

A Virtual Online Betting System

Project Report (EE547 Applied and Cloud Computing)

Hongbo Zhang, Ziyang Zhou, Jiayu Guo

December 13, 2022

1 Summary

In this project we built an online platform for users to bet on the NBA games. Every new user will receive 2000 virtual bet points upon registration, which could be placed on the upcoming matches. We will requests odds from well-known online gambling sites via the odds API. After the match ends, users who bet on the winner will get points according to the odds. The betting history will be recorded to users' profile for them to check. This project contains front-end and back-end. The frontend is deployed on vercel and the backend is deployed on the Railway.app.

2 Architecture

Our web application's architecture include 4 parts:

1. User: User have a access to login or sign up an account for first time use, and have a betting balance in his account.
2. Front-End: Interact with user's operation and show the betting webpage to user. Also, send the request to backend and receive response from back End.
3. Back-End: This is our core part in our project. It will receive request from front-end and give response back to it. It also runs routinely to requests match data from the odds api, and store the fetched data to database.
4. Database: Provide user's information and provide match information to the back-end, also will record user's information and record match's information from back-end

The following sections will present more details on the technologies, implementation and deployment of the architecture.

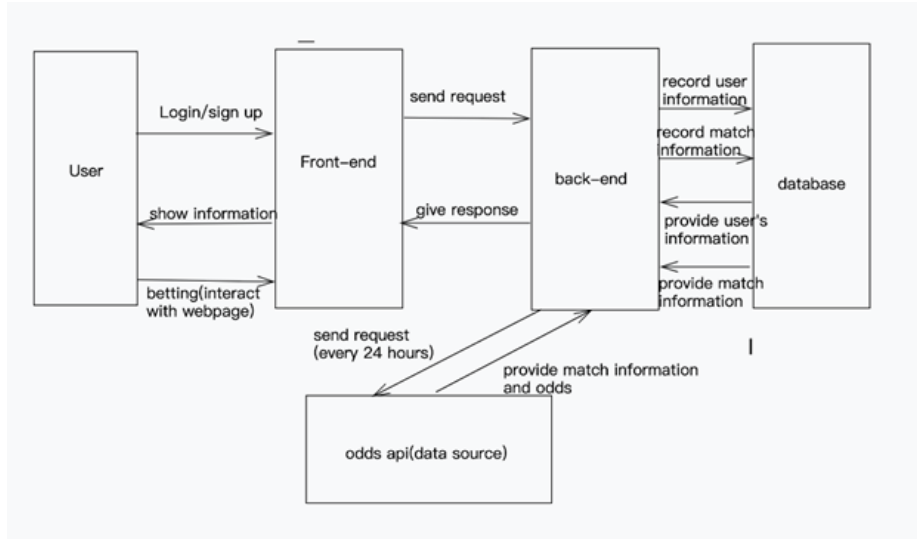


Figure 1: main architecture of the betting system

3 Technologies

3.1 Next.js and React

Next.js is a JavaScript framework that allows you to build server-rendered and statically-generated applications using React. React is a popular JavaScript library for building user interfaces. Next.js and React are commonly used together to build front-end applications because they make it easy to create fast and scalable applications with a simple and intuitive API. Next.js also provides features like automatic code splitting and optimized performance out of the box, making it a great choice for building high-performance front-end applications.

3.2 Typescript

TypeScript is a strongly typed, object-oriented, compiled programming language that builds on JavaScript. It is a superset of the JavaScript language, designed to give you better tooling at any scale.

TypeScript may be used to develop JavaScript applications for both client-side and server-side execution (as with Node.js or Deno). Multiple options are available for transpilation. The default TypeScript Compiler can be used, or the Babel compiler can be invoked to convert TypeScript to JavaScript.

3.3 Tailwind

Tailwind CSS is a utility-based low-level CSS framework designed to simplify building with speed and less focus on writing custom CSS for web applications.

