Database Systems

BLG 317E

Term Project Report
ONLYF1S

Mustafa Can Çalışkan - caliskanmu20@itu.edu.tr
Emircan Erol - erole20@itu.edu.tr
Yusuf Şahin - sahiny20@itu.edu.tr
Yusuf Emir Sezgin - sezginy20@itu.edu.tr

Faculty of Computer and Informatics Engineering
Department of Computer Engineering

CONTENTS

1. Introduction

1.1. Introduction

This project assembles a Term Project for this course. In this project, we created a Formula 1 fandom website named "ONLYF1S" using the dataset that includes years between 1953–2023 data about Formula 1 Sports. We worked on the CRUD operations, dockerizing, rabbitMQ, generic page creation, operations, authentication, and authorization.

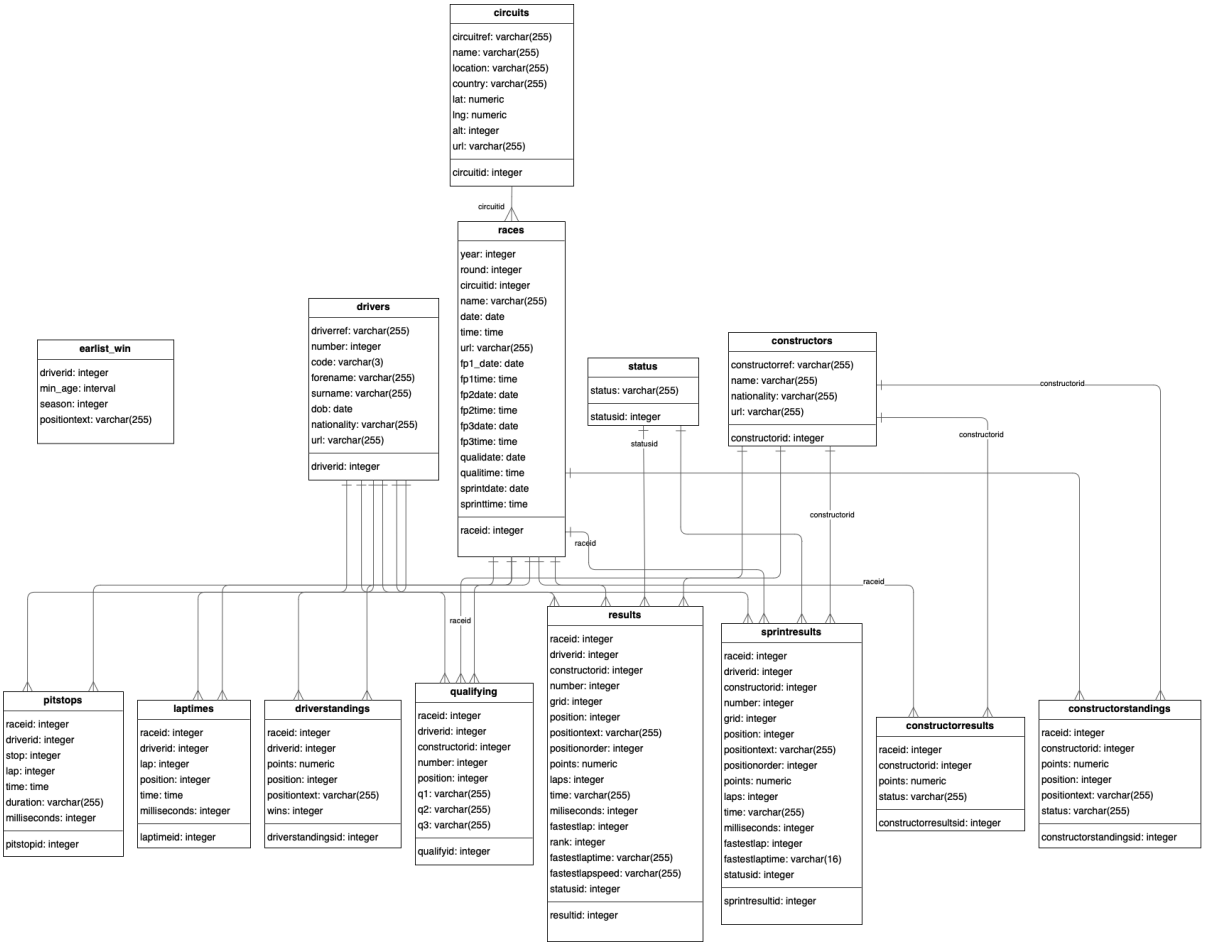
1.2. Application Overview

All users can find any drivers, constructors, circuits, etc. on their unique page. Users can search and find the specific data on every page. Users can try themselves on Quiz and F1 Blink Game. However, users that have authorization as admin can create new data, update existing data, and remove specific data as extra of the regular users' abilities to do on the website.

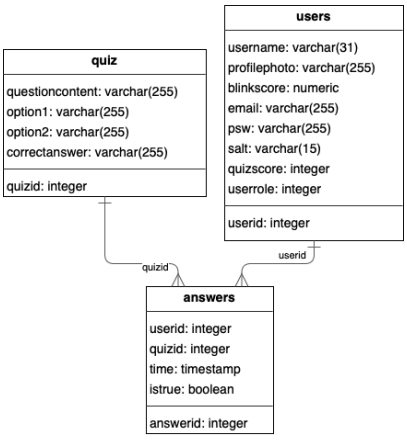
2. Database

2.1. Entity Relation Diagram (ERD)

We used Formula 1 World Championship Data set (1950 - 2023). Also, we created user, answers, and quiz tables.



(a) F1 ERD



(b) User ERD

2.2. Database Schema

Database SQL schema is as follows:

```
    driverId INT GENERATED ALWAYS AS IDENTITY (START WITH 860),
    driverRef VARCHAR(255),
    number INT,
    code VARCHAR(3),
    forename VARCHAR(255),
    surname VARCHAR(255),
    dob date,
    nationality VARCHAR(255),
    url VARCHAR(255),
    PRIMARY KEY (driverId)
);

CREATE TABLE IF NOT EXISTS constructors(
    constructorId INT GENERATED ALWAYS AS IDENTITY (START WITH 215)
       ,
    constructorRef VARCHAR(255),
    name VARCHAR(255),
    nationality VARCHAR(255),
    url VARCHAR(255),
    PRIMARY KEY(constructorId)
);

CREATE TABLE IF NOT EXISTS circuits(
    circuitId INT GENERATED ALWAYS AS IDENTITY (START WITH 80),
    circuitRef VARCHAR(255),
    name VARCHAR(255),
    location VARCHAR(255),
    country VARCHAR(255),
    lat DECIMAL,
    lng DECIMAL,
    alt INT,
    url VARCHAR(255),
    PRIMARY KEY(circuitId)
);

-- We've changed the primary key constraint, ERD was wrong in terms
    of logic.
-- Year primary key constraint has been deleted.

CREATE TABLE IF NOT EXISTS races(
```

```
    raceId INT GENERATED ALWAYS AS IDENTITY (START WITH 1121),
    year INT,
    round INT,
    circuitId INT,
    name VARCHAR(255),
    date DATE,
    time TIME,
    url VARCHAR(255),
    fp1date DATE,
    fp1time TIME,
    fp2date DATE,
    fp2time TIME,
    fp3date DATE,
    fp3time TIME,
    qualiDate DATE,
    qualiTime TIME,
    sprintDate DATE,
    sprintTime TIME,

    PRIMARY KEY(raceId),
    CONSTRAINT fkCircuitsRaces
        FOREIGN KEY(circuitId) REFERENCES circuits(circuitId)
        ON DELETE CASCADE
        ON UPDATE CASCADE

);

CREATE TABLE IF NOT EXISTS qualifying (
    qualifyId INT GENERATED ALWAYS AS IDENTITY (START WITH 10033),
    raceId INT,
    driverId INT,
    constructorId INT,
    number INT,
    position INT,
    q1 VARCHAR(255),
    q2 VARCHAR(255),
    q3 VARCHAR(255),

    PRIMARY KEY (qualifyId),

    CONSTRAINT fkQualifyingRaces
        FOREIGN KEY(raceId) REFERENCES races(raceId)
        ON DELETE CASCADE
```

```
82          ON UPDATE CASCADE,
83
84      CONSTRAINT fkQualifyingDrivers
85          FOREIGN KEY(driverId) REFERENCES drivers(driverId)
86          ON DELETE CASCADE
87          ON UPDATE CASCADE,
88
89      CONSTRAINT fkQualifyingConstructors
90          FOREIGN KEY (constructorId)     REFERENCES constructors (
                constructorId)
91          ON DELETE CASCADE
92          ON UPDATE CASCADE
93
94  );
95
96  CREATE TABLE IF NOT EXISTS status (
97      statusId INT GENERATED ALWAYS AS IDENTITY (START WITH 142),
98      status VARCHAR(255),
99      PRIMARY KEY(statusId)
100 );
101
102 CREATE TABLE IF NOT EXISTS sprintResults(
103     sprintResultId INT GENERATED ALWAYS AS IDENTITY (START WITH
            241),
104     raceId INT,
105     driverId INT,
106     constructorId INT,
107     number INT,
108     grid INT,
109     position INT,
110     positionText VARCHAR(255),
111     positionOrder INT,
112     points DECIMAL,
113     laps INT,
114     time VARCHAR(255),
115     milliseconds INT,
116     fastestLap INT,
117     fastestLapTime VARCHAR(16),
118     statusID INT,
119
120     PRIMARY KEY(sprintResultId),
121     CONSTRAINT fkRacesSprintResults
122         FOREIGN KEY(raceID)     REFERENCES races(raceId)
```

```
123         ON DELETE CASCADE
124         ON UPDATE CASCADE,
125
126     CONSTRAINT fkDriversSprintResults
127         FOREIGN KEY(driverId) REFERENCES drivers(driverId)
128         ON DELETE CASCADE
129         ON UPDATE CASCADE,
130
131     CONSTRAINT fkConstructorsSprintResults
132         FOREIGN KEY(constructorId) REFERENCES constructors(
              constructorId)
133         ON DELETE CASCADE
134         ON UPDATE CASCADE,
135
136     CONSTRAINT fkStatusSprintResults
137         FOREIGN KEY(statusId) REFERENCES status(statusId)
138         ON DELETE CASCADE
139         ON UPDATE CASCADE
140 );
141
142 CREATE TABLE IF NOT EXISTS results(
143     resultId INT GENERATED ALWAYS AS IDENTITY (START WITH 26246),
144     raceId INT,
145     driverId INT,
146     constructorId INT,
147     number INT,
148     grid INT,
149     position INT,
150     positionText VARCHAR(255),
151     positionOrder INT,
152     points DECIMAL,
153     laps INT,
154     time VARCHAR(255),
155     miliseconds INT,
156     fastestLap INT,
157     rank INT,
158     fastestLapTime VARCHAR(255),
159     fastestLapSpeed VARCHAR(255),
160     statusId INT,
161
162     PRIMARY KEY(resultId),
163
164     CONSTRAINT fkRacesResults
```

```
        FOREIGN KEY(raceId) REFERENCES races(raceId)
        ON DELETE CASCADE
        ON UPDATE CASCADE,

    CONSTRAINT fkDriverResults
        FOREIGN KEY(driverId) REFERENCES drivers(driverId)
        ON DELETE CASCADE
        ON UPDATE CASCADE,

    CONSTRAINT fkConstructorsResults
        FOREIGN KEY(constructorId) REFERENCES constructors(
            constructorId)
        ON DELETE CASCADE
        ON UPDATE CASCADE,

    CONSTRAINT FkStatusResults
        FOREIGN KEY(statusId) REFERENCES status(statusId)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS pitStops (
    pitstopid INT GENERATED ALWAYS AS IDENTITY (START WITH 10479),
    raceId INT,
    driverId INT,
    stop INT,
    lap INT,
    time TIME,
    duration VARCHAR(255),
    milliseconds INT,

    PRIMARY KEY(pitstopid),

    CONSTRAINT fkRacesPitStops
        FOREIGN KEY(raceId) REFERENCES races(raceId)
        ON DELETE CASCADE
        ON UPDATE CASCADE,

    CONSTRAINT fkDriversPitStops
        FOREIGN KEY(driverId) REFERENCES drivers(driverId)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

```
207
208 CREATE TABLE IF NOT EXISTS lapTimes(
209     laptimeid INT GENERATED ALWAYS AS IDENTITY (START WITH 560408),
210     raceId INT,
211     driverId INT,
212     lap INT,
213     position INT,
214     time TIME,
215     milliseconds INT,
216
217     PRIMARY KEY(laptimeid),
218
219     CONSTRAINT fkRacesLapTimes
220         FOREIGN KEY(raceId) REFERENCES races(raceId)
221         ON DELETE CASCADE
222         ON UPDATE CASCADE,
223
224     CONSTRAINT fkDriversLapTimes
225         FOREIGN KEY(driverId) REFERENCES drivers(driverId)
226         ON DELETE CASCADE
227         ON UPDATE CASCADE
228 );
229
230 CREATE TABLE IF NOT EXISTS driverStandings (
231     driverStandingsId INT GENERATED ALWAYS AS IDENTITY (START WITH
        72452),
232     raceId INT,
233     driverId INT,
234     points DECIMAL,
235     position INT,
236     positionText VARCHAR(255),
237     wins INT,
238
239     PRIMARY KEY (driverStandingsId),
240
241     CONSTRAINT fkRacesDriverStandings
242         FOREIGN KEY(raceId)     REFERENCES races(raceId)
243         ON DELETE CASCADE
244         ON UPDATE CASCADE,
245
246     CONSTRAINT fkDriversDriverStandings
247         FOREIGN KEY(driverId) REFERENCES drivers(driverId)
248         ON DELETE CASCADE
```

```
249         ON UPDATE CASCADE
250 );
251
252 CREATE TABLE IF NOT EXISTS constructorStandings(
253     constructorStandingsId INT GENERATED ALWAYS AS IDENTITY (START
        WITH 28693),
254     raceId INT,
255     constructorId INT,
256     points DECIMAL,
257     position INT,
258     positionText VARCHAR(255),
259     status VARCHAR(255),
260
261     PRIMARY KEY(constructorStandingsId),
262
263     CONSTRAINT fkRacesConstructorStandings
264         FOREIGN KEY(raceId)     REFERENCES races(raceId)
265         ON DELETE CASCADE
266         ON UPDATE CASCADE,
267
268     CONSTRAINT fkConstructorsConstructorStandings
269         FOREIGN KEY(constructorId) REFERENCES constructors(
            constructorId)
270         ON DELETE CASCADE
271         ON UPDATE CASCADE
272 );
273
274 CREATE TABLE IF NOT EXISTS constructorResults(
275     constructorResultsId INT GENERATED ALWAYS AS IDENTITY (START
        WITH 16870),
276     raceId INT,
277     constructorId INT,
278     points DECIMAL,
279     status VARCHAR(255),
280
281     PRIMARY KEY(constructorResultsId),
282
283     CONSTRAINT fkRacesConstructorResults
284         FOREIGN KEY(raceId) REFERENCES races(raceId)
285         ON DELETE CASCADE
286         ON UPDATE CASCADE,
287
288     CONSTRAINT fkConstructorsConstructorResults
```

```
289         FOREIGN KEY(constructorId) REFERENCES constructors(
                constructorId)
290         ON DELETE CASCADE
291         ON UPDATE CASCADE
292 );
293
294
295 CREATE TABLE IF NOT EXISTS quiz (
296     quizId INT GENERATED ALWAYS AS IDENTITY (START WITH 2),
297     questionContent VARCHAR(255),
298     option1 VARCHAR(255),
299     option2 VARCHAR(255),
300     correctAnswer VARCHAR(255),
301
302     PRIMARY KEY (quizId)
303 );
304
305 CREATE TABLE IF NOT EXISTS users(
306     userId INT GENERATED ALWAYS AS IDENTITY,
307     username VARCHAR(31) UNIQUE,
308     profilePhoto VARCHAR(255) UNIQUE,
309     blinkScore DECIMAL,
310     email VARCHAR(255) UNIQUE,
311     psw VARCHAR(255),
312     salt VARCHAR(15),
313     quizScore INT,
314     userRole INT,
315
316     PRIMARY KEY(userId),
317     CONSTRAINT usernameEmailNotnull CHECK (
318         NOT (
319             ( username IS NULL  OR  username = '' )
320             AND
321             ( email IS NULL  OR  email = '' )
322         )
323     ),
324     CONSTRAINT usersIntegrity CHECK (
325         quizScore > 0 AND
326         blinkScore > 0 AND
327         userRole = 0 OR userRole = 1
328     )
329 );
330
```

```sql
CREATE TABLE IF NOT EXISTS answers(
    answerId INT GENERATED ALWAYS AS IDENTITY,
    userId INT,
    quizId INT,
    time TIMESTAMP,
    isTrue BOOLEAN,

