NoSQL Projet : Formule 1

1. Introduction
    a. Sujet

Dans le cadre de cette ressource, nous avons créé notre propre base de données pour confirmer notre maîtrise du langage.

La base de données porte sur la **Formule 1**. Nous avons choisi ce sujet car nous avons de bonnes connaissances, et que la discipline ouvre la porte vers de nombreuses manipulations de bases de données.

    b. Structure

La structure reprend les résultats de la saison 2022 de Formule 1. Nous avons légèrement dénormalisé les données, sans pousser ce choix à l'extrême. Par exemple, l'année est présente dans beaucoup de collections pour associer les résultats entre eux (dénormalisation). Aussi, l'équipe des pilotes est insérée plusieurs fois. D'un autre côté, nous aurions pu dénormaliser encore plus, par exemple en insérant les points marqués par les pilotes à chaque course (ici, c'est calculable avec le barème associé à une saison). Nous avons donc travaillé sur une base qui reproduit les bases de données non-relationnelles (à petite échelle certes), en restant dans la simplicité.

Pilotes - Nom : str - Prénom : str - Équipe : str - Date de naissance : date

Qualifications
- Année : int
- Pays : str
- Ville : str
- Classement : array (object)
  - Pilote : str
  - Équipe : str
- Tour de pôle : int (en millisecondes)

Course - Année : int - Pays : str - Ville : str - Classement : objet   - Pilote : str

  - Équipe : str
- Durée : int (en millisecondes)
- Tours complétés : int (par le vainqueur)
- Pilote le plus rapide : str

Saisons
- Année : int
- Nombre de courses : int
- Pilote champion en titre : str
- Équipe championne en titre : str
- Barème de points en course : array (int)
- Barème de points en sprint : array (int)

Sprints
- Année : int
- Pays : str
- Ville : str
- Classement : objet
  - Pilote : str
  - Équipe : str
- Durée : int (en millisecondes)

Circuits
- Pays : str
- Ville : str
- Longueur : int (en mètres)

2. Schémas de validation
    a. Drivers

```
$jsonSchema: {
  bsonType: "object",
  title: "Drivers collection",
  required: ["lastName", "firstName", "team", "birthdate"],
  properties: {
    _id: {
      bsonType: "objectId",
    },
    lastName: {
      bsonType: "string",
      description:
        "Last name of the driver, must be a string and is required",
    },
    firstName: {
      bsonType: "string",
      description:
        "First name of the driver, must be a string and is required",
    },
    team: {
      bsonType: "string",
      description:
        "Team of the driver, must be an objectId and is required",
```

```
    },
    birthdate: {
      bsonType: "string",
      description:
        "Birthdate of the driver, must be a date and is required",
    },
  },
},
```

b. Qualifyings

```
$jsonSchema: {
  bsonType: "object",
  title: "Qualifyings collection",
  required: ["year", "country", "city", "standings", "poleLap"],
  properties: {
    _id: {
      bsonType: "objectId",
    },
    year: {
      bsonType: "int",
      description:
        "Year of the qualifying, must be an integer and is required",
    },
    country: {
      bsonType: "string",
      description:
        "Country of the qualifying, must be a string and is required",
    },
    city: {
      bsonType: "string",
      description:
        "City of the qualifying, must be a string and is required",
    },
    standings: {
      bsonType: "array",
      description:
        "Standings of the qualifying, must be an array and is required",
      required: ["driver", "team"],
      properties: {
        driver: {
          bsonType: "string",
          description:
            "Driver of the qualifying, must be a string and is required",
        },
        team: {
          bsonType: "string",
          description:
            "Team of the qualifying, must be a string and is required",
        },
      },
    },
    poleLap: {
      bsonType: "int",
      description:
        "Time of the pole lap in miliseconds, must be an integer and is required",
    },
  },
},
```