3.4 Strapi and graphql

Strapi is an open-source headless content management system (CMS) that is built on top of Node.js and is used to manage content and make it available through APIs. It allows developers to easily create and manage content types and their fields, as well as define and manage relationships between different types of content.

Strapi supports multiple databases including SQLite, MongoDB, MySQL, and Postgres, and APIs such as RESTful and graphql. In our implementation, we chose the SQLite as database and

graphql as API. Additionally, Strapi is highly customizable, allowing us to extend its capabilities and integrate it with the external datasources.

4 Pages

Figure 2 below illustrates the main pages and their relationships in our project.

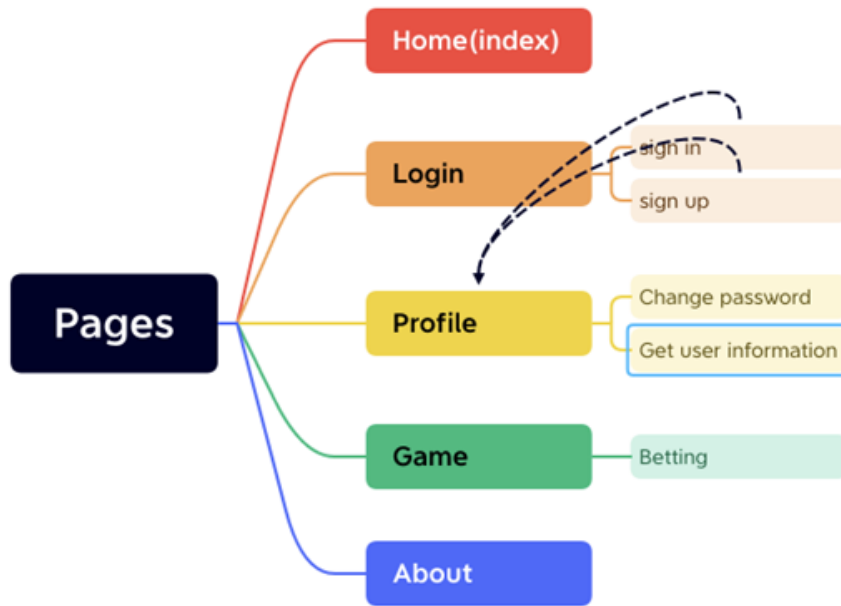




Figure 2: chart illustrating the pages and their relations


5 Implementation


5.1 Front end


We use the home page to provide some information about matches. These data are real information obtained from the NBA API every day.


[Index](#)


[Profile](#)


[About](#)


[Game](#)



HOME TEAM

Los Angeles Clippers

Portland Trail Blazers

Dallas Mavericks

Washington Wizards

Indiana Pacers

Houston Rockets

Atlanta Hawks

Philadelphia 76ers

Detroit Pistons

New York Knicks

AWAY TEAM

Boston Celtics

Minnesota Timberwolves

Oklahoma City Thunder

Brooklyn Nets

Miami Heat

Milwaukee Bucks

Chicago Bulls

Charlotte Hornets

Los Angeles Lakers

Sacramento Kings

COMMENCEMENT TIME

2023-12-13 03:40:00US

2022-12-13 03:00:00US

2022-12-13 01:40:00US

2022-12-13 00:10:00US

2022-12-13 00:10:00US

2022-12-12 00:10:00US

2022-12-11 23:40:00US

2022-12-11 23:10:00US

2022-12-11 23:10:00US

2022-12-11 23:10:00US




Figure 3: index page

Here is core code:

```

<div className="flex min-h-screen flex-col items-center py-2">
  <Head>
    <title>Game</title>
    <link rel="icon" href="/favicon.ico" />
  </Head>
  <div className="flex pt-5 w-full justify-around content-center">
    <image className="max-w-xs rounded-lg" alt="poster1" src={poster1} />
    <div className="overflow-x-auto relative shadow-md sm:rounded-lg">
      <table className="w-full text-sm text-left text-gray-500 dark:text-gray-400">
        <thead className="text-xs text-gray-700 uppercase bg-gray-50 dark:bg-gray-700 dark:text-gray-400">
          <tr>
            <th scope="col" className="py-3 px-6">
              Home team
            </th>
            <th scope="col" className="py-3 px-6">
              Away team
            </th>
            <th scope="col" className="py-3 px-6">
              Commence time
            </th>
          </tr>
        </thead>
        <tbody>