    PRIMARY KEY (answerId),
    UNIQUE (userId, quizId, time),

    CONSTRAINT fkUserAnswers
        FOREIGN KEY(userId) REFERENCES users(userId)
        ON DELETE CASCADE
        ON UPDATE CASCADE,

    CONSTRAINT fkQuizAnswers
        FOREIGN KEY(quizId) REFERENCES quiz(quizId)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS lkp_tables(
    tableId INT GENERATED ALWAYS AS IDENTITY,
    tableName varchar(30),

    PRIMARY KEY (tableId)
);

INSERT INTO lkp_tables (tableName)
SELECT table_name
FROM information_schema.tables
WHERE table_schema = 'public' AND table_name != 'lkp_tables';

CREATE VIEW EARLIST_WIN AS
SELECT re.driverid,
    MIN(AGE(r.date, d.dob)) as min_age,
    MIN(r.year) as season,
    re.positiontext
    FROM races r JOIN results re ON re.raceid = r.raceid
    JOIN drivers d ON d.driverid = re.driverid
    WHERE re.positiontext = '1'
    GROUP BY re.driverid, re.positiontext;

```

```
374 CREATE VIEW PER_WINS AS
375 SELECT d.forename, d.surname,
376 (SELECT CONCAT(MIN(r.year), ' - ', MAX(r.year))
377 FROM races r JOIN results re ON re.raceid = r.raceid
378 WHERE re.positiontext = '1' AND re.driverid = d.driverid) AS
       seasons,
379 COUNT (re.driverid) AS ENTRY,
380 COUNT(CASE WHEN re.positiontext = '1' THEN 1 ELSE NULL END) AS
       total_wins,
381 ROUND((COUNT(CASE WHEN re.positiontext = '1' THEN 1 ELSE NULL END)
       ::NUMERIC / COUNT (re.driverid))::NUMERIC * 100, 2) AS win_rate
382 FROM drivers d JOIN results re ON d.driverid = re.driverid
383 GROUP BY d.forename, d.surname, d.driverid
384 ORDER BY win_rate
385 DESC;
386
387 CREATE OR REPLACE FUNCTION get_race_results(slimNumber INTEGER)
388 RETURNS TABLE (
389     forename VARCHAR(255),
390     surname VARCHAR(255),
391     age INTERVAL,
392     name VARCHAR(255),
393     season INTEGER,
394     round INTEGER,
395     result VARCHAR(255)
396 ) AS $$
397 BEGIN
398     RETURN QUERY
399     SELECT d.forename,
400     d.surname,
401     a.max_age as age,
402     r.name,
403     a.season,
404     r.round,
405     re.positiontext as result
406     FROM races r JOIN results re ON re.raceid = r.raceid
407     JOIN drivers d ON d.driverid = re.driverid
408     JOIN status s ON re.statusid = s.statusid
409     JOIN (SELECT re.driverid,
410         MAX(AGE(r.date, d.dob)) as max_age,
411         MIN(r.year) as season,
412         s.statusid
413     FROM races r JOIN results re ON re.raceid = r.raceid
```

```
414     JOIN drivers d ON d.driverid = re.driverid
415     JOIN status s ON re.statusid = s.statusid WHERE s.statusid = 1
416     GROUP BY re.driverid, s.statusid
417     ) a ON d.driverid = a.driverid
418     AND AGE(r.date, d.dob) = a.max_age AND s.statusid = a.statusid
419     ORDER BY age DESC
420     LIMIT slimNumber;
421 END; $$ LANGUAGE plpgsql;
422
423 CREATE FUNCTION user_insertion_logger() RETURNS TRIGGER AS $$
424 BEGIN
425    RAISE LOG 'A new row with id % and username % has been inserted'
          , NEW.userId, NEW.username;
426    RETURN NEW;
427 END;
428 $$ LANGUAGE plpgsql;
429
430 CREATE TRIGGER user_logger
431 AFTER INSERT ON users
432 FOR EACH ROW
433 EXECUTE PROCEDURE user_insertion_logger();
```

3. Implementation

3.1. Database Management System and General Tools

In our project, we utilized Flask [4], the web development framework of Python, alongside PostgreSQL [5] as the database management system. We supported the site's appearance using HTML, CSS, and JavaScript codes.

3.2. Dockerizing

To ensure consistent environmental setup and overcome local constraints, we leverage a Docker [3] application for our project. The F1 project consists of three main components: the first is the PostgreSQL server for database management, the second is the application image for executing Flask, and the third is the RabbitMQ [2] server, a messaging system that facilitates communication between different components of the project. The PostgreSQL server is connected to the application container, ensuring seamless database management. The project's front end is hosted through a proxy at the application layer, providing a comprehensive and interconnected environment. Although using Docker has lots of advantages, typing the same commands again and again can be boring. Therefore, we have implemented a Makefile to make it easier to handle Docker-based commands. Makefile commands are as follows:

```
1  up:
2      docker-compose up
3      # This method builds all the images
4
5  down:
6      docker-compose down
7      # This method stops all the containers
8
9  down-volumes:
10     docker-compose down -v
11     # This method removes the created volumes besides stopping all
           the containers
12
13 rmi:
14     docker image rm itudb2319_app itudb2319-app postgres rabbitmq
           :3-management
15     # This method removes all the images
16
17 rm-app:
18     docker image rm itudb2319-app itudb2319_app
19     # This method removes the app image
```

```
20
21 rm-db:
22     docker image rm postgres
23     # This method removes the Postgresql image
24
25 rm-rabbitmq:
26     docker image rm rabbitmq:3-management
27     # This method removes the RabbitMQ image
```

An example command to stop the containers, keep the volumes and remove the RabbitMQ image

```
make down rm-rabbitmq
```

Warning: The make commands might give some errors due to permission based your group settings. In this case, you should run this command

