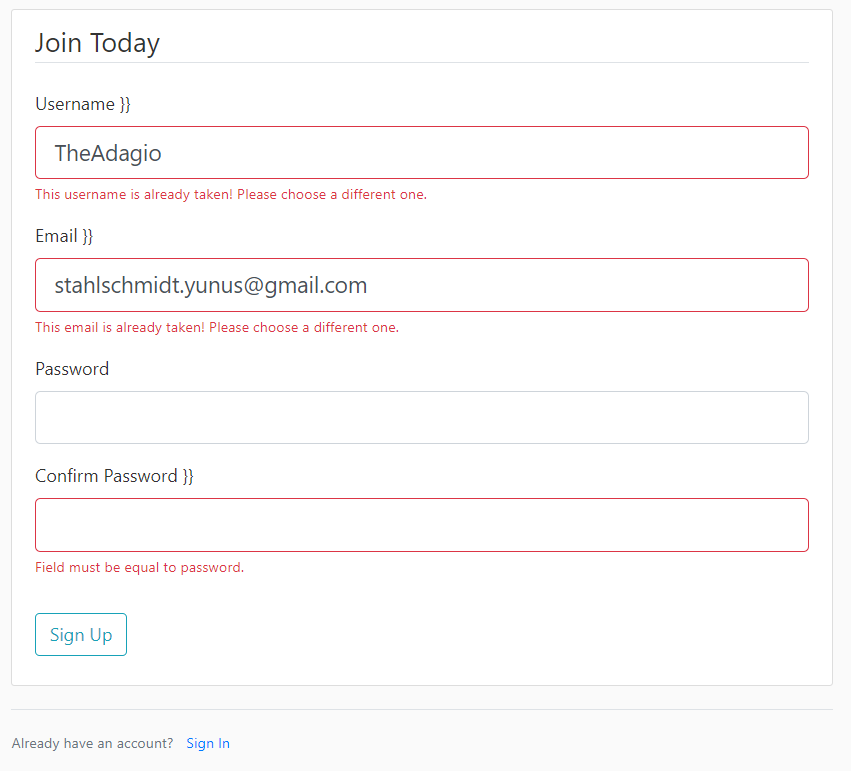
* + - 1. **Where Clauses**

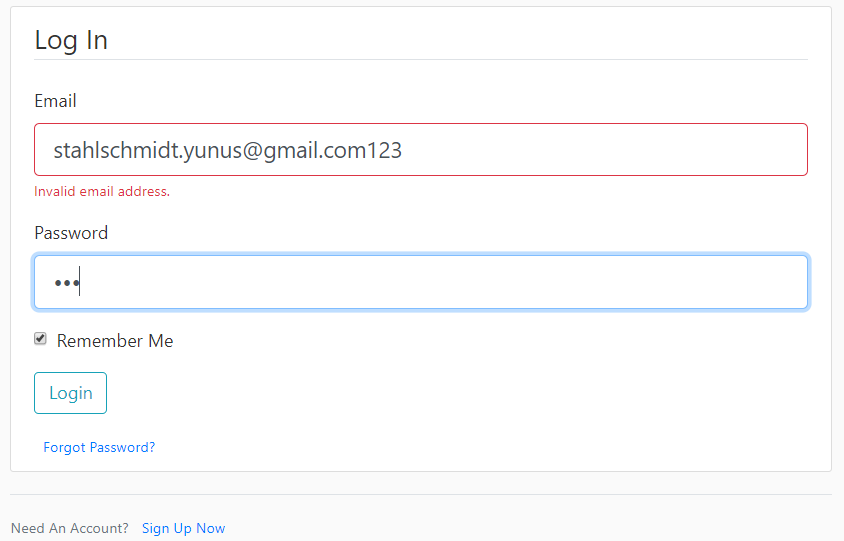
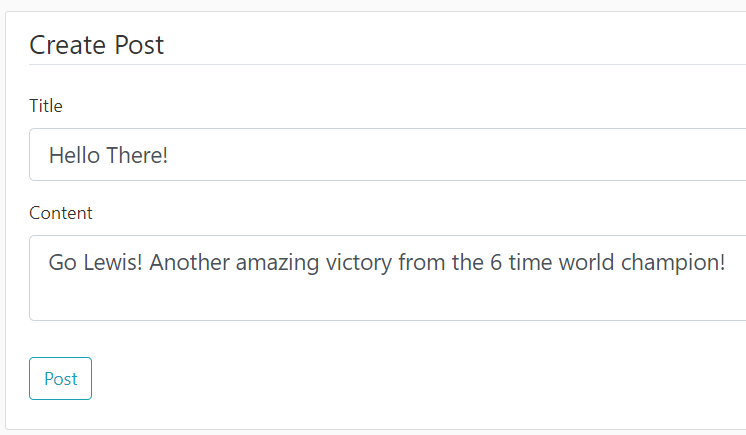
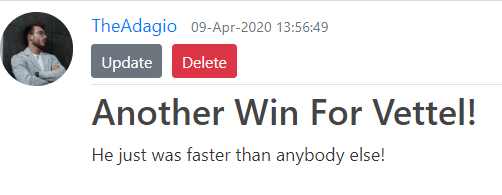
These clauses are added to the query depending on which filters have been selected in the form. If more than one filter is selected then of course with an and clause in between.