c. Races

```
$jsonSchema: {
  bsonType: "object",
  title: "Races collection",
  required: [
    "year",
    "country",
    "city",
    "standings",
    "duration",
    "lapsCompleted",
    "fastestDriver",
  ],
  properties: {
    _id: {
      bsonType: "objectId",
    },
    year: {
      bsonType: "int",
      description: "Year of the race, must be an integer and is required",
    },
    country: {
      bsonType: "string",
      description: "Country of the race, must be a string and is required",
    },
    city: {
      bsonType: "string",
      description: "City of the race, must be a string and is required",
    },
    standings: {
      bsonType: "array",
      description:
        "Standings of the qualifying, must be an array and is required",
      required: ["driver", "team"],
      properties: {
        driver: {
          bsonType: "string",
          description:
            "Driver of the qualifying, must be a string and is required",
        },
        team: {
          bsonType: "string",
          description:
            "Team of the qualifying, must be a string and is required",
        },
      },
    },
    duration: {
      bsonType: "int",
      description:
        "Duration of the race in milliseconds, must be an integer and is required",
    },
    lapsCompleted: {
      bsonType: "int",
      description:
        "Number of laps completed by the winner, must be an integer and is required",
    },
    fastestDriver: {
      bsonType: "string",
      description:
        "Name of the driver with the fastest lap, must be a string and is required",
    },
  },
},
```

d. Seasons

```
  $jsonSchema: {
    bsonType: "object",
    title: "Seasons collection",
        required:  ["year",  "numberOfRaces",  "defendingDriverChampion",  "defendingTeamChampion",  "racePointsSystem",
"sprintPointSystem"],
    properties: {
      _id: {
        bsonType: "objectId",
      },
      year: {
        bsonType: "int",
        description: "Year of the season, must be an integer and is required",
      },
      numberOfRaces: {
        bsonType: "int",
        description: "Number of races in the season, must be an integer and is required",
      },
      defendingDriverChampion: {
        bsonType: "string",
        description: "Defending driver champion, must be an objectId and is required",
      },
      defendingTeamChampion: {
        bsonType: "string",
        description: "Defending team champion, must be an objectId and is required",
      },
      racePointsSystem: {
        bsonType: "array",
        description: "Race points system, must be an array and is required",
        items: {
          bsonType: "int",
          description: "Points for each position, must be an integer",
        },
      },
      sprintPointSystem: {
        bsonType: "array",
        description: "Sprint points system, must be an array and is required",
        items: {
          bsonType: "int",
          description: "Points for each position, must be an integer",
        },
      },
    }
  },
```

e. Sprints

```
  $jsonSchema: {
    bsonType: "object",
    title: "Sprints collection",
    required: ["year", "country", "city", "standings", "duration"],
    properties: {
      _id: {
        bsonType: "objectId",
      },
      year: {
        bsonType: "int",
        description:
          "Year of the sprint race, must be an integer and is required",
      },
      country: {
        bsonType: "string",
        description:
          "Country of the sprint race, must be a string and is required",
      },
      city: {
```

```
      bsonType: "string",
      description:
        "City of the sprint race, must be a string and is required",
    },
    standings: {
      bsonType: "array",
      description:
        "Standings of the sprint race, must be an array and is required",
      required: ["driver", "team"],
      properties: {
        driver: {
          bsonType: "string",
          description:
            "Driver of the sprint race, must be a string and is required",
        },
        team: {
          bsonType: "string",
          description:
            "Team of the sprint race, must be a string and is required",
        },
      },
    },
    duration: {
      bsonType: "int",
      description:
        "Duration of the sprint race, must be an integer and is required",
    },
  },
},
```

f. Tracks

```
$jsonSchema: {
  bsonType: "object",
  title: "Tracks collection",
  required: ["country", "city", "length"],
  properties: {
    _id: {
      bsonType: "objectId",
    },
    country: {
      bsonType: "string",
      description: "Country of the track, must be a string and is required",
    },
    city: {
      bsonType: "string",
      description: "City of the track, must be a string and is required",
    },
    length: {
      bsonType: "int",
      description:
        "Length of the track in meters, must be an integer and is required",
    },
  },
},
```

3. Requêtes de test

Dans le CLI mongosh

```
f1> db.getCollectionInfos()
```

Chaque collection est testée individuellement

```
db.collection(collection)
  .insertOne({
    test: 0,
  })
  .then(() => {
    console.log("insertion success");
  })
  .catch((err) => {
    console.error(err);
  });
```