```
sudo make down rm-rabbitmq
```

3.3. BULK - CUD Operations

As expected, the Admin can perform Create, Update or Delete operations. However, if the Admin wants to affect so many rows, it might be undesirable. Therefore, we have implemented Bulk CUD. This method generates queries based on the uploaded CSV file's name and the file itself. There is a convention for naming the CSV file:

- Every table has an ID, which is kept in the lkp_tables table
- Operation abbreviation: UP, IN and DEL

Example file name: 4IN.csv

- 4 is the ID of driver table
- IN stands for the INSERT operation
- Each line in the 4IN.csv file represents a record
- {column_name} : {column_value} represents a key-value relation
- **-** is the delimiter for the {column_name} : {column_value} pairs
- **,** is the delimiter for the primary keys

Using this method, one-table-related operations become much easier. This is an example content of the file 4UP.csv:

6,forename:'Example'-surname:'Driver'-driverref:'example'
3,forename:'Existing'-surname:'Driver'-driverref:'BLG'

- 6 is the primary key which will be used in WHERE driverid = %s condition
- Every key and value are separated by **:**
- Every key-value pair is separated by **-**

3.4. Before Page Implementations

3.4.1. Generalization

Rewriting the implementation for CRUD operations in the front end was highly time-consuming and confusing. A generic blueprint and HTML is created for this purpose. After generic templates, the front side of the page could be created with just one line of code.

3.4.2. SQL Injection

Opting for parametrized execution of queries is the preferred approach when dealing with dynamic queries. By utilizing parameters, psycopg (PostgreSQL adapter for Python) gains the ability to discern and mitigate potential SQL attacks. This method enhances security by ensuring that user inputs are treated as parameters rather than embedded directly into the SQL query, thus minimizing the risk of malicious exploits. It's a proactive measure that fortifies the system against potential vulnerabilities associated with dynamic queries.