```

Figure 4: index core code

5.1.2 Login

We have 3 features related to user account handling: register, log in, and change password.

For sign in, if the username does not exist, no changes will be made to this page. If the password is wrong, there will be a corresponding prompt. The sign up page asks you to re-enter your password in case you make a mistake.

Figure 5: content definitions

The password could also be changed when user enter his profile page.

Figure 6: change password

Here is some core code:

```

</label>
<input
  type="username"
  id="username"
  className="bg-gray-50 border border-gray-300 text-gray-900 text-sm rounded-md"
  placeholder="john"
  required
/>
</div>
<div className="mb-6">
  <label
    htmlFor="password"
    className="block mb-2 text-sm font-medium text-gray-900 dark:text-white"
  >
    Password
  </label>
  <input
    type="password"
    id="password"
    className="bg-gray-50 border border-gray-300 text-gray-900 text-sm rounded-md"
    placeholder="password"
    required
  />
</div>
<div>
  <div className="mb-6">
    <label
      htmlFor="confirm_password"
      className="block mb-2 text-sm font-medium text-gray-900 dark:text-white"
    >
      Confirm password
    </label>
    <input
      type="password"
      id="confirm_password"
      className="bg-gray-50 border border-gray-300 text-gray-900 text-sm rounded-md"
      placeholder="password"
      required
    />
  </div>
  <button
    type="submit"
    className="text-white bg-blue-700 hover:bg-blue-800 focus:ring-4 focus:ring-blue-300"
  >
    Submit
  </button>
</div>

```

Figure 7: sign page HTML code

```

)).then((res) => res.json(), err => err.json())
).then((res) => {
  if (res.errors) {
    alert(JSON.stringify(res.errors))
    // console.log(isLogin)
  }
  else {
    console.log("res", res);
    const data = isLogin ? res?.data?.user?.data?.[0] : res?.data?.createUser?.data
    document.cookie = `username=${form.username.value};`;
    document.cookie = `password=${form.password.value};`;
    document.cookie = `userId=${data.id};`;
    document.cookie = `points=${data.attributes?.bet_points};`;
    if (isLogin) {
      const password = data.attributes?.password
      if (password === form.password.value) {
        router.push("/profile")
      } else {
        alert("wrong username or password")
      }
    } else {
      router.push("/profile")
    }
    //console.log(`username=${form.username.value}; password=${form.password.value};`);
  }
});

```

Figure 8: Jump to profile page

5.1.3 Profile

Once the user create a new account or successfully login, the web page will automatically jump to the profile interface. New users will have 2000 initial bet points. If the user has already logged in, the user could also click the button in the upper left corner of the page to jump to this page. The profile page have the function of logging out and changing the password. It uses graphql to query

the user information and the associate history bet records.

Log out Change password

username: zzy balance: 2000

HOME TEAM	AWAY TEAM	CHOSEN TEAM	ODDS	POINTS	COMMENCE TIME	PAYBACK
Dallas Mavericks	Milwaukee Bucks	home	1.87	1	2022-12-10 03:10:00UST	pending
Utah Jazz	Minnesota Timberwolves	home	1.87	1	2022-12-10 02:10:00UST	pending
Utah Jazz	Minnesota Timberwolves	away	2.3	5	2022-12-10 02:10:00UST	pending