- where races.year = {season}  
- and races.round = {round}  
- where drivers.forename = '{driver\_forename}' and drivers.surname = '{driver\_surname}  
- where constructors.name = '{const}'  
- where results.positionText = '{fin\_pos}'  
- where results.grid = '{grid}'  
- where results.grid = '{grid}'  
- where results.rank = '{fast\_lap\_rank}'  
- where circuits.name = '{circuit}'  
- where status.status = '{status}'

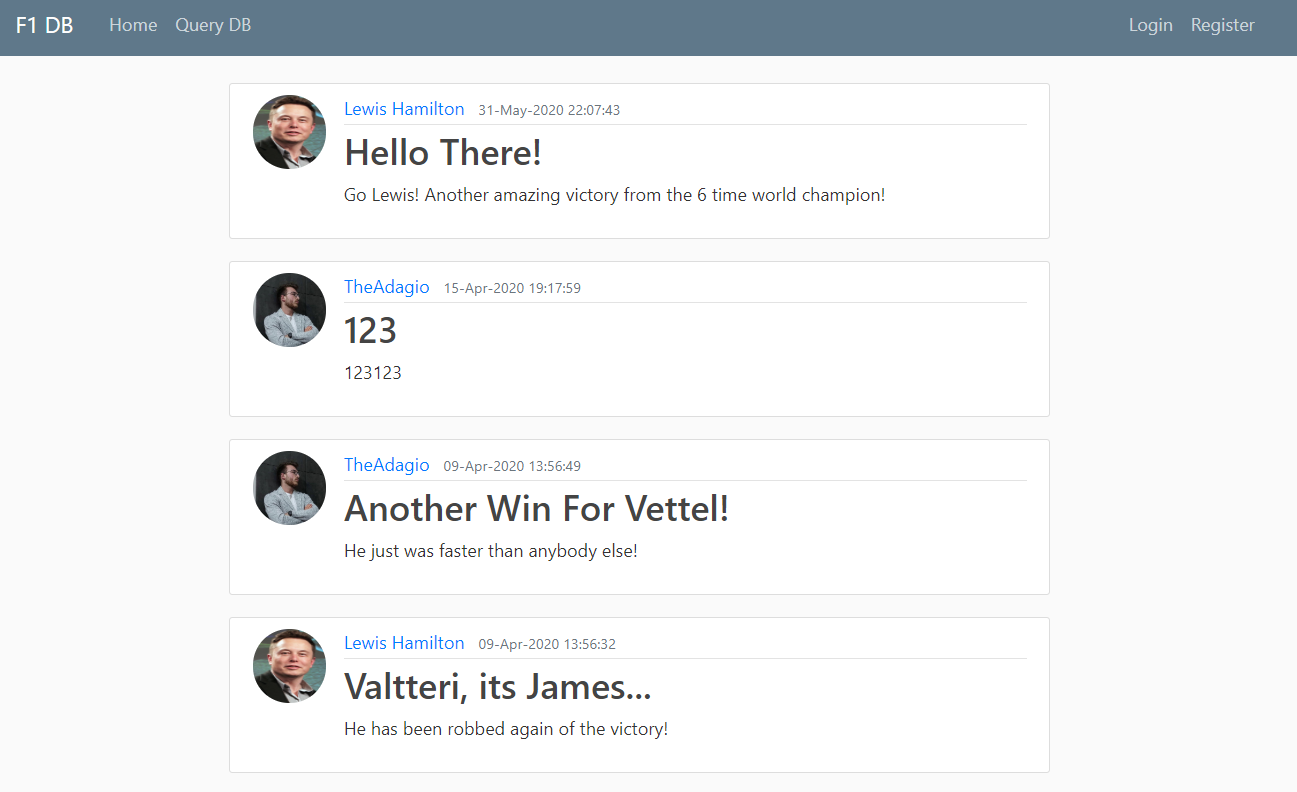
* 1. **Programming Languages & DBMS**
* Python
* Java Script
* MySQL