3.4.3. Authentication and Authorization

Access control in the system is managed through distinct admin and user roles. Specifically, administrative privileges are granted solely to administrators, enabling them to create, update, and delete records from tables for enhanced security. On the other hand, regular logged-in users are granted access to the quiz game functionality.

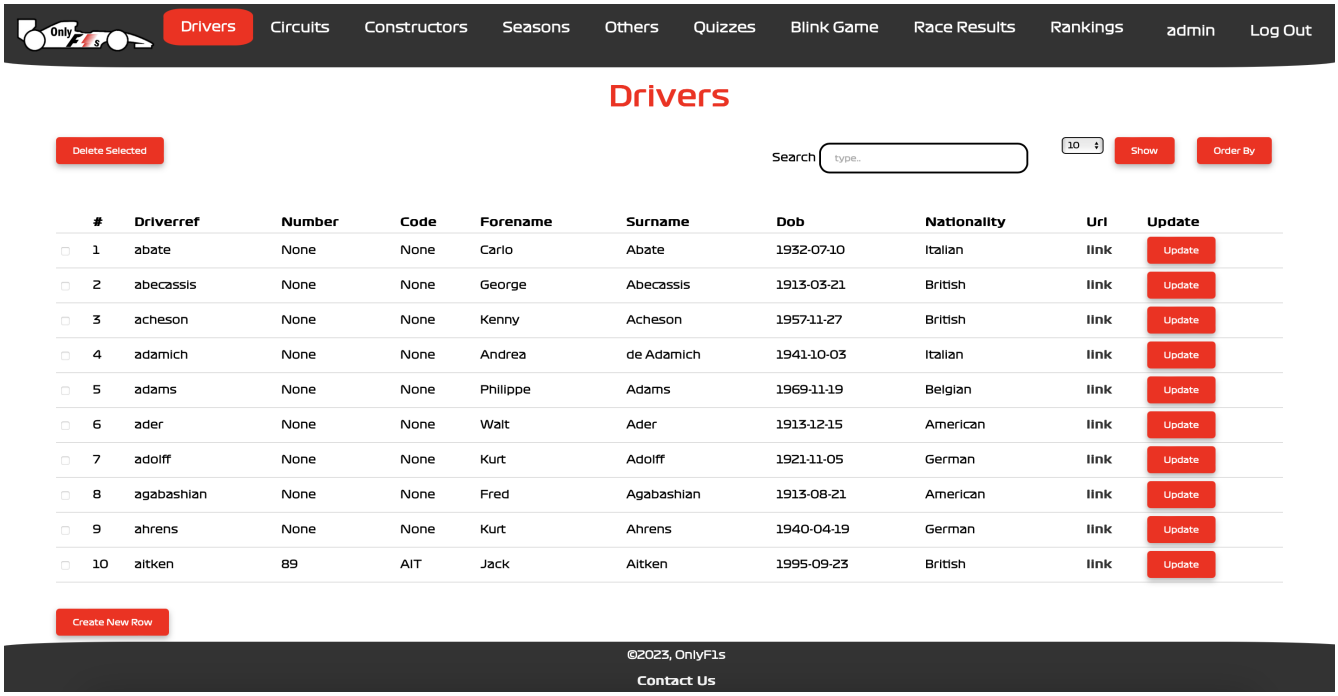
User accounts are created through the signup section, requiring a username and password. Subsequently, authorization occurs via the login screen. After signing in, users can update their usernames and email addresses by navigating to the user panel and clicking on their usernames. To conclude a session, users can log out using the designated button on the user panel. This hierarchical approach to authorization and user account management ensures a secure and streamlined experience within the system.

3.5. Page Implementations

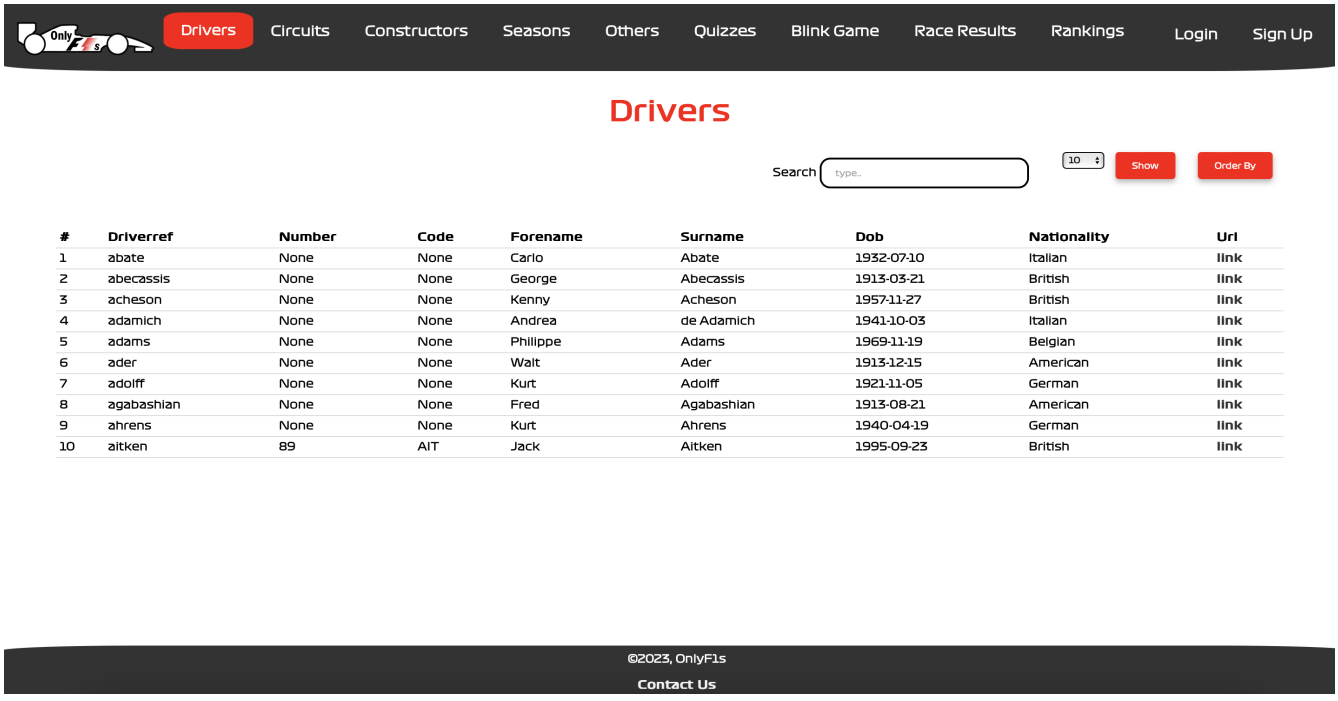
Specific page details are given in this section.

3.5.1. Drivers

On this page, users can see drivers who raced or currently racing in Formula 1. Users can interact with a driver to see its details.



(a) Drivers Page

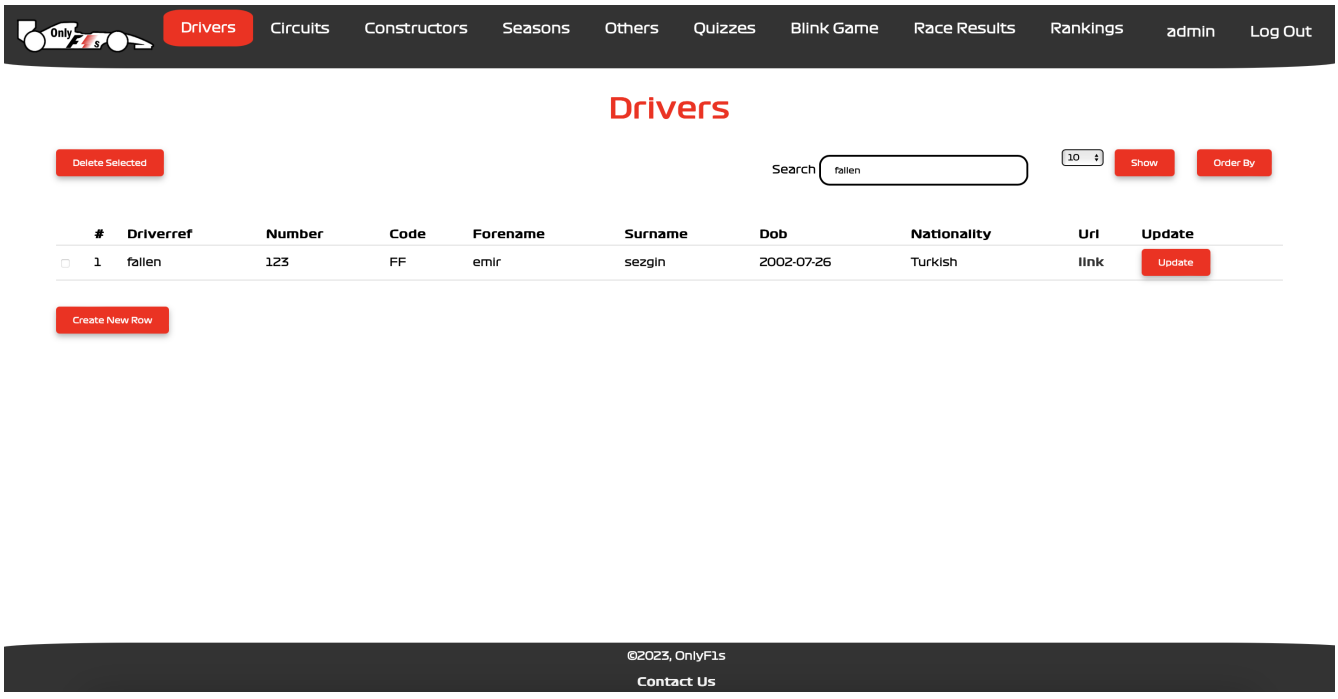


(a) Generic User View of The Drivers Page

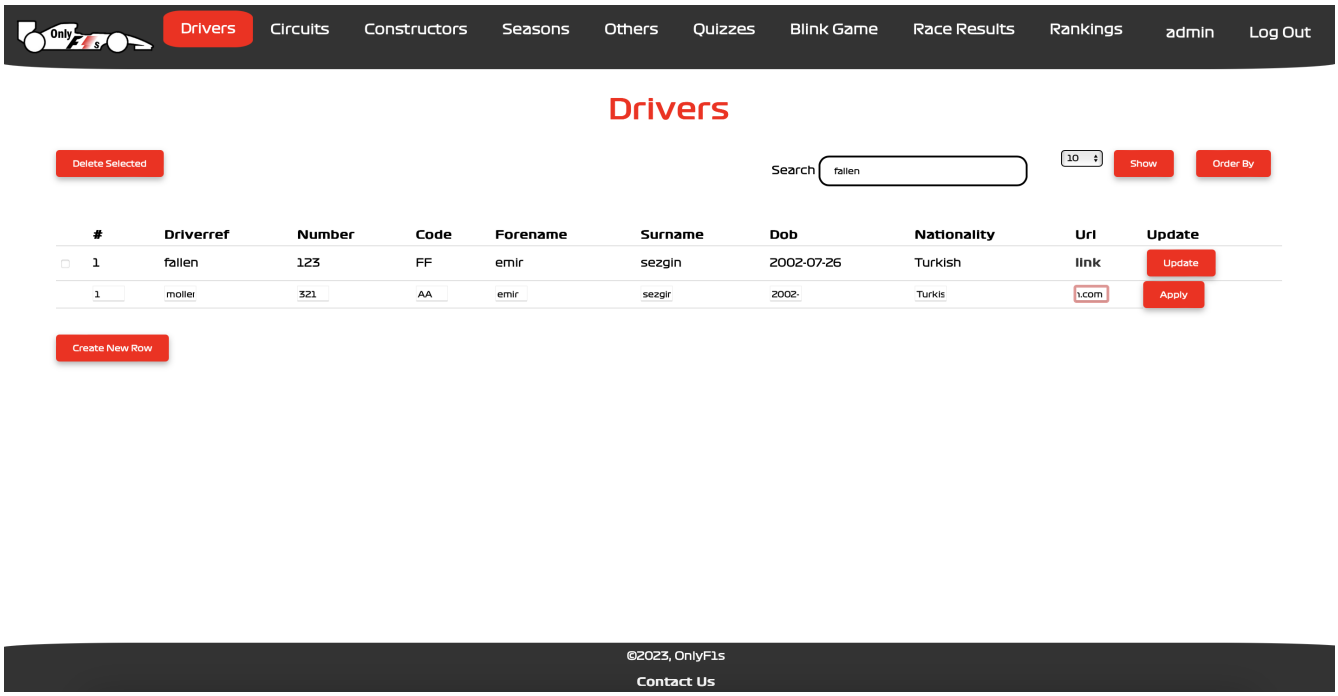
3.5.1.1. CRUD on Drivers Page

A User can perform CRUD operations if it is authorized as an admin.

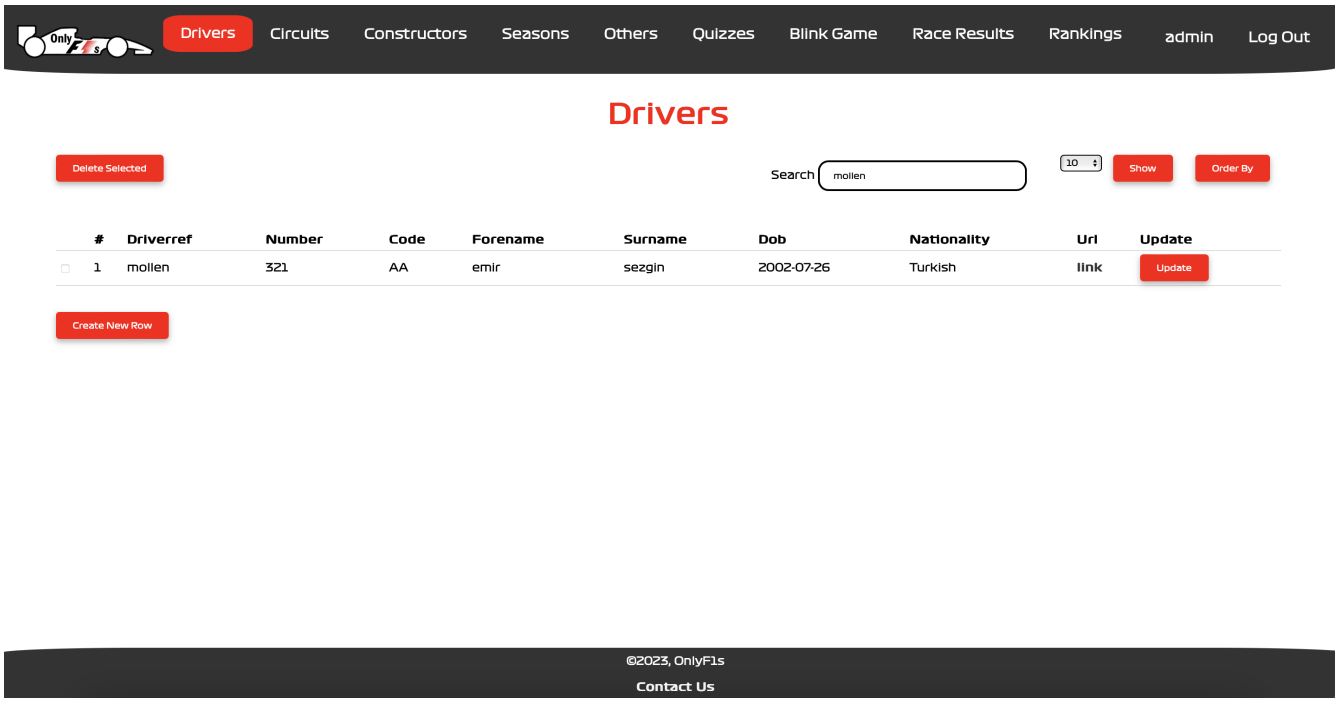
(a) Creating a Driver



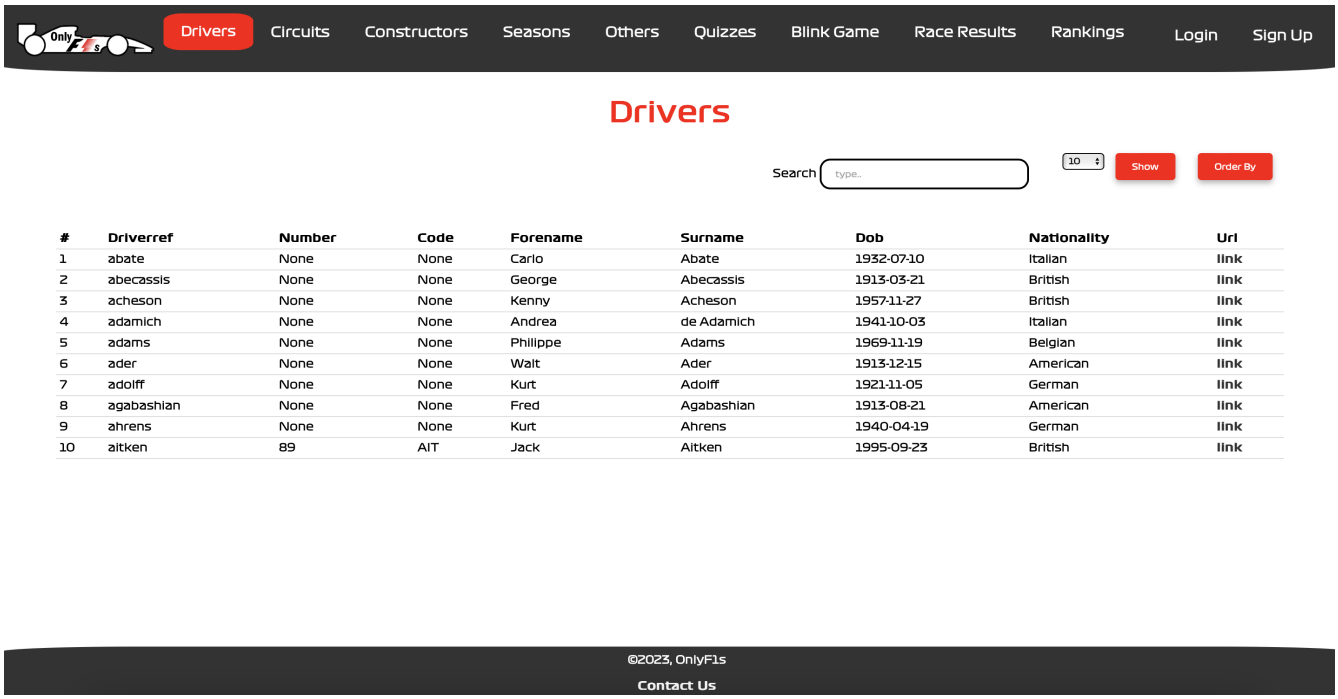
(a) Searching for a Driver



(a) Update a Driver



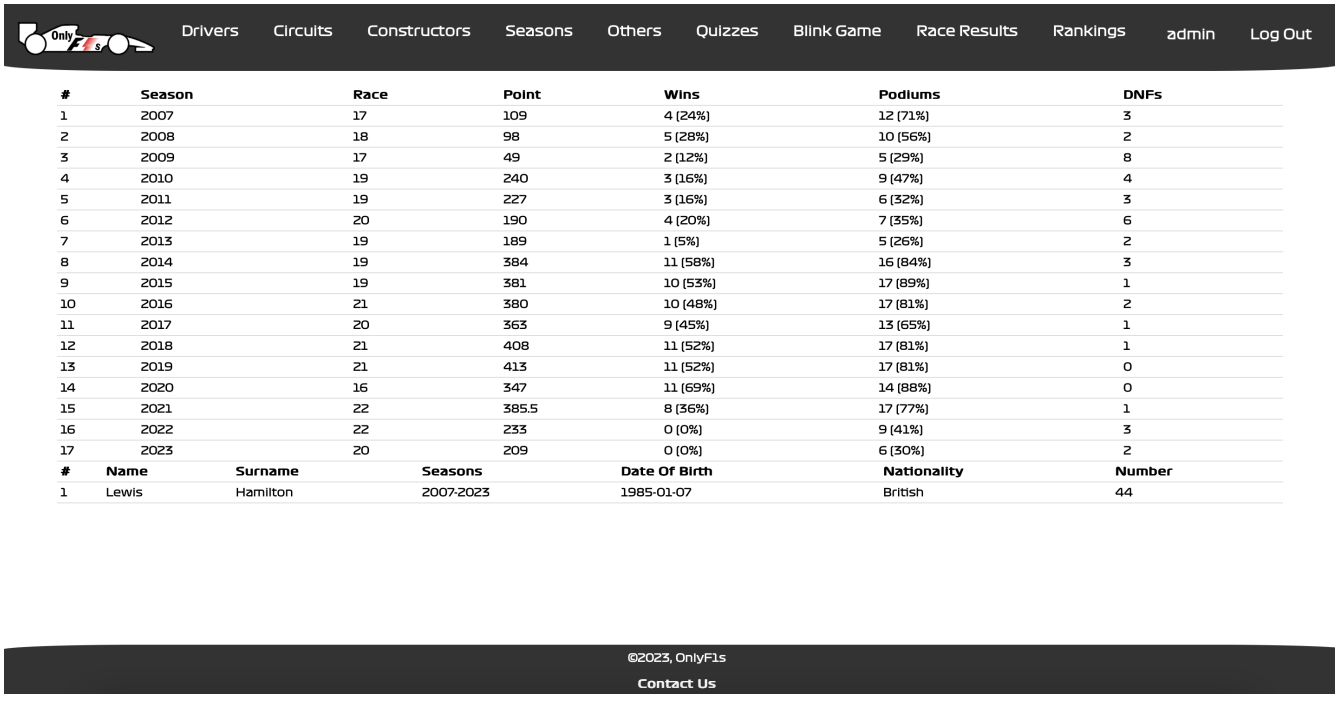
(a) Updated Data



(a) Generic User View of The Drivers Page

3.5.1.2. Driver Details

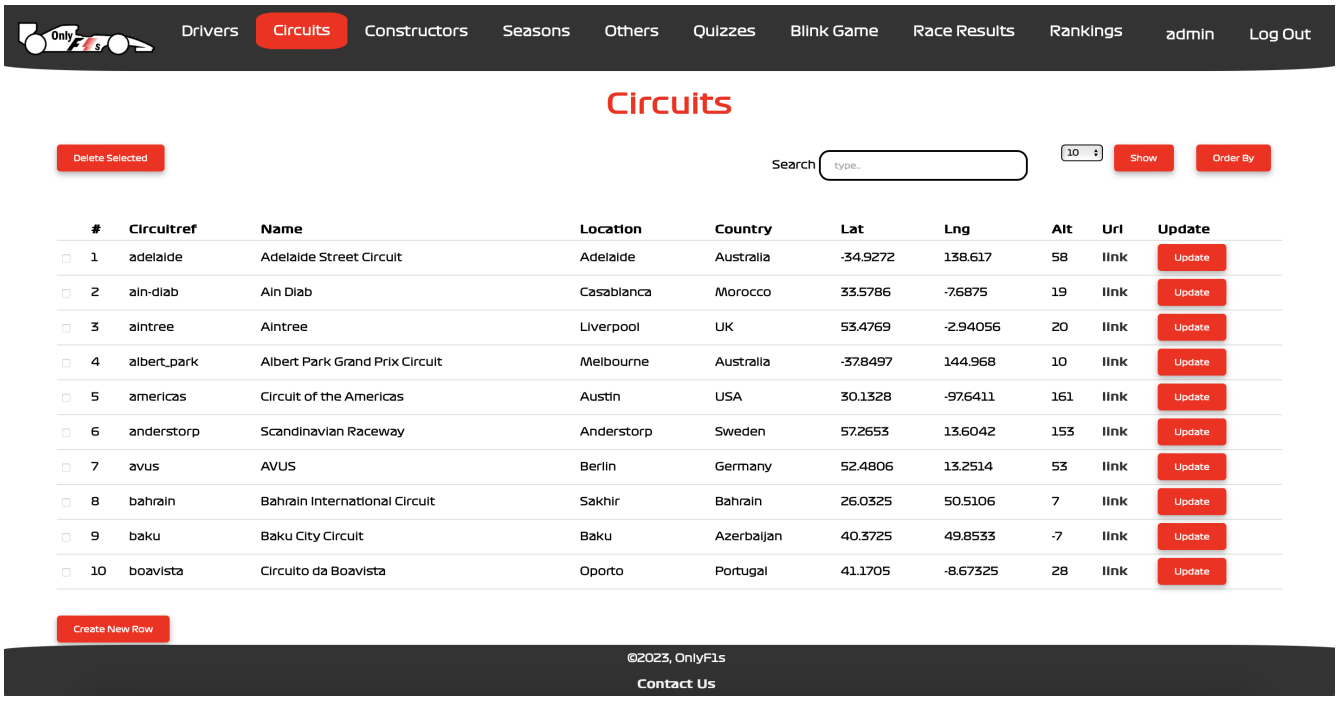