Figure 9: Profile page

```

(user {
  <div className="block max-w-sm p-6 bg-white border border-gray-200 rounded-lg shadow-md hover:bg-gray-100 dark:
    <p className="font-normal text-gray-700 dark:text-gray-400">{"username: " + user.username + " balance: " + user.poin
  </div>
  <div className="overflow-x-auto relative shadow-md sm:rounded-lg">
    <table className="w-full text-sm text-left text-gray-500 dark:text-gray-400">
      <thead className="text-xs text-gray-700 uppercase bg-gray-50 dark:bg-gray-700 dark:text-gray-400">
        <tr>
          <th scope="col" className="py-3 px-6">
            Home team
          </th>
          <th scope="col" className="py-3 px-6">
            Away team
          </th>
          <th scope="col" className="py-3 px-6">
            Chosen team
          </th>
          <th scope="col" className="py-3 px-6">
            Odds
          </th>
          <th scope="col" className="py-3 px-6">
            Points
          </th>
          <th scope="col" className="py-3 px-6">
            Commence time
          </th>
          <th scope="col" className="py-3 px-6">
            Payback
          </th>
        </tr>
      </thead>
      <tbody>
        {history ? "loading" : history.map((bet) =>
          <tr className="bg-white border-b dark:bg-gray-900 dark:border-gray-700">
            <td
              className="py-4 px-6"
            >
              {bet.attributes.home}
            </td>
            <td className="py-4 px-6">{bet.attributes.away}</td>
            <td className="py-4 px-6">{bet.attributes.chosen}</td>

```

Figure 10: Profile core code

5.1.4 Game

After logging in, users can place bets on the Game tab. There are different games to choose from in the interface. After choosing a team, users can choose how much to bet on. When the bet is successful committed, there will be a message reminding the bet. The process of this part of data use graphql api to query the available bets and corresponding match information.

HOME TEAM	AWAY TEAM	COMMENCE TIME	BET
Los Angeles Clippers	Boston Celtics	2022-12-13 03:40:00UST	Bet
Portland Trail Blazers	Minnesota Timberwolves	2022-12-13 03:10:00UST	Bet
Dallas Mavericks	Oklahoma City Thunder	2022-12-13 01:40:00UST	Bet
Washington Wizards	Brooklyn Nets	2022-12-13 00:10:00UST	Bet
Indiana Pacers	Miami Heat	2022-12-13 00:10:00UST	Bet

Figure 11: available bets

TEAM	ODDS	SELECT
Los Angeles Clippers	2.4	<input checked="" type="radio"/>
Boston Celtics	1.61	<input type="radio"/>

100

Figure 12: betting page

bet-nextjs.vercel.app 显示

Bet success!

Figure 13: successful bet response

5.2 Backend and database

5.2.1 Database schema

We use the SQLite, the default database setting of strapi, to store our user and match data. There are 3 main collections in our database: available bets, users and bets. The schema is shown in the following table. It is built using strapi's content builder. Notice that every available bet (match) has a UID which relates matches to bet record. The bet record also has an one to one relation to an exact user.

Bet

Build the data architecture of your content

Configure the view

NAME	TYPE
<div>Ab</div> chosen	Text
<div>Ab</div> home	Text
<div>Ab</div> away	Text
<div>123</div> points	Number
<div></div> commence_time	Datetime
<div>123</div> price	Number
<div>123</div> payback	Number
<div></div> user_1	Relation with User?
<div>Ab</div> miD	Text

User1

Build the data architecture of your content

Configure the view

NAME	TYPE
<div>Ab</div> username	Text
<div>123</div> bet_points	Number
<div>Ab</div> password	Text