* 1. **Data Size & Source**
*  (Screenshot of current DB Size)
* I got my data for the Formula 1 related part from the Ergast Developer API at: <http://ergast.com/mrd/db/> which publishes the most up to date database images shortly after the race.
* Circuits Table: 74 Rows
* Constructors Table: 210 Rows
* Constructor Standings Table: 12.326 Rows
* Drivers Table: 848 Rows
* Driver Standings Table: 32.586 Rows
* Lap Times Table: 472.504 Rows
* Pit Stops Table: 7.436 Rows
* Posts Table: 15 Rows
* Qualifying Table: 8.354 Rows
* Races Table: 1.033 Rows
* Results Table: 24.620 Rows
* Seasons Table: 71 Rows
* Status Table: 135 Rows
* Total Points Table: 0 Rows
* User Predictions Table: 0 Rows
* Users Table: 5 Rows
  1. **Working Features**
  + Users can register to the website, where their password is stored as a hash to ensure security. The website also tells them if their selected username or provided email is already being used or the entered passwords don’t match or are too short / long.

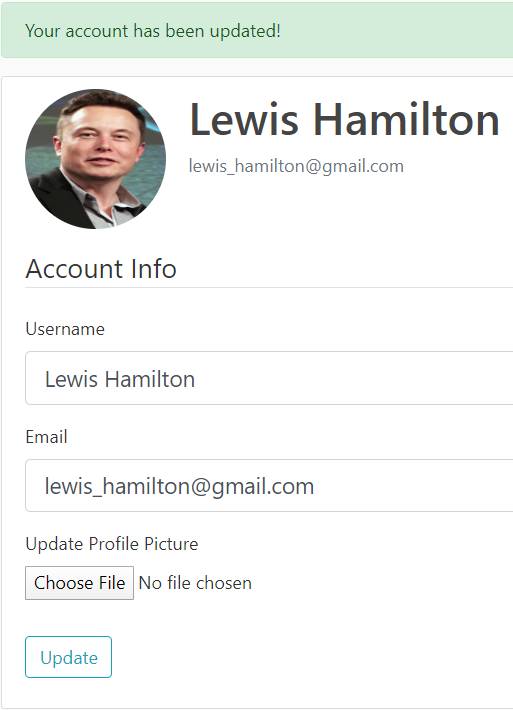
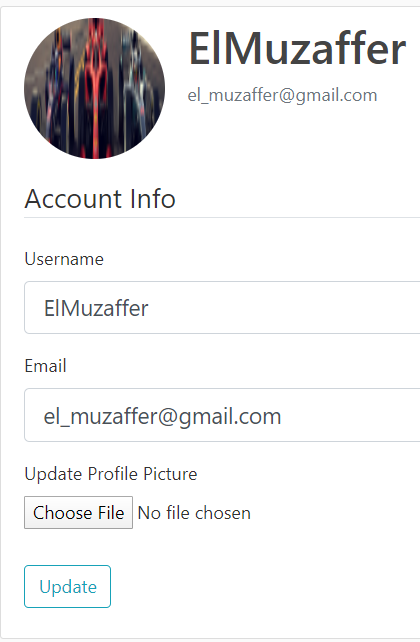