A driver's specific career info and personal info are shown on this page. Career info shows results about the season of raced years which are podium, win, DNF counts, and rates.



(a) A Specific Driver Details Page

3.5.2. Circuits

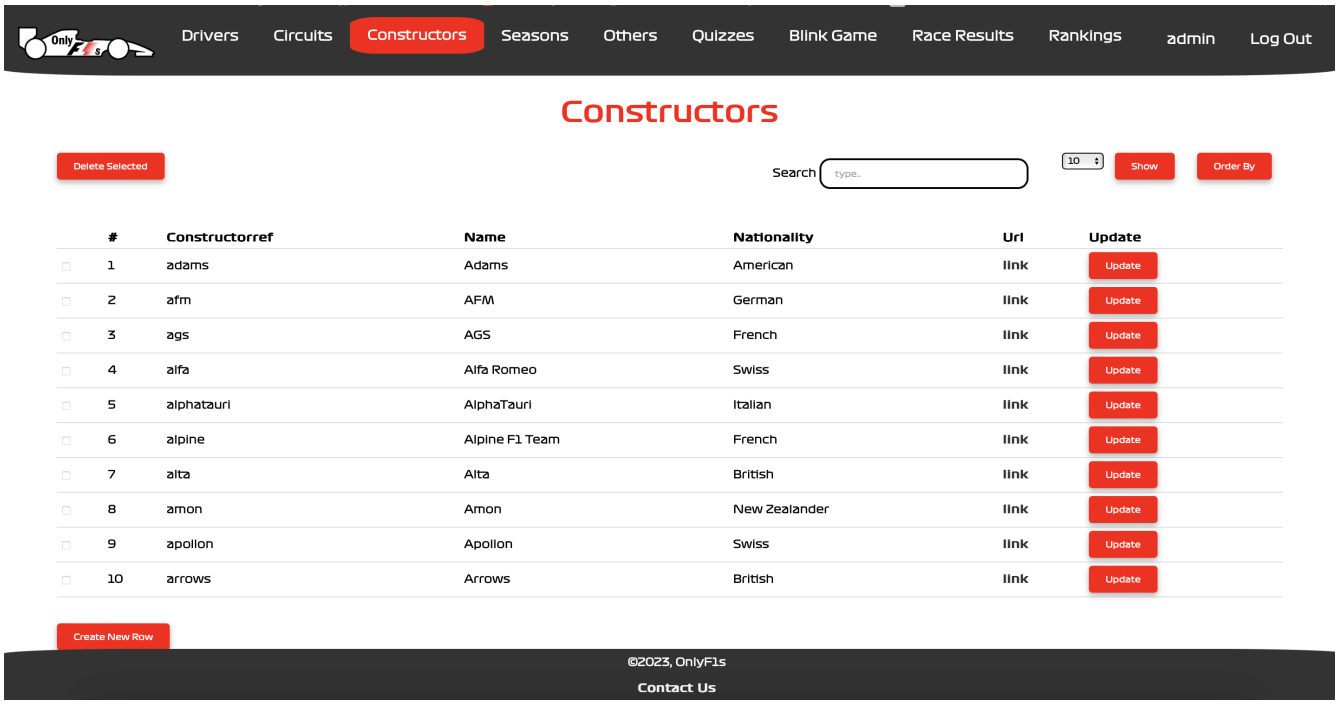
On this page, users can see circuits around the world that are used and currently used.



(a) Circuits Page

3.5.3. Constructors

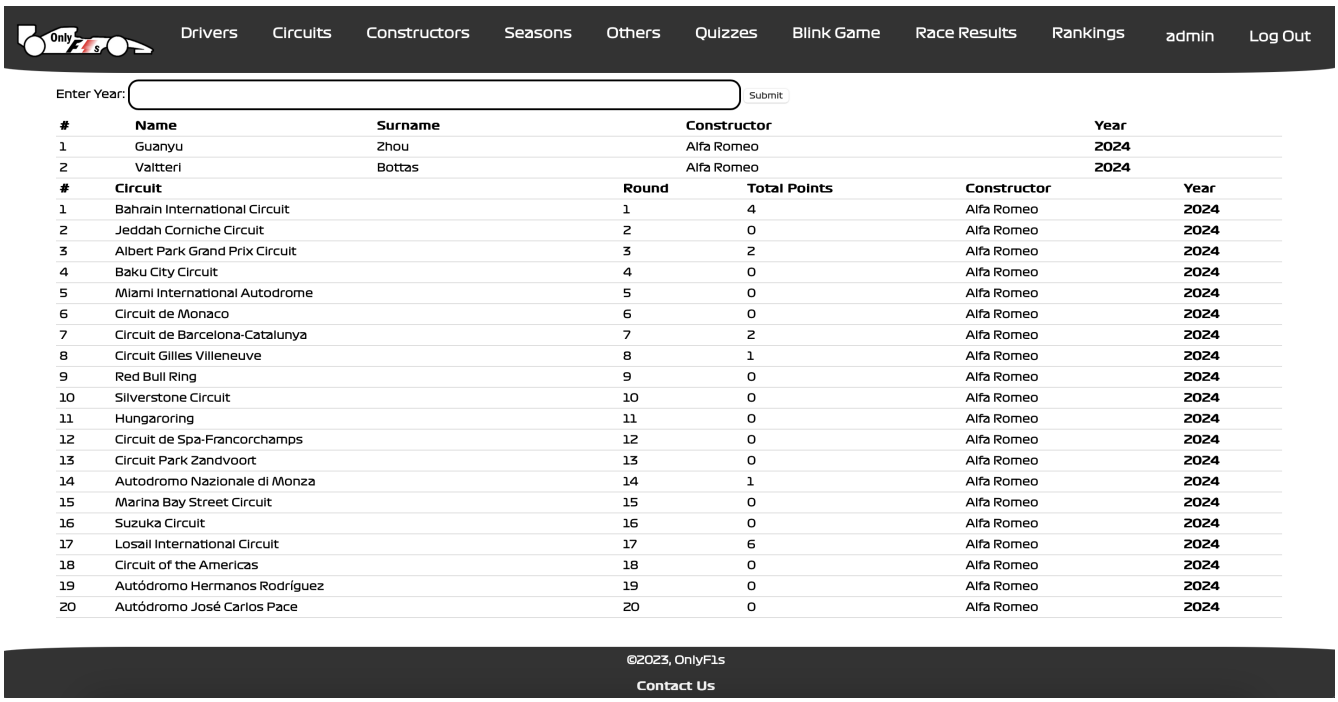
On this page, users can see constructors who raced or currently racing in Formula 1. Users can interact with a constructor to see its details.



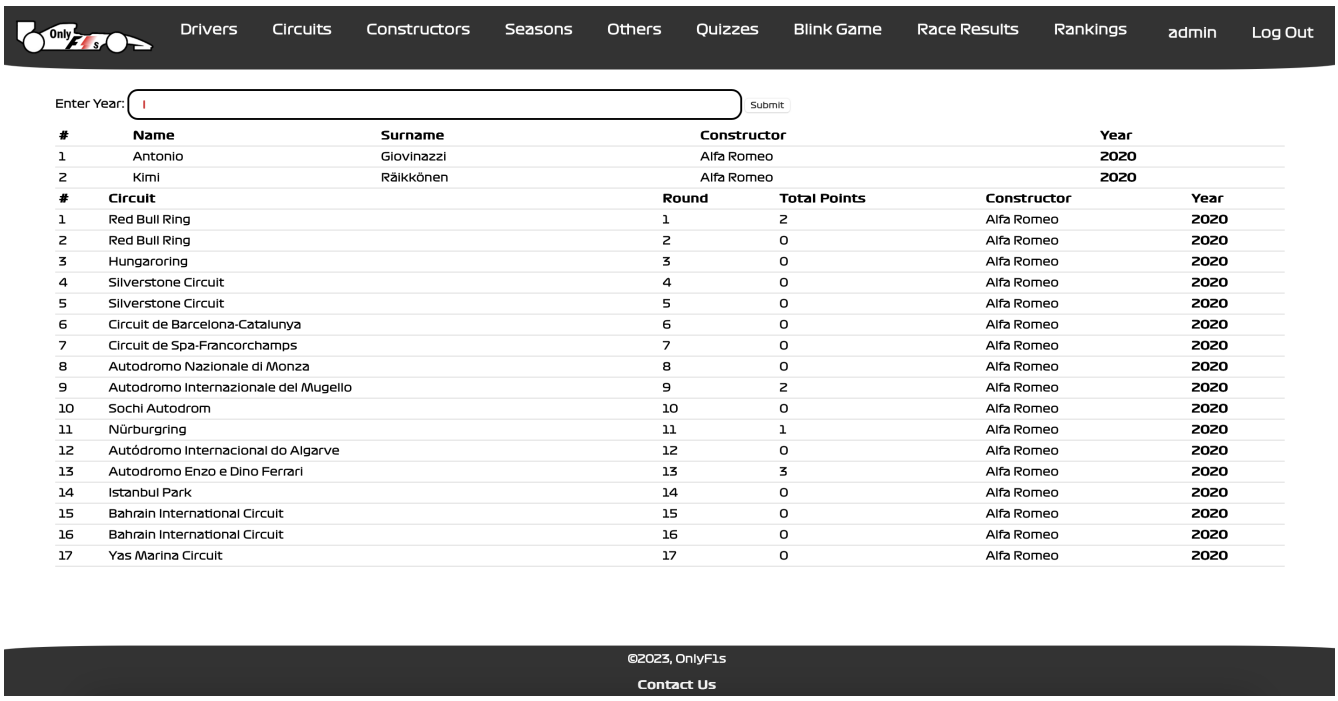
(a) Constructors Page

3.5.3.1. Constructor Details

Seasonal driver and circuit details are shown on this page. Users can search a season year and data about the drivers and circuits that have a relation with the specific constructor.



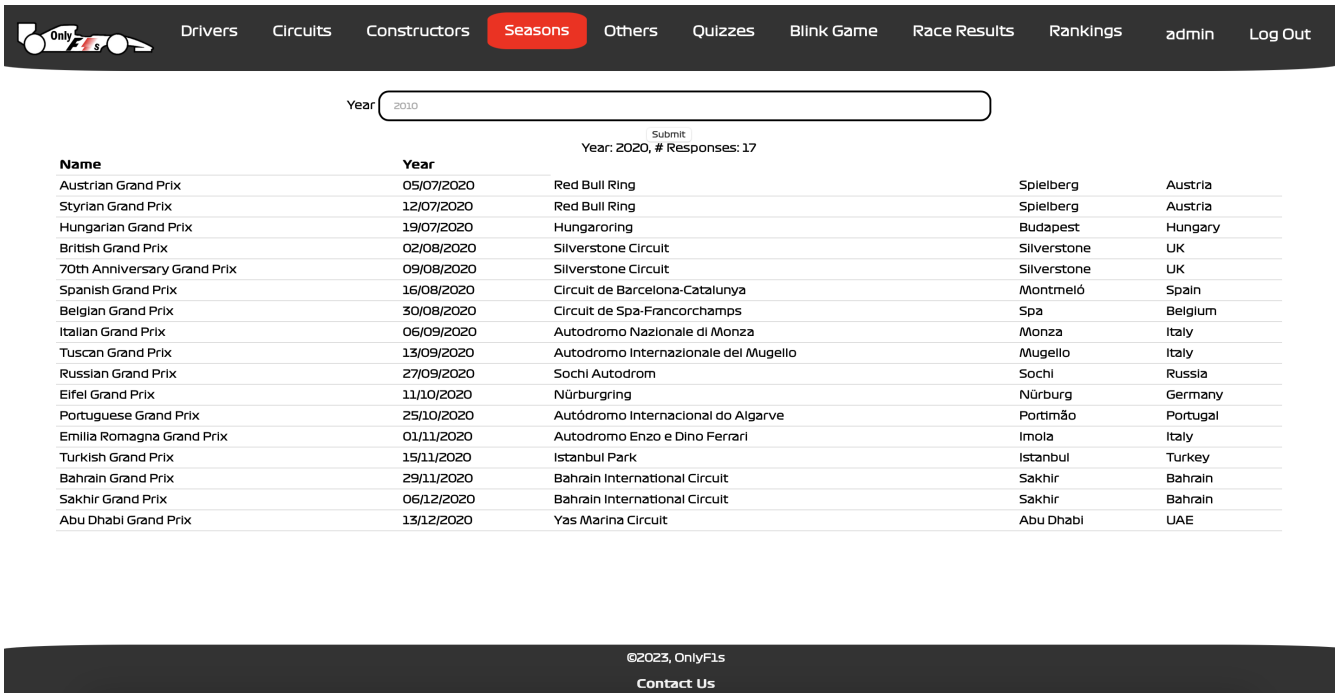
(a) A Specific Constructor Details Page



(a) Searched Info on the Constructor Details Page

3.5.4. Seasons

On this page, users can find the corresponding races on the searched season year.

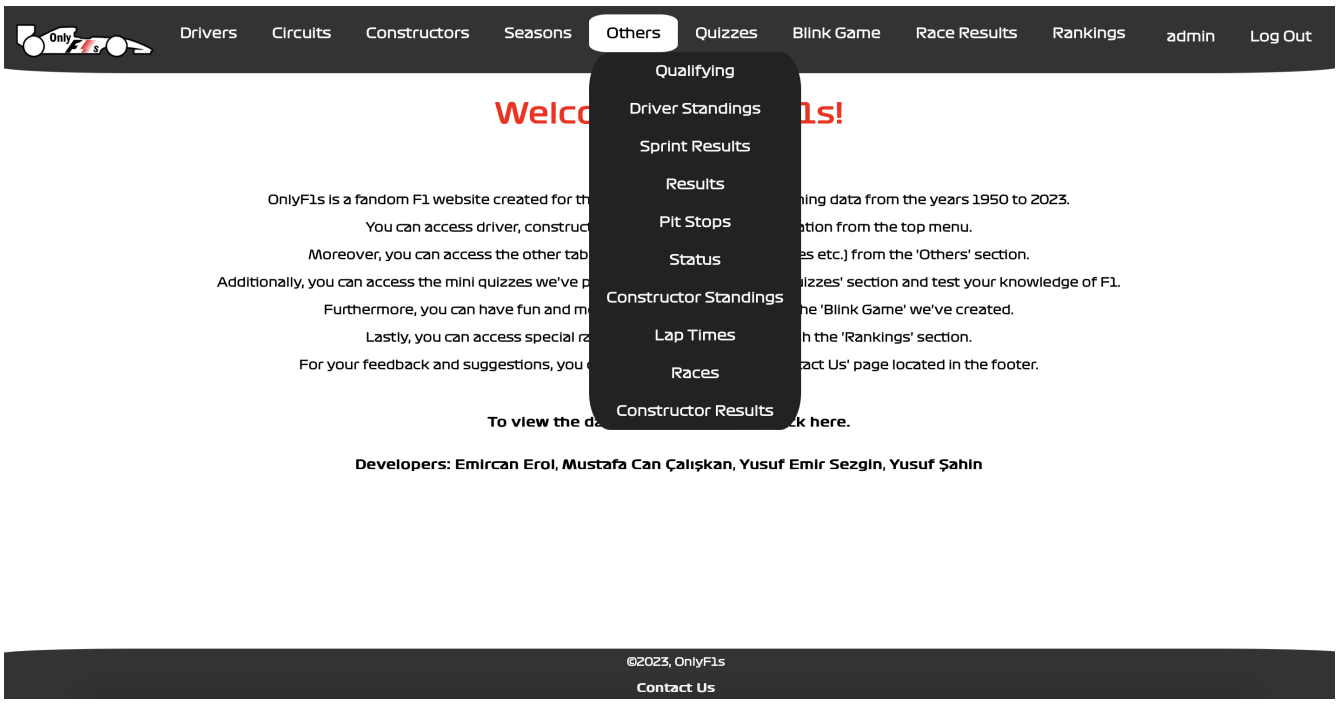


Name	Year			
Austrian Grand Prix	05/07/2020	Red Bull Ring	Spielberg	Austria
Styrian Grand Prix	12/07/2020	Red Bull Ring	Spielberg	Austria
Hungarian Grand Prix	19/07/2020	Hungaroring	Budapest	Hungary
British Grand Prix	02/08/2020	Silverstone Circuit	Silverstone	UK
70th Anniversary Grand Prix	09/08/2020	Silverstone Circuit	Silverstone	UK
Spanish Grand Prix	16/08/2020	Circuit de Barcelona-Catalunya	Montmeló	Spain
Belgian Grand Prix	30/08/2020	Circuit de Spa-Francorchamps	Spa	Belgium
Italian Grand Prix	06/09/2020	Autodromo Nazionale di Monza	Monza	Italy
Tuscan Grand Prix	13/09/2020	Autodromo Internazionale del Mugello	Mugello	Italy
Russian Grand Prix	27/09/2020	Sochi Autodrom	Sochi	Russia
Eifel Grand Prix	11/10/2020	Nürburgring	Nürburg	Germany
Portuguese Grand Prix	25/10/2020	Autódromo Internacional do Algarve	Portimão	Portugal