4. Requêtes simples

Combien de pilotes différents ont gagné une course ?

```
var count = 0;
var memo = [];
db.collection("races")
  .find({})
  .toArray()
  .then((docs) => {
    docs.forEach((doc) => {
      var winner = doc.standings[0].driver;
      if (!memo.includes(winner)) {
        count++;
        memo.push(winner);
      }
    });
  }).then(() => {
    console.log("Q4.1 : Combien de pilotes différents ont gagné une course ?");
    console.log(`>>> ${count} (${memo})`);
    console.log();
  });
```

Quelle est la longueur moyenne d'un circuit en 2022 (en mètres) ?

```
var count = 0;
var sum = 0;
db.collection("tracks")
  .find({})
  .toArray()
  .then((docs) => {
    count = docs.length;
    docs.forEach((doc) => {
      sum += doc.length;
    });
  }).then(() => {
    console.log(
      "Q4.2 : Quelle est la longueur moyenne d'un circuit en 2022 (en mètres) ?"
    );
    console.log(">>> " + (sum / count).toFixed(2) + "m");
    console.log();
  });
```

Quel pilote a obtenu le plus de pôles positions (1er en qualifications) ?

```
var count = {};
db.collection("qualifyings")
  .find({})
```

```
  .toArray()
  .then((docs) => {
    // Count the number of pole positions for each driver
    docs.forEach((doc) => {
      var poleman = doc.standings[0].driver;
      if (count[poleman] == undefined) {
        count[poleman] = 1;
      } else {
        count[poleman]++;
      }
    });

    // Find the driver(s) with the most poless
    var max = 0;
    var driver = [];
    for (var key in count) {
      if (count[key] > max) {
        max = count[key];
        driver = [key];
      } else if (count[key] == max) {
        driver.push(key);
      }
    }

    // Returns a drvier or an array of drivers (str)
    if (driver.length == 1) {
      count = driver[0];
    } else {
      count = driver;
    }
  }).then(() => {
    console.log(
      "Q4.3 : Quel pilote a obtenu le plus de pôles positions (1er en qualifications) ?"
    );
    console.log(">>> " + count);
    console.log();
  });
```

Combien de podiums l'équipe Ferrari a-t-elle obtenu ?

```
var count = 0;
db.collection("races")
  .find({})
  .toArray()
  .then((docs) => {
    docs.forEach((doc) => {
      var podium = doc.standings.slice(0, 3);
      podium.forEach((driver) => {
        if (driver.team == "Ferrari") {
          count++;
        }
      });
    });
  }).then(() => {
    console.log("Q4.4 : Combien de podiums l'équipe Ferrari a-t-elle obtenu ?");
    console.log(">>> " + count);
    console.log();
  });
```

Quel est l'âge moyen des pilotes ?

```
var ages = [];
db.collection("drivers")
  .find({})
  .toArray()
  .then((docs) => {
```

```
  docs.forEach((doc) => {
    // traite la date de naissance
    var birthdate = doc.birthdate.split("-");
    var date = new Date(birthdate[2], birthdate[1] - 1, birthdate[0]);

    // calcule l'âge
    var diff = Date.now() - date.getTime();
    var age_date = new Date(diff);
    var age = Math.abs(age_date.getUTCFullYear() - 1970);

    ages.push(age);
  });
}).then(() => {
  console.log("Q4.5 : Quel est l'âge moyen des pilotes ?");
  console.log(">>> " + (ages.reduce((a, b) => a + b) / ages.length).toFixed(2) + " ans");
  console.log();
});
```

5. Opérations de modification et de suppression

Modifier les données pour que l'année soit 2021.

```
db.collection("qualifyings").updateMany(
  { year: 2022 },
  { $set: { year: 2021 } }
);

db.collection("races").updateMany({ year: 2022 }, { $set: { year: 2021 } });
db.collection("sprints").updateMany({ year: 2022 }, { $set: { year: 2021 } });
db.collection("seasons").updateMany({ year: 2022 }, { $set: { year: 2021 } });
```