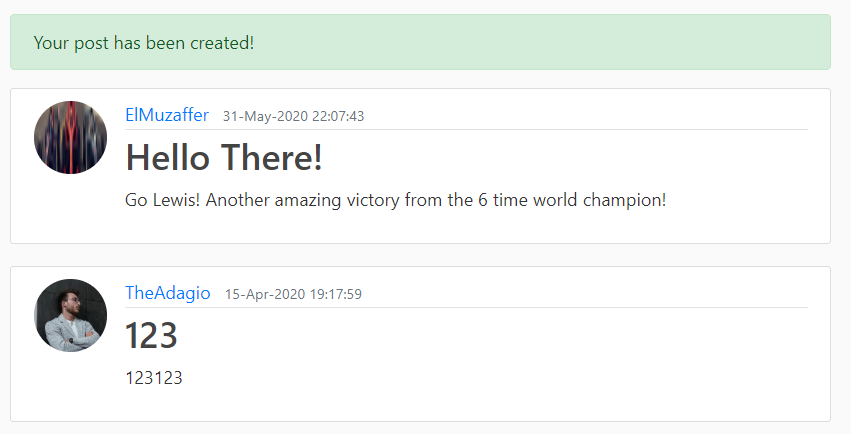
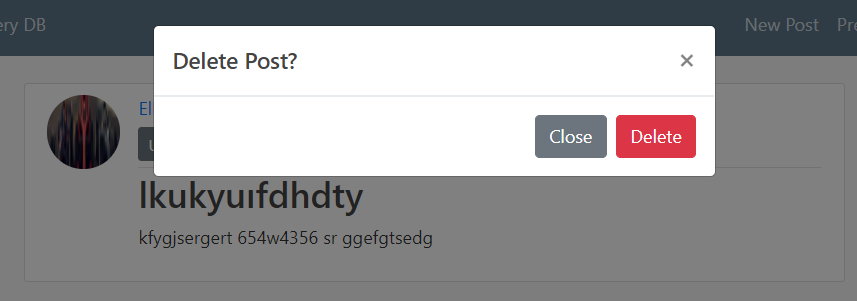
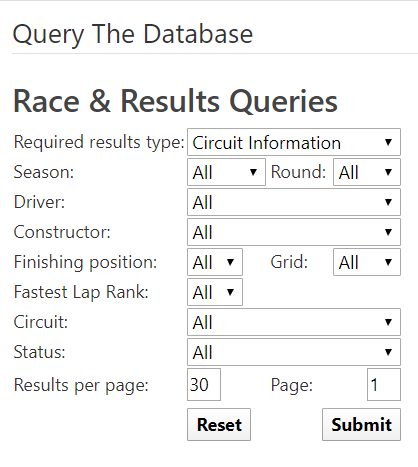
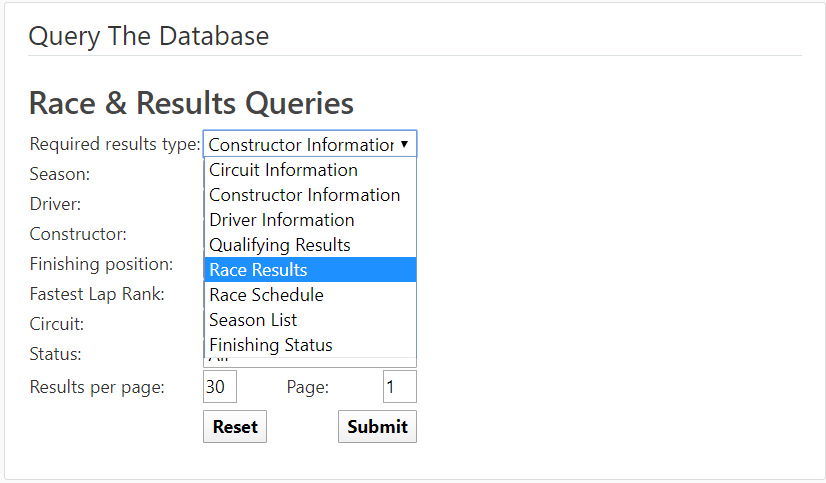
* + Users can login to the page where they are also being checked for valid credentials.  
    