Available_bet

Build the data architecture of your content

Configure the view

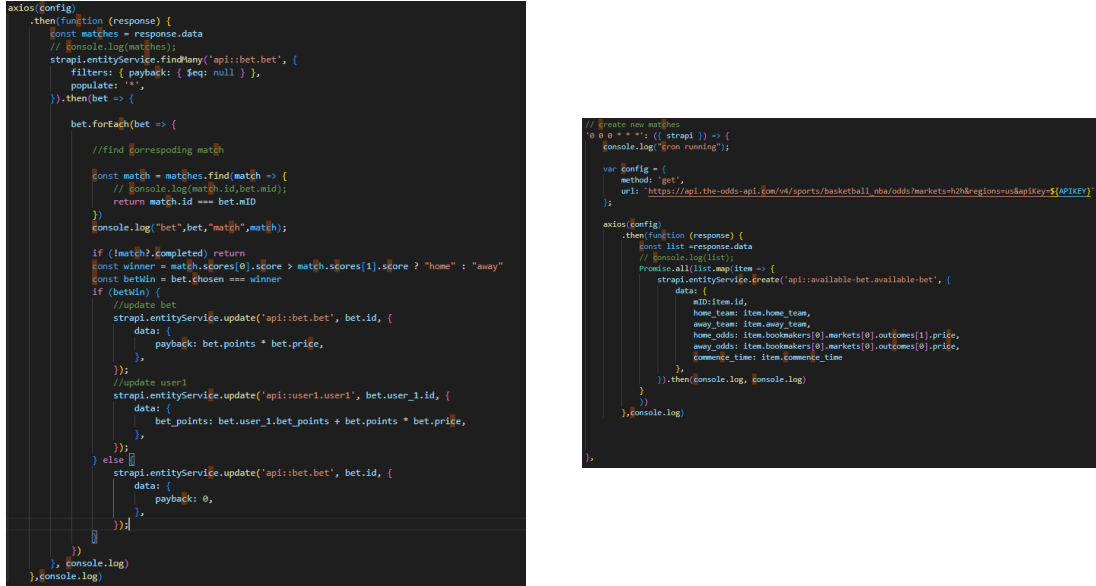
NAME	TYPE
<div>Ab</div> home_team	Text
<div>Ab</div> away_team	Text
<div>123</div> home_odds	Number
<div>123</div> away_odds	Number
<div></div> commence_time	Datetime
<div></div> miD	UID

Figure 14: sign pages

5.2.2 External API usage and data manipulation

We use axios to requests match data and odds via the odds api. Since the odds api limit the maximum requests to 500 per month in the free tier, we set the match update and bet settlement interval to 24 hours.

The requested match information includes home and away teams, commence time and the home and away odds. We requests the decimal odds. For example, 1.5 means users will get 150 points back including an initial 100 points betted. 2 cron tasks were set to handle the match creation and bet settlement. The code are shown in the figures below.



```
axios(config)
  .then(function (response) {
    const matches = response.data
    // console.log(matches);
    strapi.entityService.findMany('api::bet.bet', {
      filters: { payoffback: { $eq: null } },
      populate: '*',
    }).then(bet => {
      bet.forEach(bet => {
        // find corresponding match
        const match = matches.find(match => {
          // console.log(match.id, bet.mid);
          return match.id === bet.mid
        })
        console.log("bet", bet, "match", match);
        if (!match?.completed) return
        const winner = match.scores[0].score > match.scores[1].score ? "home" : "away"
        const betWin = bet.chosen === winner
        if (betWin) {
          // update bet
          strapi.entityService.update('api::bet.bet', bet.id, {
            data: {
              payoffback: bet.points * bet.price,
            },
          });
          // update user
          strapi.entityService.update('api::user1.user1', bet.user_1.id, {
            data: {
              bet_points: bet.user_1.bet_points + bet.points * bet.price,
            },
          });
        } else {
          strapi.entityService.update('api::bet.bet', bet.id, {
            data: {
              payoffback: 0,
            },
          });
        }
      })
    }, console.log)
  }, console.log)
```

```
// create new matches
0 0 0 * * *; ( strapi ) => {
  console.log("cron running");

  var config = {
    method: 'get',
    url: 'https://api.the-odds-api.com/v4/sports/basketball_nba/odds/markets=h2h&regions=us&apiKey=${APIKEY}',
  };

  axios(config)
    .then(function (response) {
      const list = response.data
      // console.log(list);
      Promise.all(list.map(item => {
        strapi.entityService.create('api::available-bet.available-bet', {
          data: {
            mid: item.id,
            home_team: item.home_team,
            away_team: item.away_team,
            home_odds: item.bookmakers[0].markets[0].outcomes[1].price,
            away_odds: item.bookmakers[0].markets[0].outcomes[0].price,
            commence_time: item.commence_time
          },
        })
      })).then(console.log, console.log)
    }, console.log)
  }, console.log)
```