Échanger l'équipe des pilotes Hamilton et Verstappen.

```
var hamiltonTeam = "";
var verstappenTeam = "";
db.collection("drivers")
  .find({ lastName: { $in: ["Hamilton", "Verstappen"] } })
  .toArray()
  .then((docs) => {
    var h = "";
    var v = "";
    docs.forEach((doc) => {
      if (doc.lastName == "Hamilton") {
        h = doc.team;
      } else {
        v = doc.team;
      }
    });
    hamiltonTeam = h;
    verstappenTeam = v;
  })
  .then(() => {
    var collections = ["races", "qualifyings", "sprints"];

    // Verstappen -> Hamilton
    db.collection("drivers").updateOne(
      { lastName: "Verstappen" },
      { $set: { team: hamiltonTeam } }
    );

    collections.forEach((collection) => {
```

```
    db.collection(collection).updateMany(
      { "standings.driver": "Verstappen" },
      { $set: { "standings.$.team": hamiltonTeam } }
    );
  });

  // Hamilton -> Verstappen
  db.collection("drivers").updateMany(
    { lastName: "Hamilton" },
    { $set: { team: verstappenTeam } }
  );

  var collections = ["races", "qualifyings", "sprints"];
  collections.forEach((collection) => {
    db.collection(collection).updateMany(
      { "standings.driver": "Hamilton" },
      { $set: { "standings.$.team": verstappenTeam } }
    );
  });
});
```

Déclasser Carlos Sainz de 5 places pour la course de Miami.

```
var classement;
db.collection('races').find(
  { city: "Miami" }
).toArray().then((docs) => {
  // Récupération du classement de la course et changement des positions
  classement = docs[0].standings;
  for (let i = 0; i < classement.length; i++) {
    pilote = classement[i].driver;
    if (pilote == "Sainz") {
      var j = i;
      while (j + 1 < classement.length && j < i + 5) {
        var tmp = classement[j];
        classement[j] = classement[j + 1];
        classement[j + 1] = tmp;
        j++;
      }
      break;
    }
  }
}).then(() => {
  // Enregistrement du nouveau classement
  db.collection('races').updateOne(
    { city: "Miami" },
    {
      $set: { standings: classement }
    });
```

Supprimer les Grand Prix dont les pays sont représentés par plusieurs circuits.

```
collections = ["tracks", "qualifyings", "sprints", "races"];
pays = []
non_uniques = []

// Rechercher les pays accueillant plusieurs circuits
db.collection("tracks").find().toArray().then((docs) => {
  docs.forEach((doc) => {
    if (pays.includes(doc.country)) {
      non_uniques.push(doc.country)
      console.log("[++] " + doc.country);
    } else {
      pays.push(doc.country)
      console.log("[+] " + doc.country);
```

```
    }
  });

}).then(() => {
  collections.forEach((collection) => {
    console.log("Collection : " + collection);
    non_uniques.forEach((pays) => {
      db.collection(collection).deleteMany({ country: pays })
    })
  })
})
```

Échanger les positions des coéquipiers en sprint.

```
var positions;
db.collection("sprints").find().toArray().then((docs) => {
  docs.forEach((doc) => {
    positions = {};
    var classement = doc.standings;
    console.log("1", doc.city, classement);

    for (let i = 0; i < classement.length; i++) {
      var line = classement[i];
      if (positions[line.team] === undefined) {
        positions[line.team] = i;
      }
      else {
        var tmp = classement[positions[line.team]];
        classement[positions[line.team]] = classement[i];
        classement[i] = tmp;
      }
    }
    console.log("2", doc.city, classement);
    db.collection("sprints").updateOne({ _id: doc._id }, { $set: { standings: classement } });
  })
})
```

6. Requêtes complexes

Combien de podiums le champion en titre par équipes a-t-il obtenu ?

```
var count = 0
db.collection("races").aggregate([
  {
    $lookup: {
      from: "seasons",
      localField: "year",
      foreignField: "year",
      as: "theSeason",
    },
  }
])
  .toArray()
  .then((results) => {
    results.forEach((result) => {
      var champion = result.theSeason[0].defendingTeamChampion;
      for (let i = 0; i < 3; i++) {
        if (result.standings[i].team == champion) {
          count++;
        }
      }
    });
  }).then(() => {
```

```
  console.log("Q6.1 : Combien de podiums le champion en titre par équipes a-t-il obtenu ?");
  console.log(">>> " + count + " podiums");
  console.log();
});
```