  + Users can create, update and delete posts.  
     

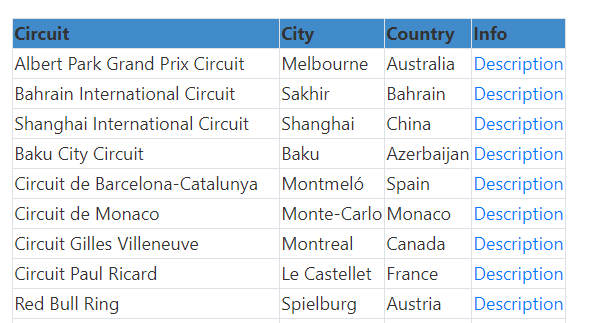
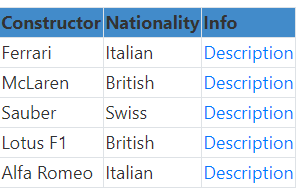
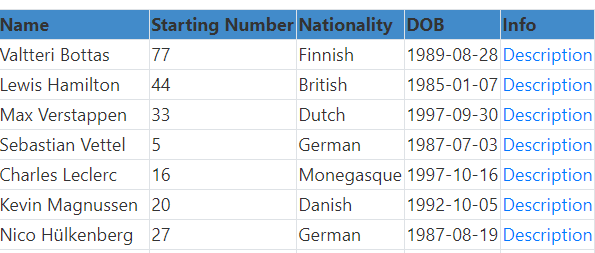
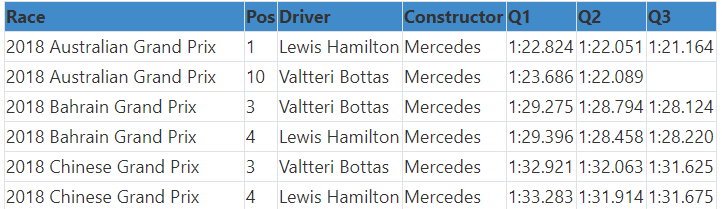
* + Users are able to view their own and other posts on the home page to stay up to date with other F1 fans.