Emilia Romagna Grand Prix	01/11/2020	Autodromo Enzo e Dino Ferrari	Imola	Italy
Turkish Grand Prix	15/11/2020	Istanbul Park	Istanbul	Turkey
Bahrain Grand Prix	29/11/2020	Bahrain International Circuit	Sakhir	Bahrain
Sakhir Grand Prix	06/12/2020	Bahrain International Circuit	Sakhir	Bahrain
Abu Dhabi Grand Prix	13/12/2020	Yas Marina Circuit	Abu Dhabi	UAE

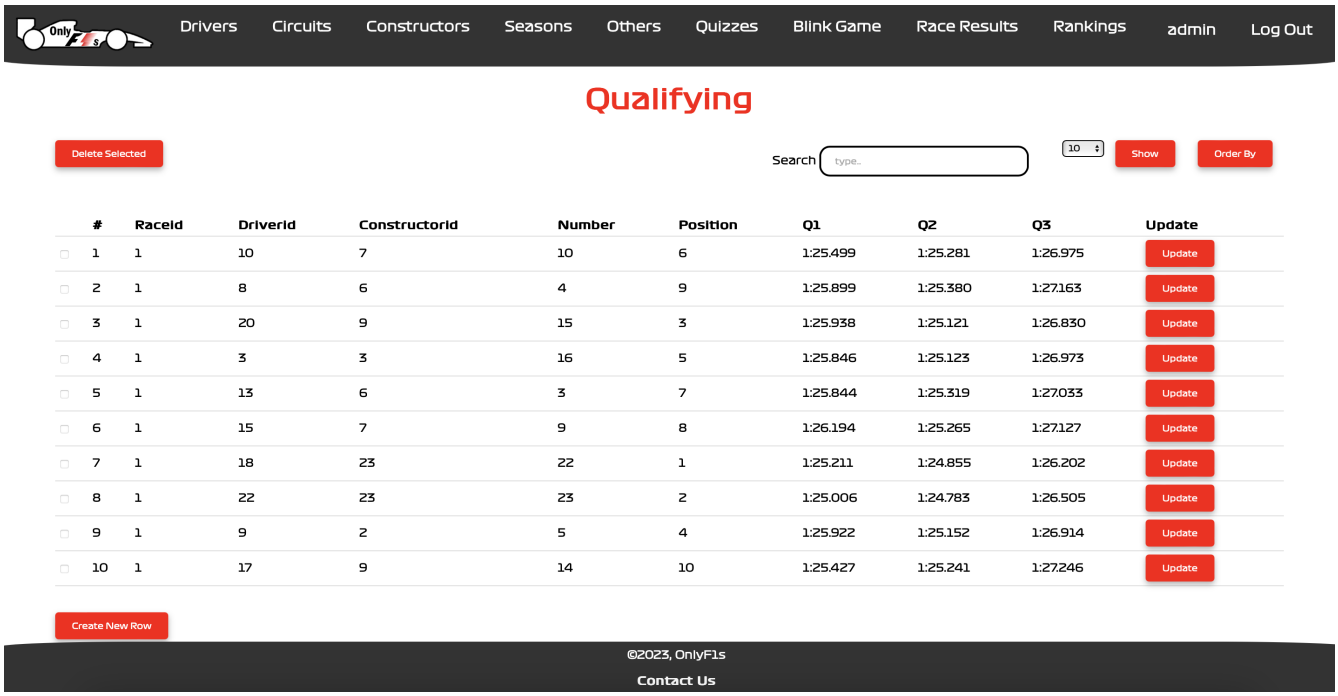
(a) Seasons Page

3.5.5. Others

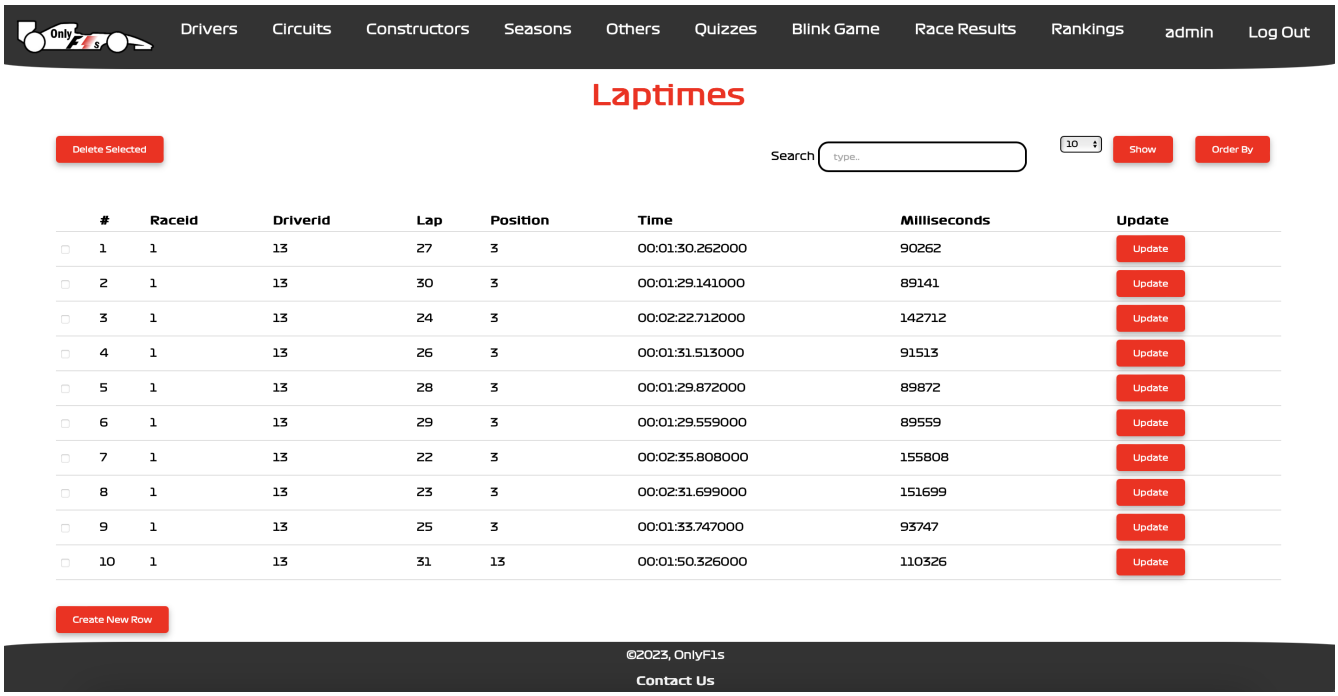
The pages were not explicitly utilized but were shown and generated generically.



(a) Others Page



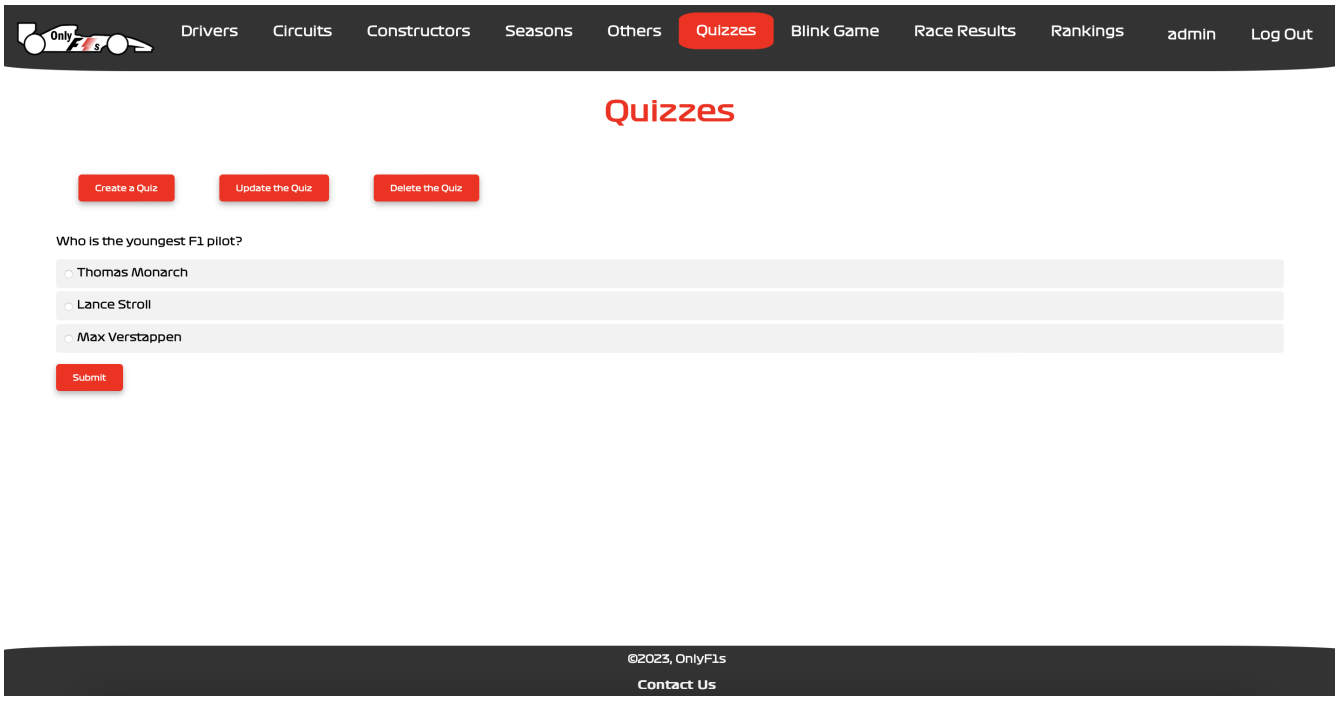
(a) An Example for the Others Page



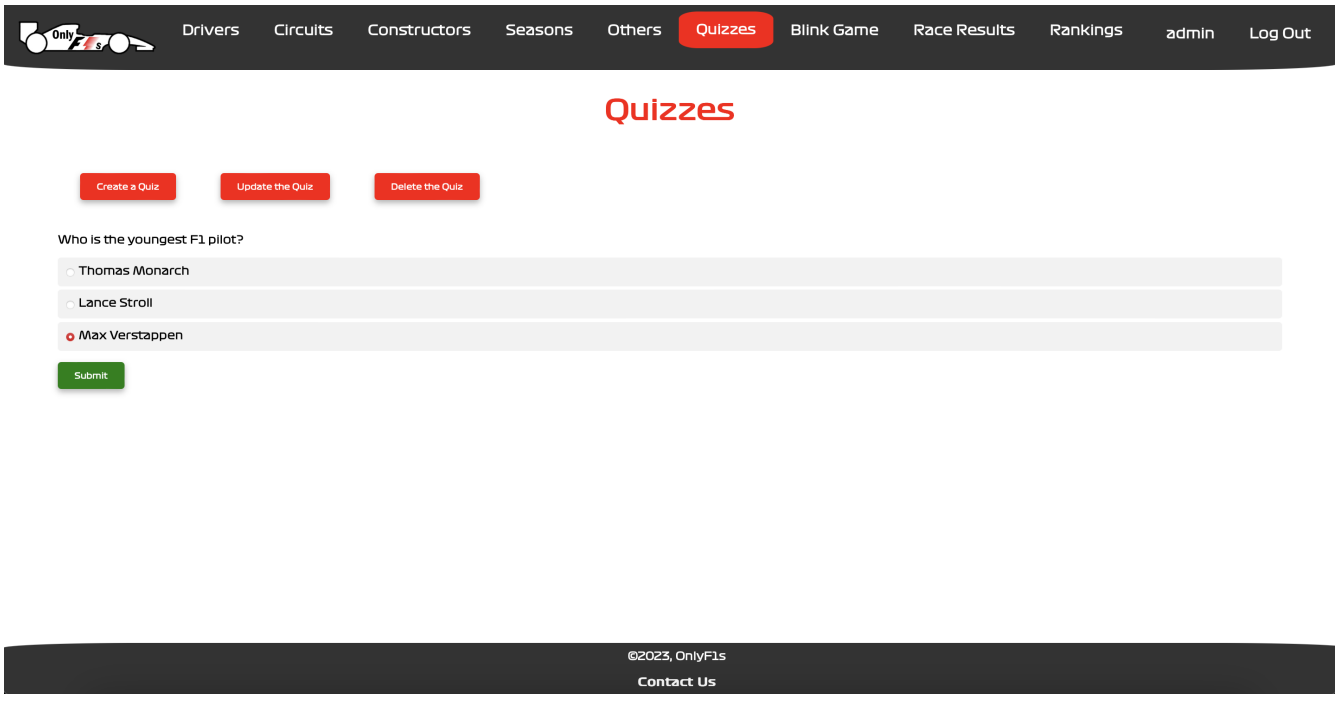
(a) An Example for the Others Page

3.5.6. Quiz

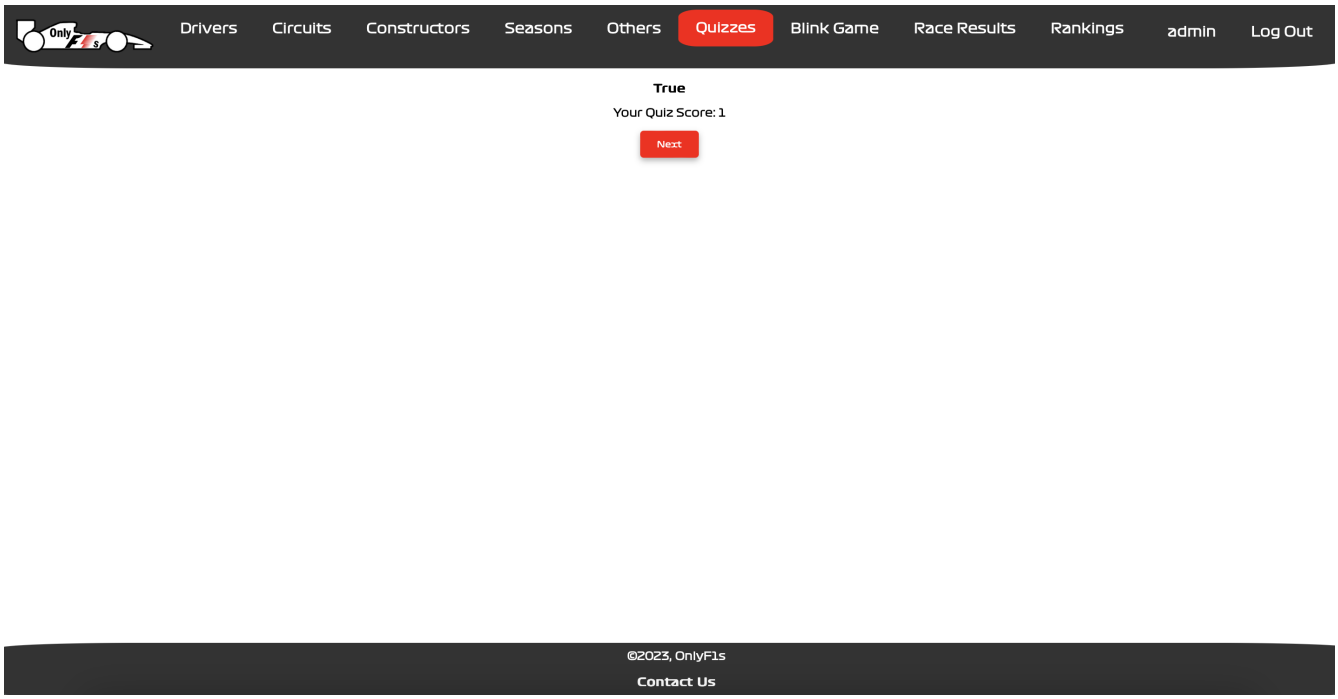
On this page, users can interact with quiz questions that show randomly, and their score is stored in the database. The initially established many-to-many relationship between the "Users" and "Quizzes" tables has been altered to one-to-many relations, specifically with the "Users" table. The adjustment aims to streamline the database model and enhance clarity in the representation of user-quiz relationships.



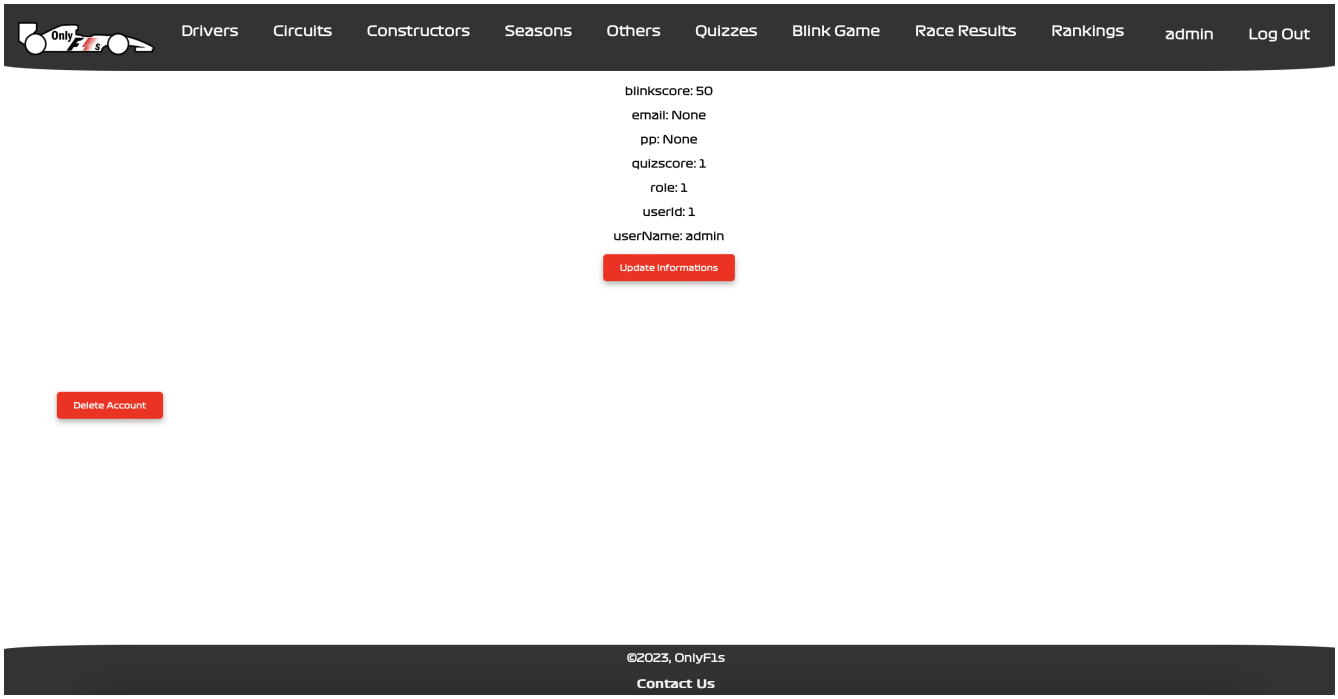
(a) Quiz Page



(a) Answering a Question

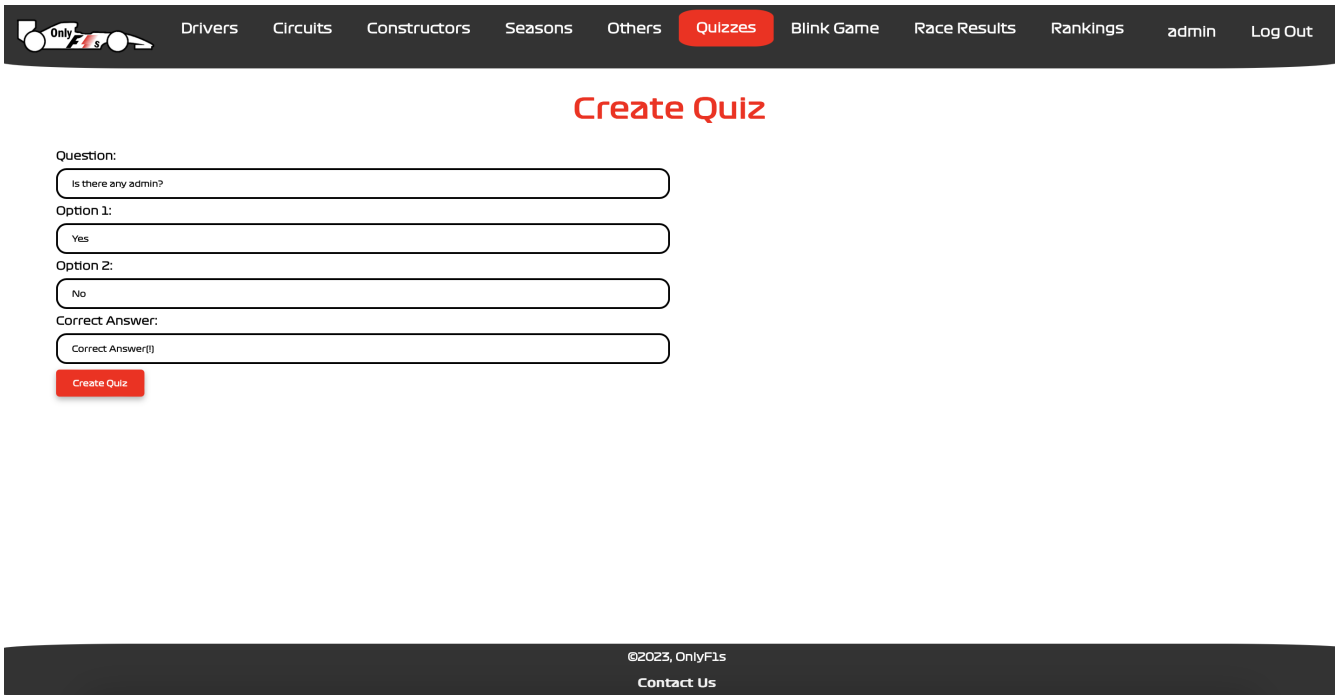


(a) Answer of the Question

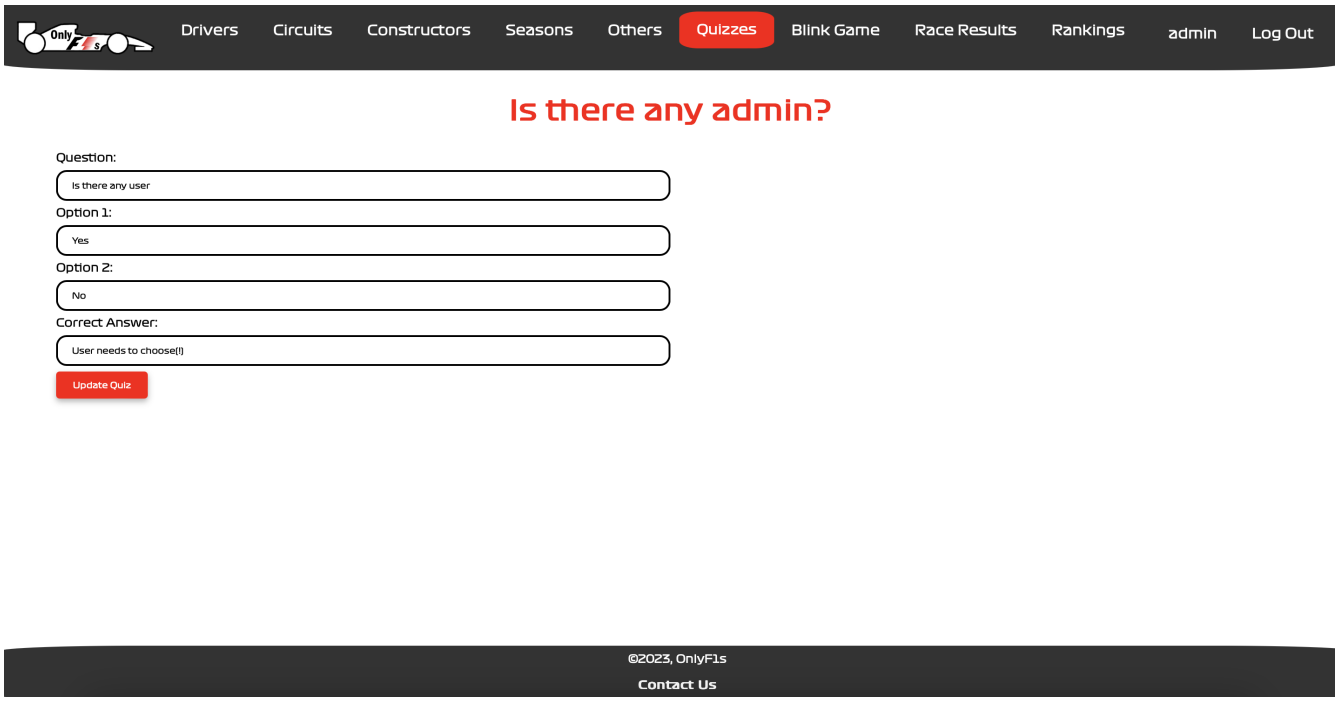


(a) Quiz Score Updated on User Page

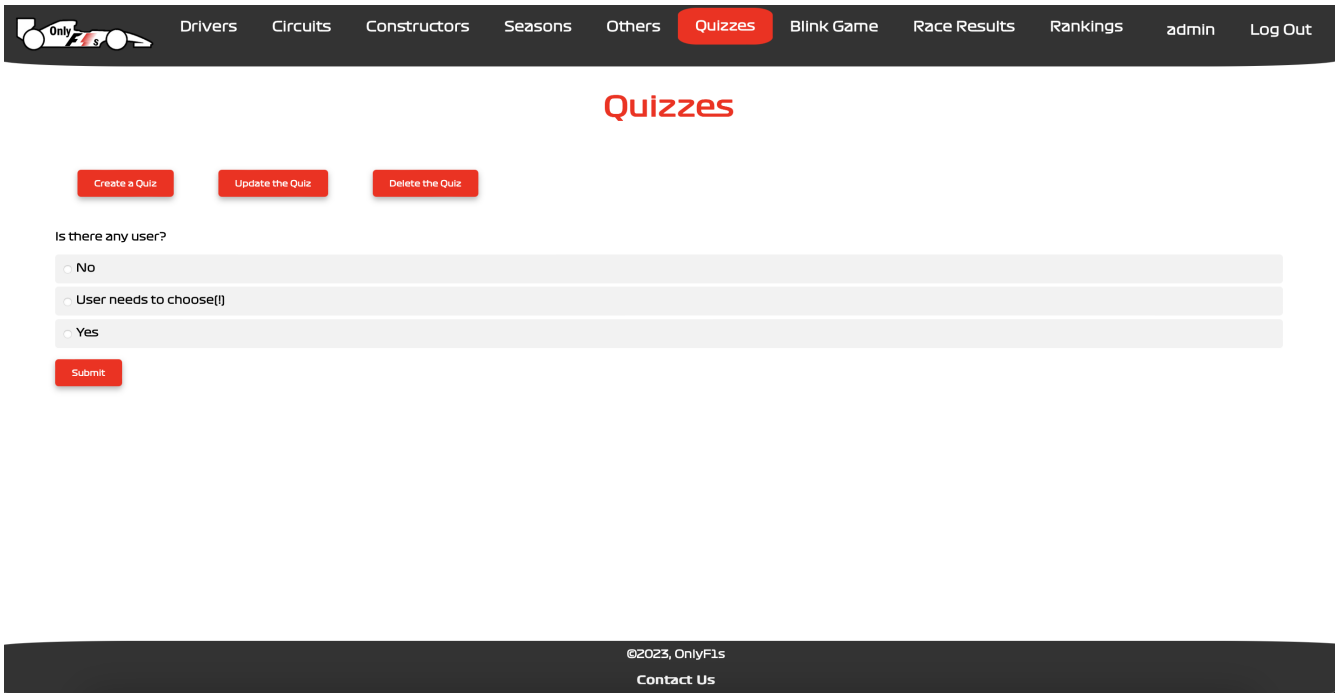
3.5.6.1. CRUD Operations On Quiz Page



(a) Create a New Question



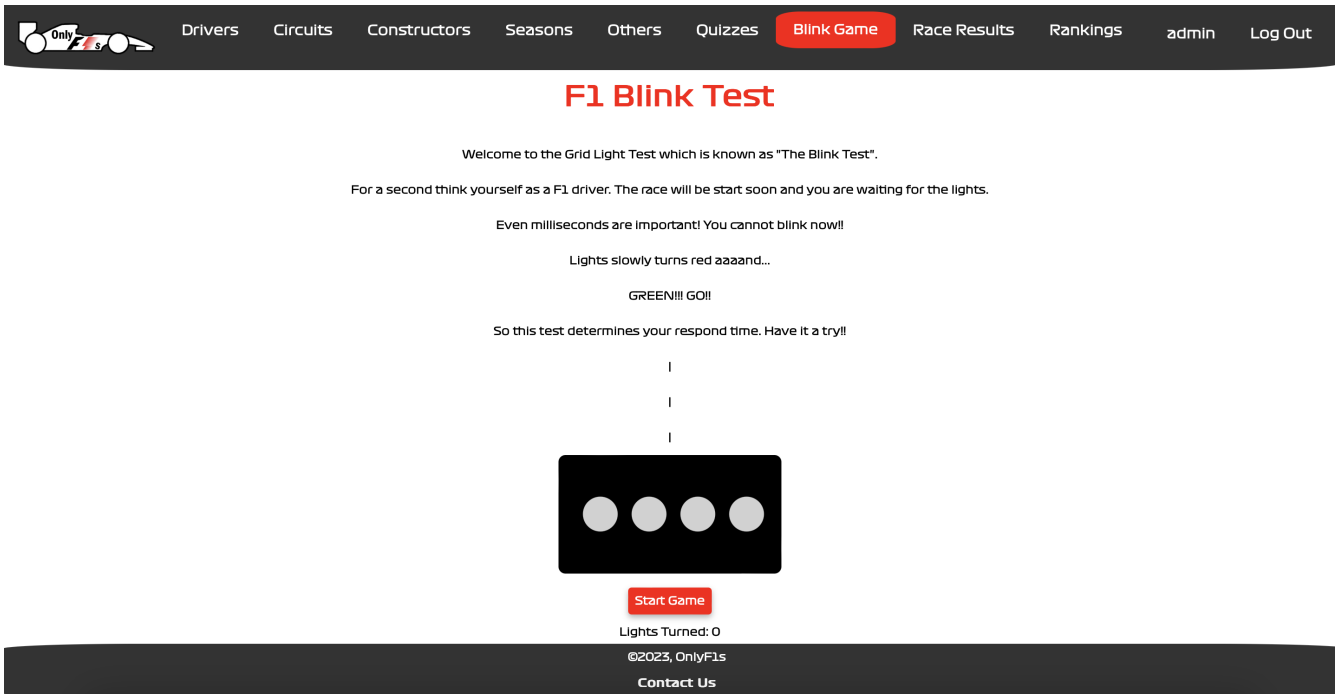
(a) Update a Question



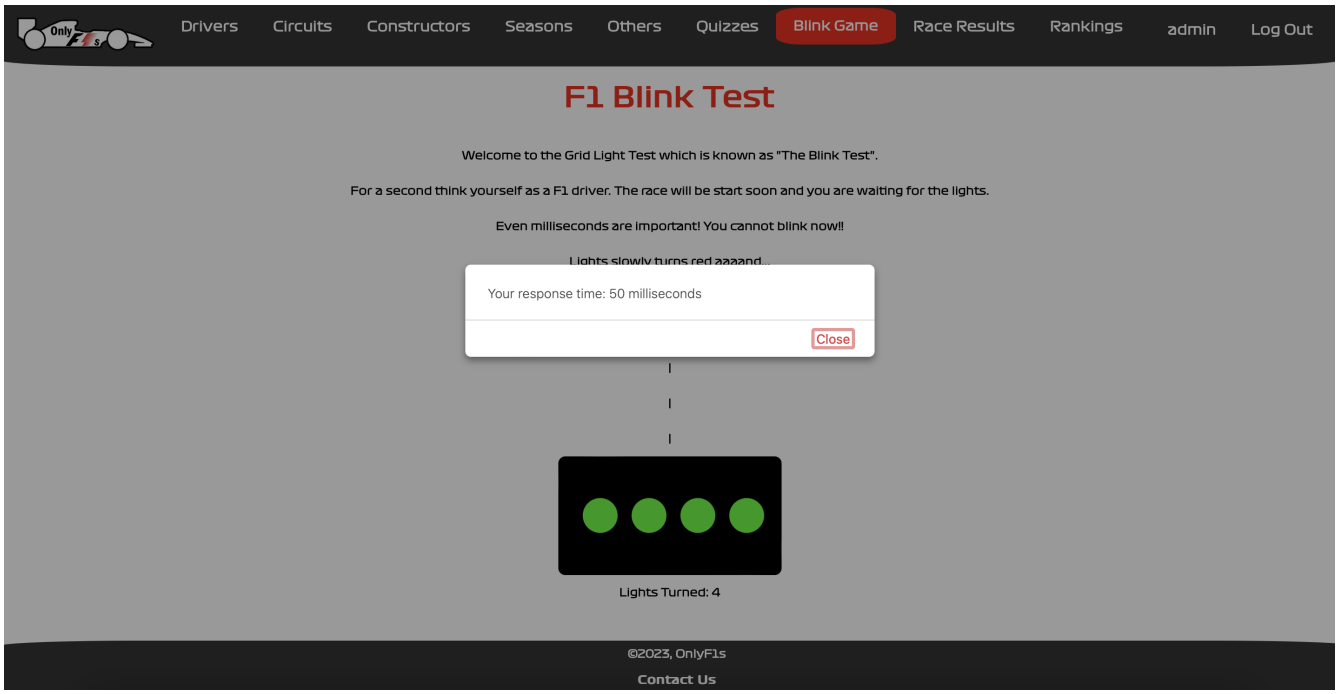
(a) Updated Question as Shown In the Page

3.5.7. Blink Game

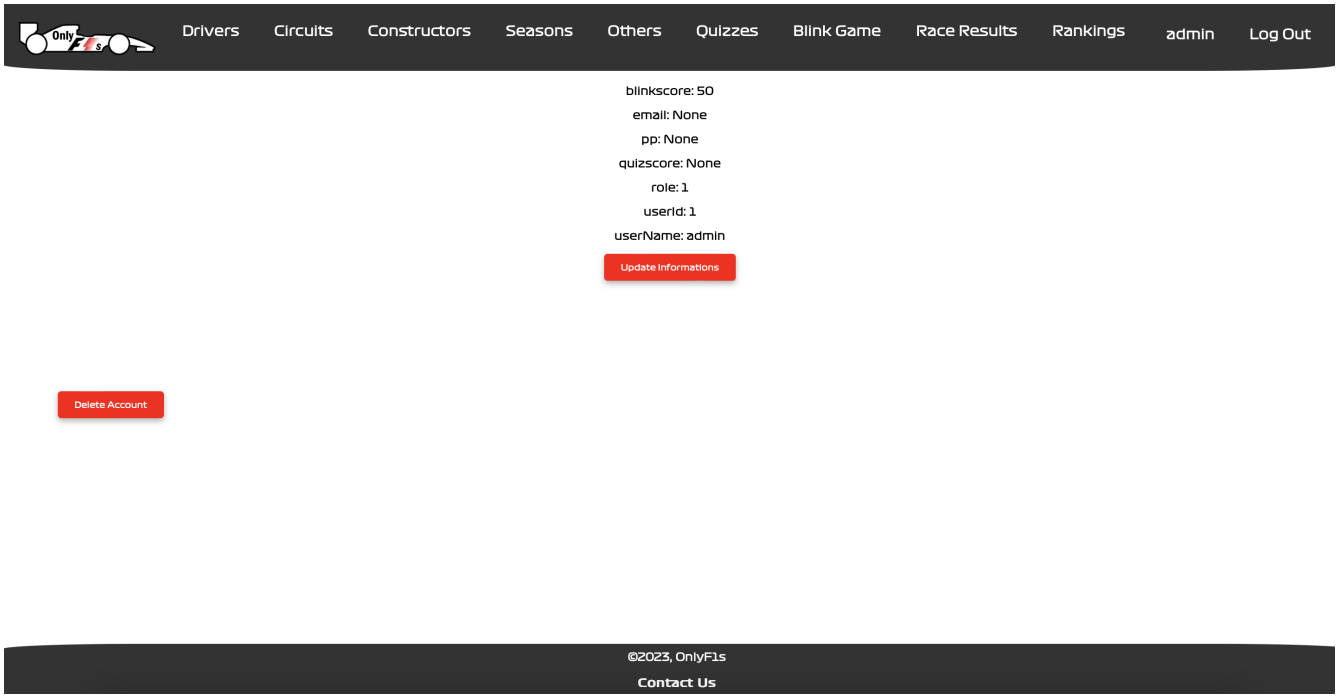
On this page, users can test their response time to the F1 starting light test. The best response time is stored in the database for each user.



(a) Blink Game Page



(a) Respond Time Pop-Up Page