Sur quel circuit la vitesse moyenne était-elle la plus élevée en course ?

```
db.collection("races")
 .aggregate([
  {
   $lookup: {
    from: "tracks",
    localField: "city",
    foreignField: "city",
    as: "theTrack",
   },
  },
 ])
 .toArray()
 .then((results) => {
  let arr = [];

  results.forEach((result) => {
   let country = result.country;
   let city = result.city;
   let duration = result.duration / 3600000;
   let dist = (result.theTrack[0]["length"] * result.lapsCompleted) / 1000;

   arr.push({
    country: country,
    city: city,
    duration: duration,
    distance: dist,
    speed: dist / duration,
   });
  });

  let best = arr[0];

  arr.forEach((item) => {
   if (item.speed > best.speed) {
    best = item;
   }
  });

  console.log("Q6.2 : Sur quel circuit la vitesse moyenne était-elle la plus élevée en course ?");
  console.log(">>> " + best.country + ", " + best.city + " at " + best.speed.toFixed(2) + " km/h");
  console.log();
 })
 .catch((err) => console.error(err));
```

Qui a gagné le plus de positions entre la qualification et la course en France ?

```
var max = 0;
var pilotes = [];
db.collection("races").aggregate([
 {
  $match: {
   city: "Le Castellet"
  }
 },
 {
  $lookup: {
   from: "qualifyings",
   localField: "city",
```

```
      foreignField: "city",
      as: "theQuali",
    },
  }
])
  .toArray()
  .then((result) => {

    var qualif = result[0].theQuali[0].standings;
    var course = result[0].standings;
    var pilote;

    for (var i = 0; i < qualif.length; i++) {
      pilote = qualif[i].driver;
      for (var j = 0; j < course.length; j++) {
        if (course[j].driver == pilote) {
          if (i - j > max) {
            max = i - j;
            pilotes = [];
            pilotes.push(pilote);
          } else {
            if (i - j == max) {
              pilotes.push(pilote);
            }
          }
        }
      }
    }
    if (pilotes.length == 1) {
      pilotes = pilotes[0];
    }
  }).then(() => {
    console.log("Q6.3 : Qui a gagné le plus de positions entre la qualification et la course en France ?");
    console.log(`>>> ${pilotes} a/ont gagné ${max} postions entre la qualification et la course.`);
    console.log();
  });
```

Quel est le classement du championnat des pilotes ?

```
var general_standings = {};

db.collection("races").aggregate([
  {
    $lookup: {
      from: "sprints",
      localField: "city",
      foreignField: "city",
      as: "theSprint"
    },
  },
  {
    $lookup: {
      from: "seasons",
      localField: "year",
      foreignField: "year",
      as: "theSeason"
    },
  },
])
  .toArray()
  .then((results) => {

    results.forEach((document) => {
      var theRace = document.standings;
      var theSprint = document.theSprint[0];
      if (theSprint != undefined) theSprint = theSprint.standings;
```

```
    var points_race = document.theSeason[0].racePointsSystem;
    var points_sprint = document.theSeason[0].sprintPointSystem;

    var driver;
    /*
     The drivers from the top 10 are awarded some points according to the points system.
     The driver who set the fastest lap is awarded 1 point if he is in the top 10.
    */
    for (let i = 0; i < theRace.length; i++) {
      driver = theRace[i].driver;
      if (general_standings[driver] == undefined) general_standings[driver] = points_race[i];
      else general_standings[driver] += points_race[i];

      if (driver == document.fastestDriver && i < 10) {
        if (general_standings[driver] == undefined) general_standings[driver] = 1;
        else general_standings[driver] += 1;
      }
    }

    /*
      The drivers from the top 8 are awarded some points according to the points system.
    */
    if (theSprint != undefined) {
      for (let i = 0; i < theSprint.length; i++) {
        driver = theSprint[i].driver;
        if (general_standings[driver] == undefined) general_standings[driver] = points_sprint[i];
        else general_standings[driver] += points_sprint[i];
      }
    }

  });

  const sortedObj = Object.entries(general_standings).sort(([, a], [, b]) => b - a);
  general_standings = Object.fromEntries(sortedObj);

  console.log("6.4. Quel est le classement du championnat des pilotes ?")
  console.log(">>> Le classement du championnat des pilotes est le suivant :")
  console.log(general_standings)
  console.log();
})
```