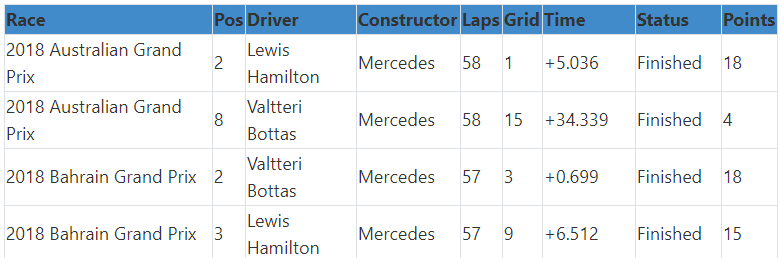
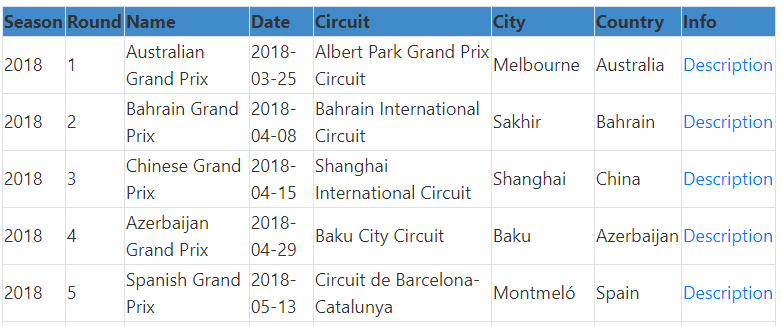


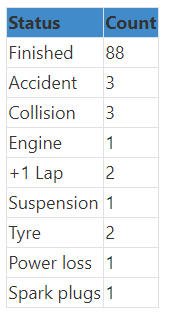
* + Users can manage their account from the account page, where they can update their profile picture, username and email. The system won’t accept usernames and emails that are already being used.



* + Users get flash messages when they update anything, successfully login, etc.  
     
  + Users can query the database for different kinds of results with 11 different applicable filters to make a more specific search:  
     

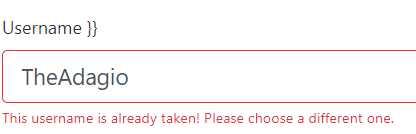
1. Circuit Info: The users can query the database for all circuits that hosted races for Formula 1 where they are provided with the circuits’ city and country and a link to the Wikipedia page for even more information about the circuits.  
   
2. Constructor Info: The users can query the database for all constructors / teams that have taken part in Formula 1. The result provides them with the nationality of the team and the link to the Wikipedia page in order to find out even more information about the teams.  
   
3. Driver Info: The users can query the database for all drivers that have ever raced in the Formula 1, where they can see if the driver had a specific number that he always stayed true to, his nationality, date of birth and a link to the Wikipedia page of the drivers’ biography.  
   
4. Qualifying Results: The users can query the database for all qualifying sessions that have taken place in Formula 1 where they can see which drivers got which position in each race and their times for Q1, Q2 and Q3 if available.  
   

1. Race Results: The users can query the database for all results for each driver of races that have taken place in Formula 1 where they can see how many laps the driver completed, what his starting position was, his finishing time and status and how many points he scored.  
   
2. Race Schedule: The users can query the database for the racing schedules for all Formula 1 seasons where they can see the round for each race of each season, its date, the circuit it took place on and the corresponding location and also a link to the Wikipedia page for even more information about each race.  
   
3. Season List: The users can query the database for the list of all seasons that fulfill all selected filters if they are looking for special occurrences where they will also be provided with the Wikipedia link to that season.  
   
4. Finishing Status: Here the users can look up how many times a specific finishing status applies to their filters, for example they can look up how many times a specific driver was disqualified or was retired from the race due to an engine failure, a puncture, oil leaks, etc.



* 1. **Non-Working Features / Bugs**
     + 1. **Non Working Features**
     + Users can’t change their password yet.
     + User can’t make predictions yet for the upcoming races.
     + Users aren’t being ranked yet according to their predictions for the races.
     + There is no ML based race prediction yet.
     + The program doesn’t auto update the database yet, partially due to no races taking place yet so there is no real practice test possible.
     + There is no pagination yet for the posts on the homepage and the results of the queries.

**4.6..2. Bugs**

* + There is a problem with the login manager so a list was created to hold the user object of the user that logged in so when the session expires and logout is clicked it gives an error.
  + The java script that resets the round value has a bug where it doesn’t display the options for the selected season when you go back from the query results page to the query page.
  + In the register and login forms there is a rendering bug when something invalid is submitted that shows “}}”   
    ****