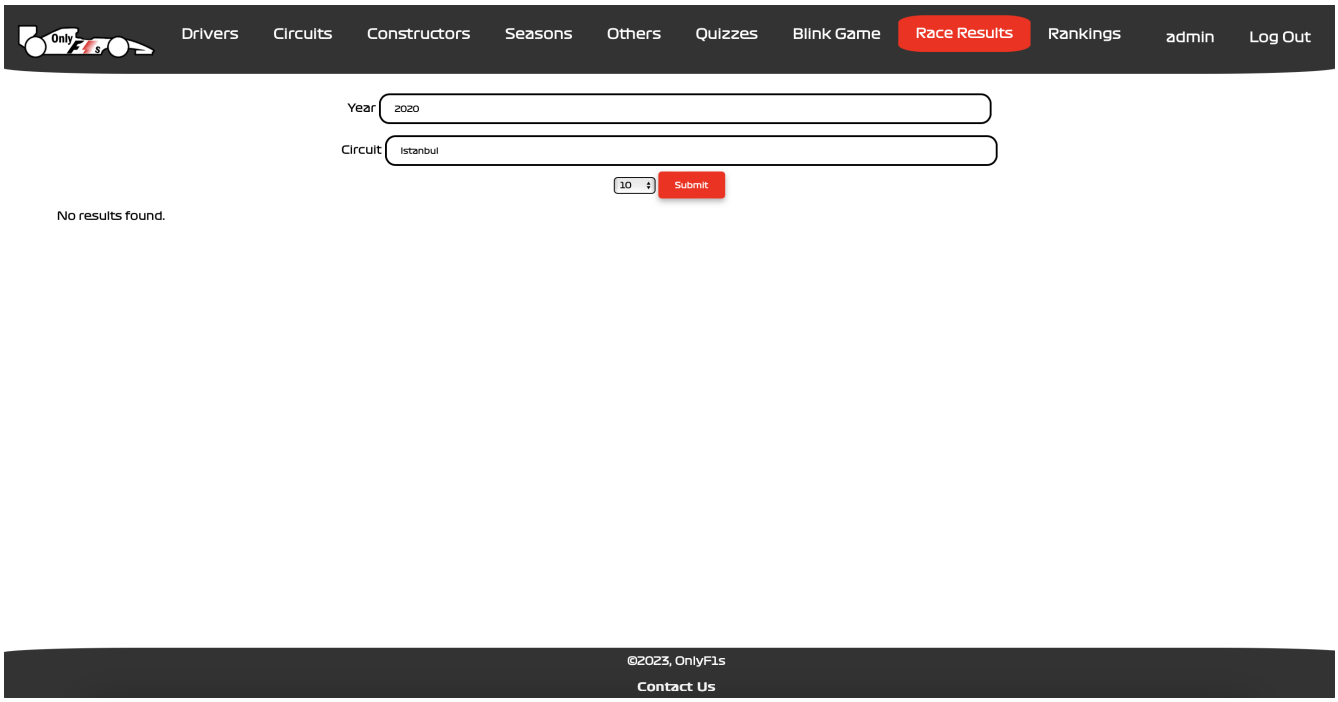


(a) Blink Game Score Updated on User Page

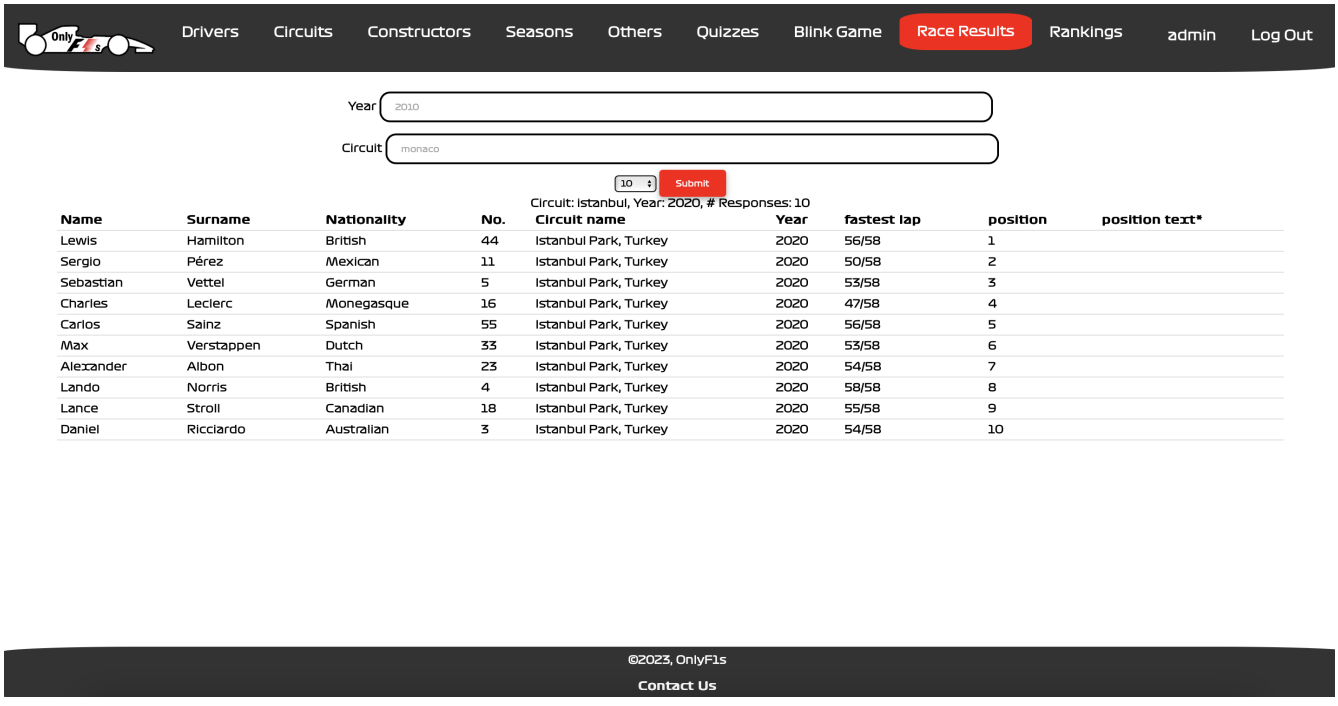
3.5.8. Race Results

The "Race Results" table, integral to our database project, consolidates data from races, drivers, circuits, and results to provide a comprehensive overview of race-related information. This table is designed to offer a detailed representation of race outcomes, featuring an incorporated response limiter and a search bar for enhanced functionality. The response limiter empowers users to manage the displayed data volume effectively,

fostering a focused examination of race results. Simultaneously, the search bar enables users to perform precise searches based on specific parameters, such as the year or circuit, facilitating efficient retrieval of race-related details from the database.



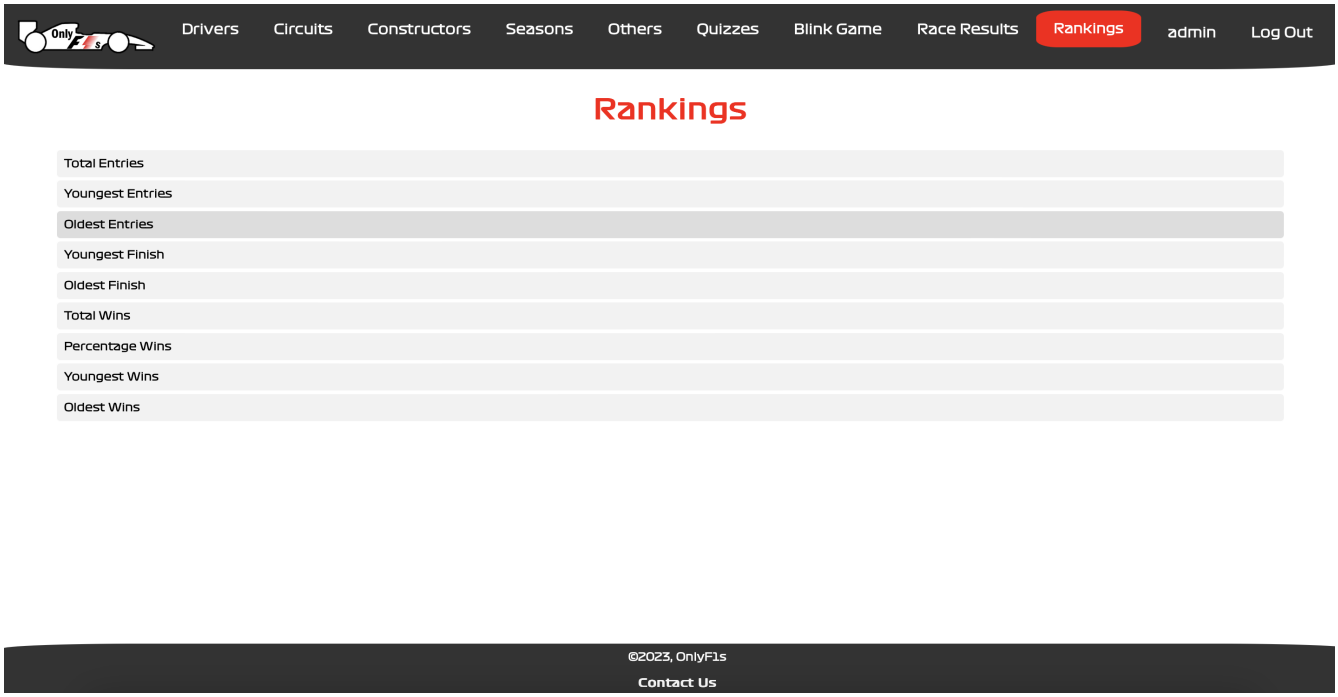
(a) Race Results Page



(a) Searched Results

3.5.9. Rankings

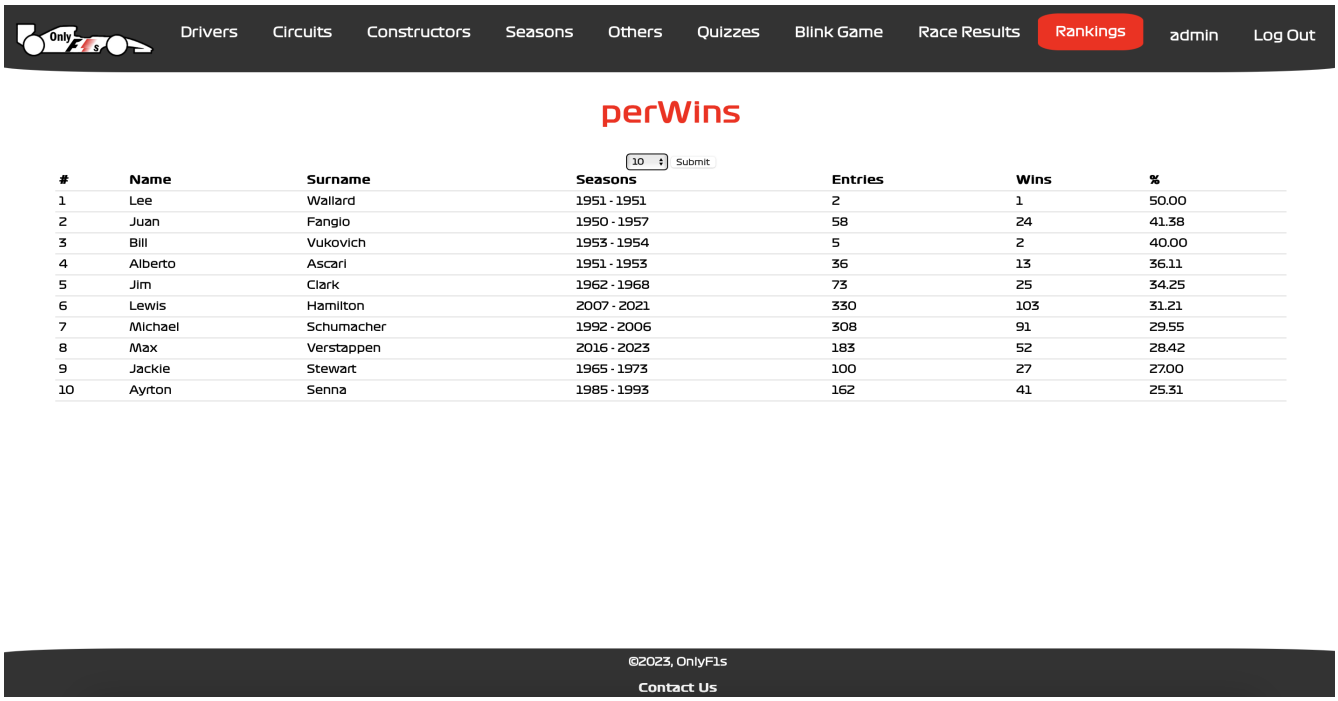
On this page, different comparison tables are shown as a list. Users can interact and see the specific comparison table.



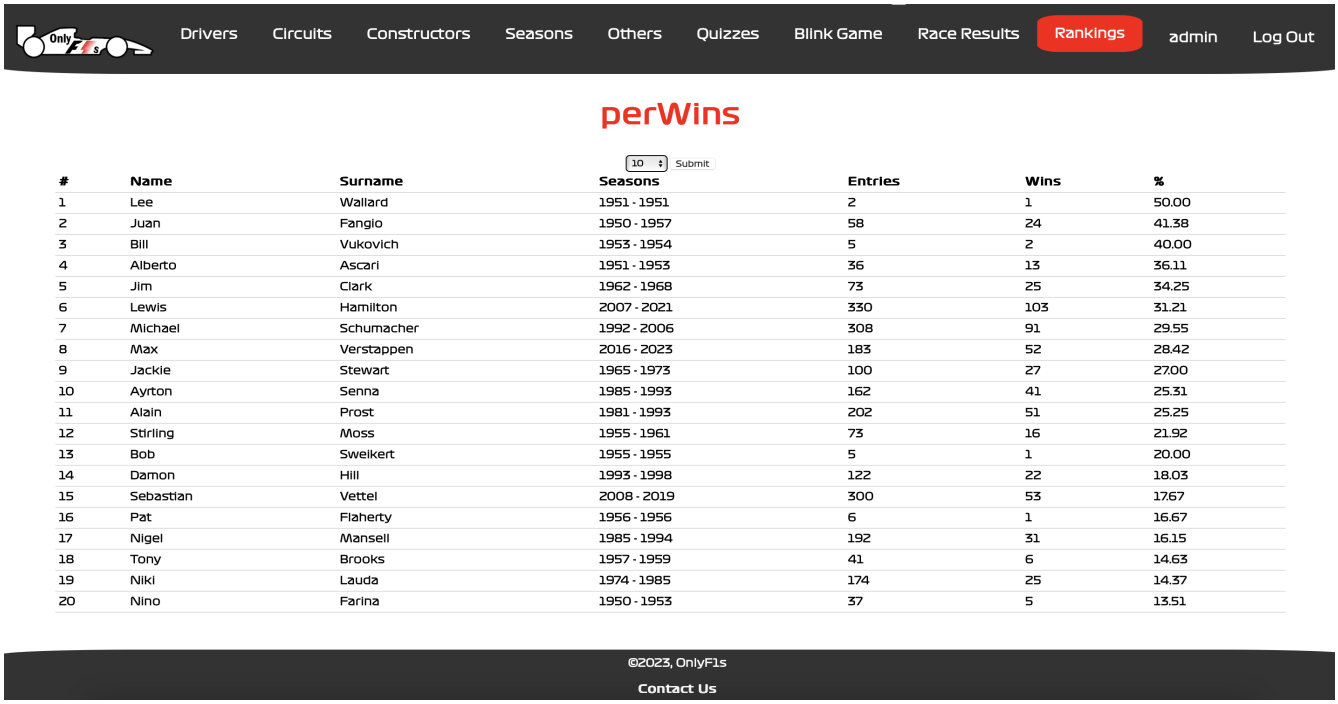
(a) Rankings Page

3.5.9.1. A Ranking Example

PerWins Page shows drivers, drivers' race entry counts, win rates, the season of the first wins, and the season of the last wins.



(a) perWins Ranking Page

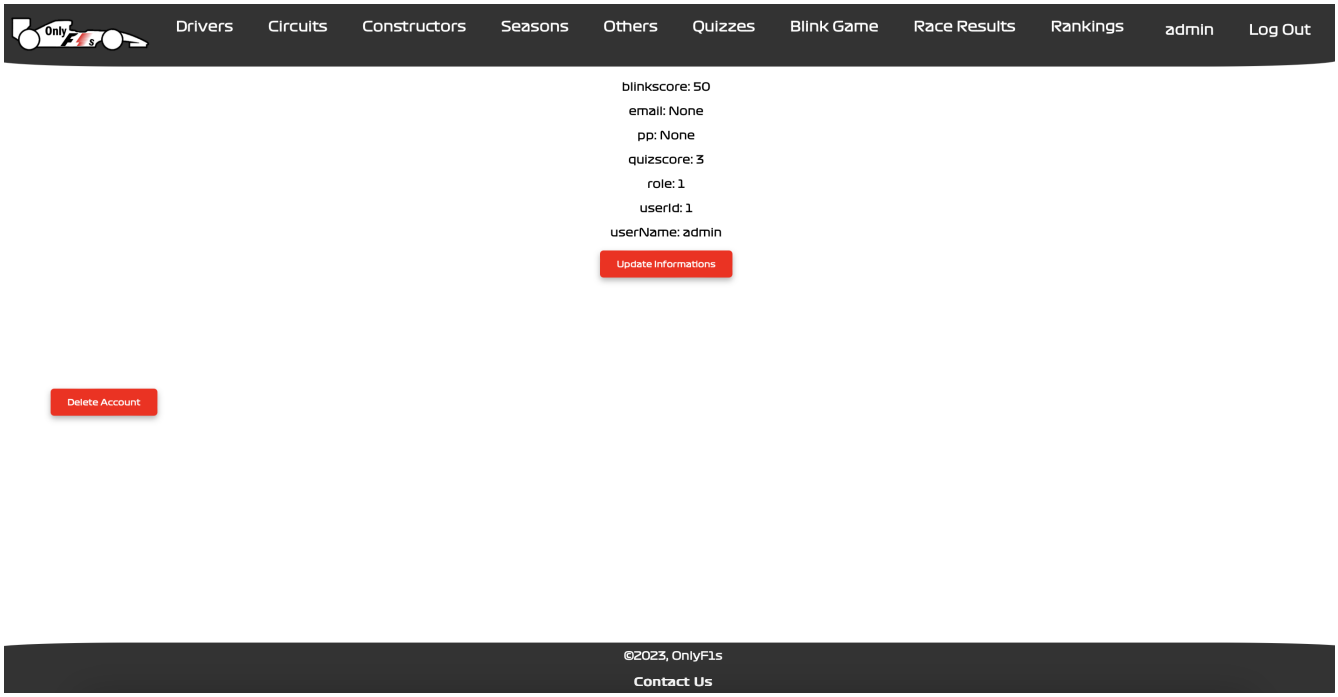


(a) Limited Ranking Table

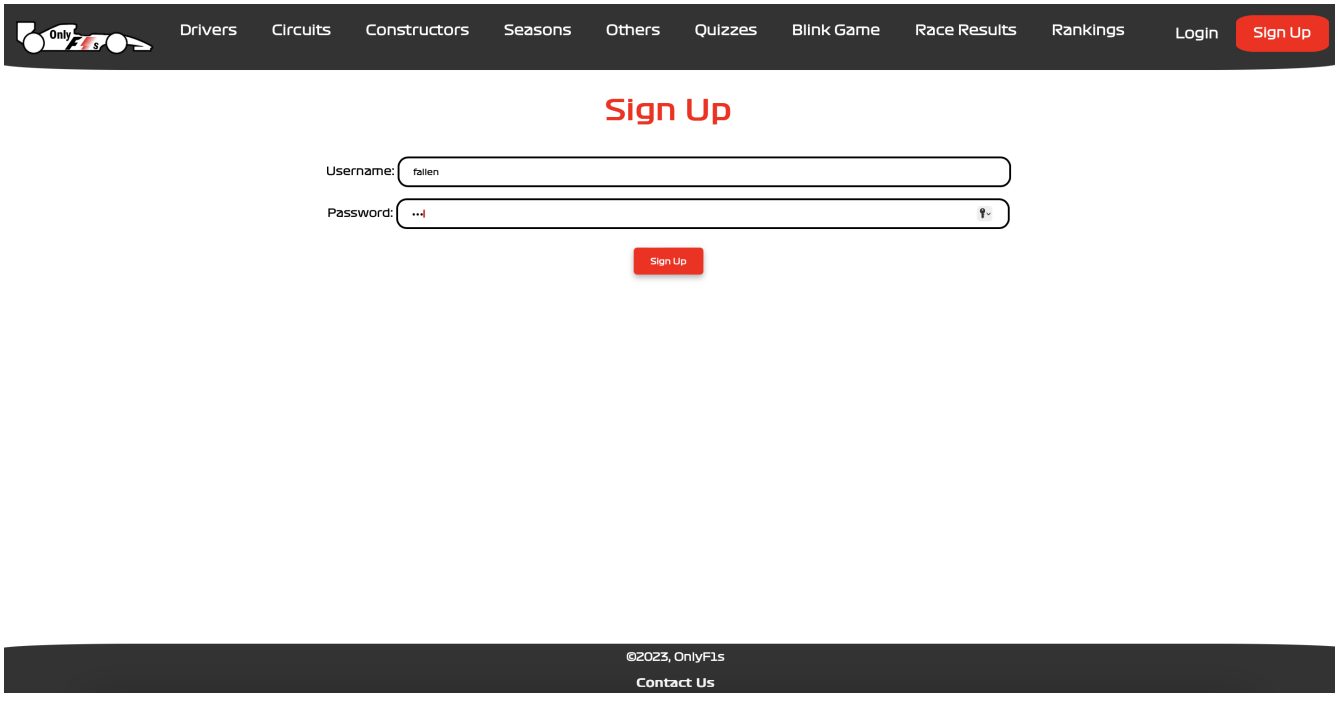
3.5.10. User

Upon logging in, users have access to their personalized information within the system, including their username, game scores, and email. For added flexibility, users are empowered to update their email and username if desired. Additionally, an option is provided for users to delete their accounts, granting them control over their account

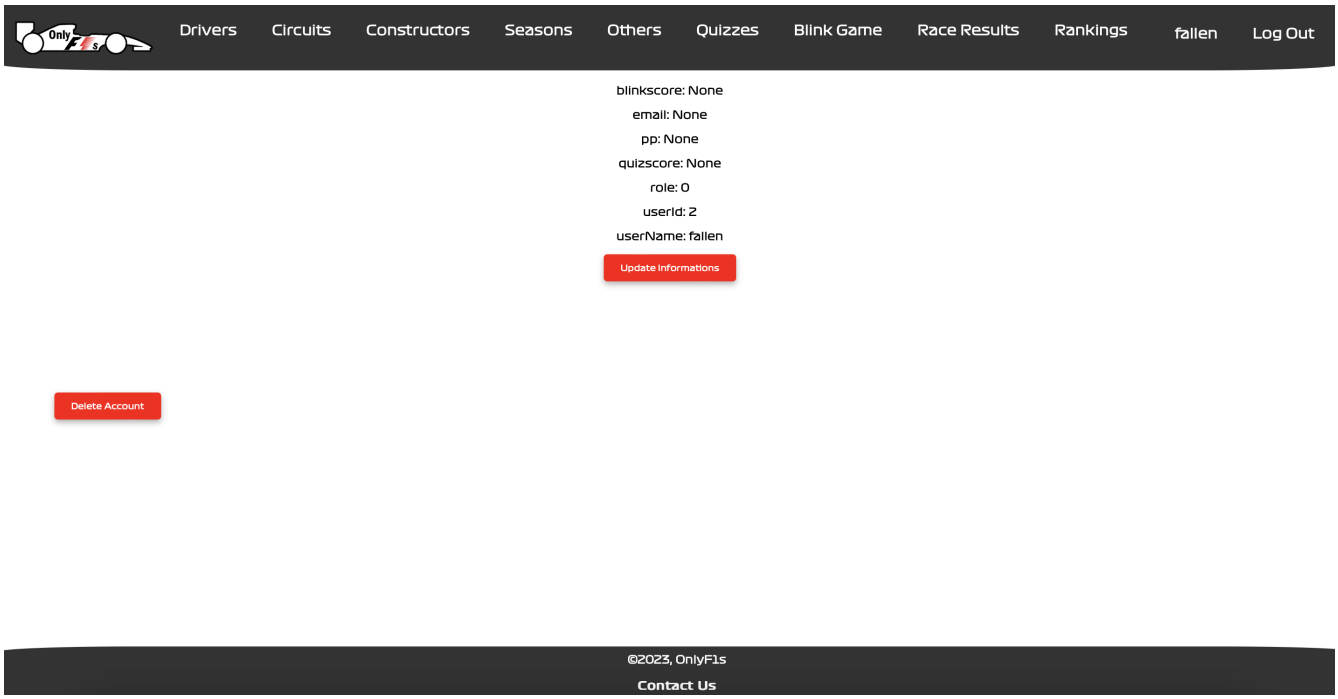