Si l'équipe Red Bull était déclassée de toutes les sessions, qui aurait gagné le championnat par équipe ?

```
var general_standings = {};
db.collection("races").aggregate([
  {
    $lookup: {
      from: "sprints",
      localField: "city",
      foreignField: "city",
      as: "theSprint"
    },
  },
  {
    $lookup: {
      from: "seasons",
      localField: "year",
      foreignField: "year",
      as: "theSeason"
    },
  },
])
  .toArray()
  .then((results) => {

    results.forEach((document) => {
```

```
    var theRace = document.standings;
    var theSprint = document.theSprint[0];
    if (theSprint != undefined) theSprint = theSprint.standings;

    var points_race = document.theSeason[0].racePointsSystem;
    var points_sprint = document.theSeason[0].sprintPointSystem;

    var driver;
    /*
     The drivers from the top 10 are awarded some points according to the points system.
     The driver who set the fastest lap is awarded 1 point if he is in the top 10.
    */
    for (let i = 0; i < theRace.length; i++) {
      driver = theRace[i].driver;
      if (general_standings[driver] == undefined) general_standings[driver] = points_race[i];
      else general_standings[driver] += points_race[i];

      if (driver == document.fastestDriver && i < 10) {
        if (general_standings[driver] == undefined) general_standings[driver] = 1;
        else general_standings[driver] += 1;
      }
    }

    /*
      The drivers from the top 8 are awarded some points according to the points system.
    */
    if (theSprint != undefined) {
      for (let i = 0; i < theSprint.length; i++) {
        driver = theSprint[i].driver;
        if (general_standings[driver] == undefined) general_standings[driver] = points_sprint[i];
        else general_standings[driver] += points_sprint[i];
      }
    }

  });

  const sortedObj = Object.entries(general_standings).sort(([, a], [, b]) => b - a);
  general_standings = Object.fromEntries(sortedObj);

  console.log("6.4. Quel est le classement du championnat des pilotes ?")
  console.log(">>> Le classement du championnat des pilotes est le suivant :")
  console.log(general_standings)
  console.log();
})
```

7. Application web
    a. Interface

Présentation de l'application Web affichant les données stockées sur MongoDB, exposées via une API express.

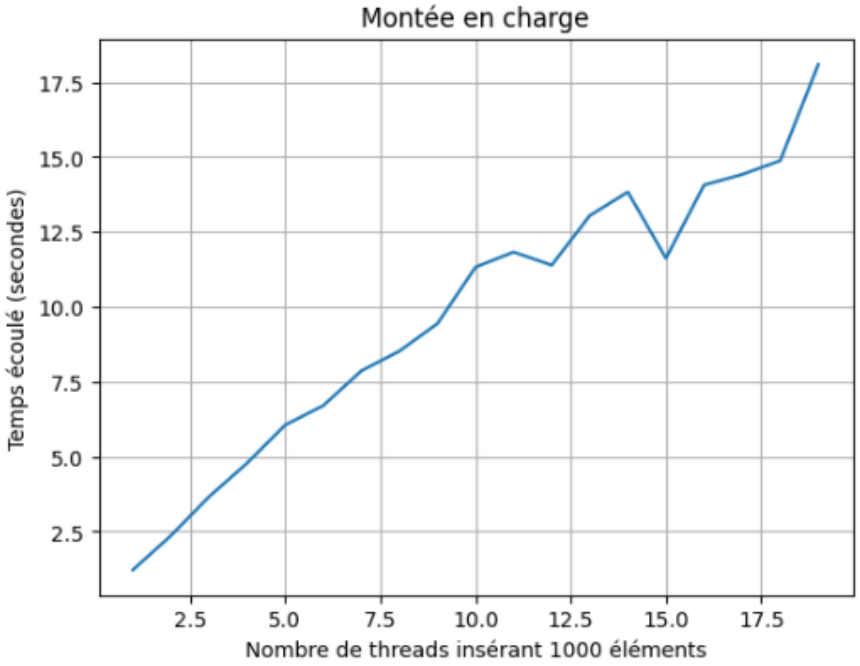
Les différentes collections sont représentées par une mosaïque sur la page d'accueil. On retrouve une section présentant les derniers temps fort en bas de page. La navbar contient un lien pour retourner à la page d'accueil. Ainsi qu'un menu déroulant regroupant les différentes pages. Il est possible d'accéder aux différentes pages en cliquant sur les cartes de la mosaïque, à travers le menu déroulant de la navbar, ou bien via les liens dans le footer.