Figure 15: code for the cron tasks

5.2.3 GraphQL API

By default Strapi create REST endpoints for each of our content-types. With the GraphQL plugin, we managed to add a GraphQL endpoint to fetch and mutate our content. Strapi uses a Shadow CRUD to automatically generates the type definitions, queries, mutations and resolvers based on the data model defined in the previous section. However, some of our data needs to be protected from the frontend for the sake of security. So I disabled some of the CRUD operations to avoid data leak or illegal mutations. The core code are shown in below figure.

```
register(strapi){
  const extensionService = strapi.plugin('graphql').service('extension');
  extensionService.shadowCRUD('api::bet.bet').disableMutations();
  extensionService.shadowCRUD('api::available-bet.available-bet').disableMutations();
  extensionService.shadowCRUD('api::user1.user1').field('bet_points').disable();
  extensionService.shadowCRUD('api::user1.user1').disableQueries();
},
```

Figure 16: Code to disable some endpoints

The main graphql schema, queries and mutations are listed in the below figures.



Figure 17: GraphQL endpoints

5.2.4 New bet process

Although strapi provides customization and overites on schema and resolvers. We didn't managed to successfully develop a working graphql resolver for the new bet records created by the users. So instead we use a special RESTful API for the newly created bets. The API code is shown below. If a bet is created before the commence time and user have enough bet points, the bet record will be created. Otherwise the backend wont edit the database and emits an error message.

6 Cloud Deployment

The cloud deployment consists of 2 parts. The frontend is deployed on vercel. The backend and the database are deployed together on the Railway App. Both 2 platforms provides auto-generated domain name, which saved us a lot of time in domain configuration.

6.1 Cloud provider and instance types

Vercel is a cloud platform for hosting websites and web applications, providing easy and fast deployment of applications built with React and Next.js.

Railway is another cloud deployment platform which provides simple and efficient deployment from local code base. Railway also offers a consumption-based pricing model, making the cost negligible. The backend and database are deployed on an elastic virtual machine which has at most 512MB RAM, 2vCPUs and 1GB disk, which is sufficient to handle large scale user bet activities and match updates. The figure 18 shows the of the compute resource usage.



Figure 18: compute resource usage

7 Problems and Difficulties

We initially planed to use the NBA api to get the match information. The NBA api could provide more abundant information including history winning rate of a certain team. However, the NBA api could not be used on the cloud server since the api author banned those ips. So we changed to requests those information from the odds api, which functions well after deployment. We also encountered some difficulties in customize the graphql resolver in the strapi framework, as mentioned in the previous section.

8 References

1. <https://nextjs.org/>
2. <https://reactjs.org/docs/handling-events.html>
3. <https://tailwindcss.com/>
4. <https://www.youtube.com/watch?v=mTz0GXj8NN0> (next.js)
5. <https://strapi.io/>
6. <https://the-odds-api.com/>
7. <https://vercel.com/docs>
8. <https://docs.railway.app/>
9. <https://axios-http.com/docs/intro>
10. <https://en.wikipedia.org/wiki/Cron>

9 Team Roles

Hongbo Zhang: Back-end and deployment

Ziyang Zhou: Front-end

Jiayu Guo: Responsible for front-end and back-end communication