management. This user-centric approach ensures that individuals can easily manage and customize their profiles based on their preferences and changing information.



(a) User Information Page for Admin



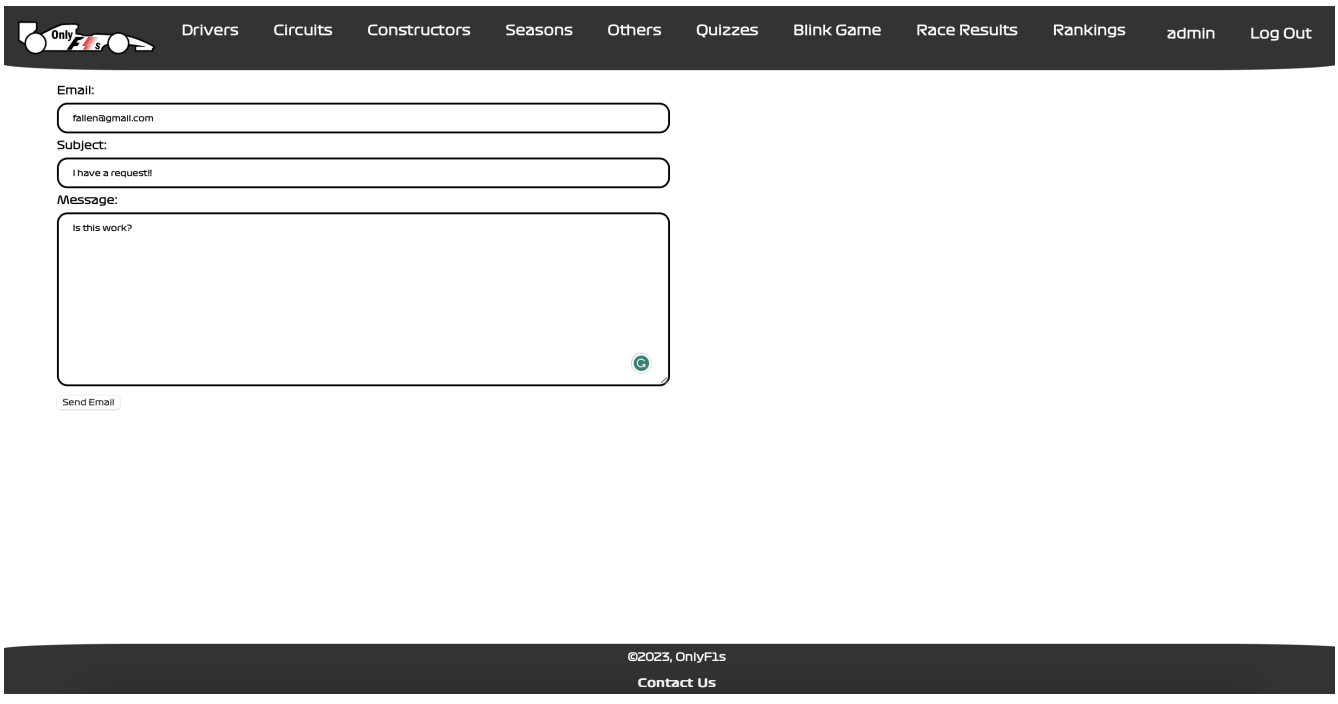
(a) Signing Up as a New User



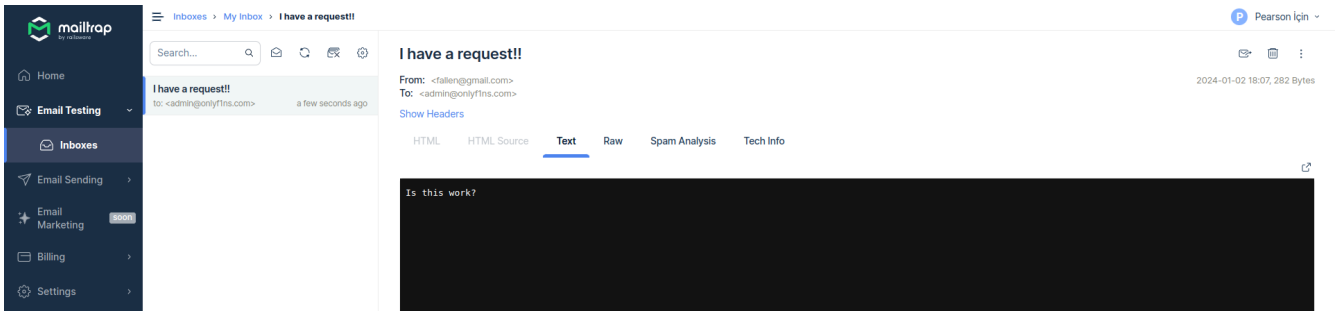
(a) New Created User Information

3.5.11. Contact Us

Users can interact with "Contact Us" at the bottom of every page to send emails to the admin mail.



(a) Contact Us Page



(a) An Email Sent By the User and Shown By the Admin Email

4. Conclusion

4.1. Discussions

We had designed a module to automatically generate and send emails for the 'Contact Us' section of the website. To ensure that the creation and sending of emails could be carried out asynchronously from the other functionalities of the site (preventing the site from remaining in a loading state until the email was sent), we planned to implement RabbitMQ. However, due to time constraints, we had to cancel the implementation.

We also planned to manage CRUD operations over a remote database (using services akin to Cockroach Labs [1]), but due to insufficient time, we had to cancel this as well.

4.2. Conclusion

Ultimately, we have established a Formula 1 Fandom website using Python's Flask framework. On this site, all users can view Formula 1 related content, participate in quizzes and Blink Test activities (with their scores visible on their respective panels), and an admin user can perform CRUD operations on the site's data. Throughout the project, all team members collaborated and contributed equally.

Bibliography

[1] Cockroach Labs. https://www.cockroachlabs.com/.

[2] RabbitMQ Features. https://www.rabbitmq.com/#features/.

[3] Developer Experience. Dockerizing. https://developerexperience.io/articles/dockerizing.

[4] Pallets Projects. Flask Documentation. https://flask.palletsprojects.com/en/3.0.x/.

[5] PostgreSQL Global Development Group. PostgreSQL - About. https://www.postgresql.org/about/.