Chaque page contient trois tabs. Le premier, Explore, affiche l'ensemble des données de la collection concernée. Le deuxième, More, permet d'accéder à des requêtes individuelles de la base de données renvoyant certaines informations spécifiques. Le troisième, Add, met à disposition un formulaire dynamique permettant d'insérer des entrées dans la base de données.
Sur la page des pilotes, il est possible d'accéder aux informations propres à chacun des pilotes à travers une carte spécialisée en cliquant sur un pilote.
Sur la page des qualifications, il est possible d'accéder au classement général de la course concernée à travers des modals. La page des sprints fonctionne de manière similaire.
Sur la page des saisons, des popovers permettent de visualiser les systèmes de points des courses et des sprints.

b. Vidéo de démonstration

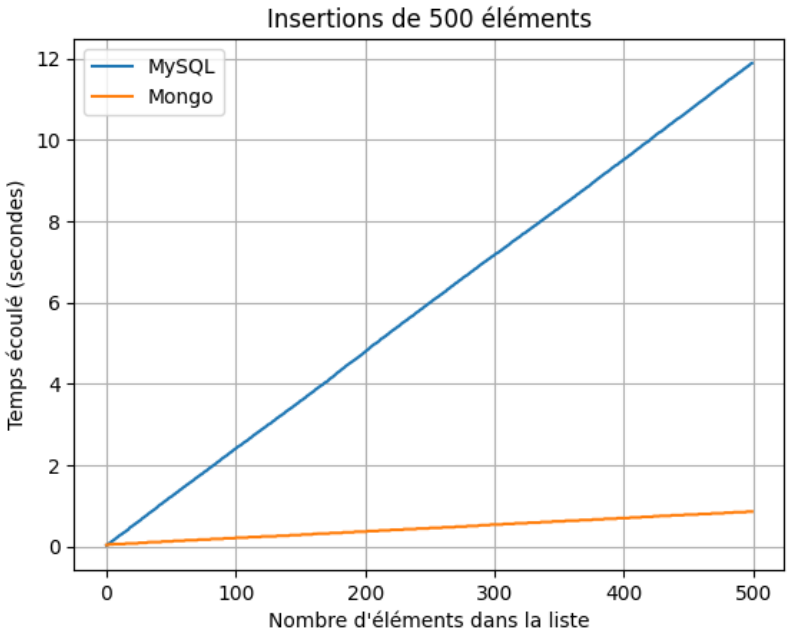
https://youtu.be/mzrhgIanFio

8. Montée en charge



9. Comparaisons avec d'autres outils
   a. MySQL

Insertions MongoDB → Temps d'exécution : 0.9598865509033203 secondes
Insertions MySQL → Temps d'exécution : 11.960264205932617 secondes



MongoDB est 12x plus rapide lors de l'insertion.

b. Mongoose

Mongoose est une bibliothèque javascript, que l'on peut intégrer à un projet avec npm. Cette bibliothèque vient greffer du code en plus pour faciliter interaction du développeur avec MongoDB depuis Javascript.
Mongoose permet notamment de définir des schémas de données, d'insérer des données et autres fonctions relatives à MongoDB. Dans tous les cas, on remarque bel et bien que le code est simplifié par rapport à "du Mongo classique". Si nous avions utilisé Mongoose directement, toutes les requêtes que nous avons réalisées auraient été plus simples à réaliser, et à comprendre.

```
mongoose
 .connect("mongodb://localhost:27017/f1_mongoose")
 .then(() => {
   console.log("Successfully connected to the database");
   return Promise.all([
     Driver.insertMany(docDrivers, { ordered: true }),
     Race.insertMany(docRaces, { ordered: true }),
   ]);
 })
 .then(() => {
   console.log("Successfully inserted data into the collections");
   return mongoose.connection.close();
 })
 .then(() => {
   console.log("Connection to the database has been closed");
 })
 .catch((error) => {
   console.error("Error occurred while connecting to the database", error);
 });
```