공유데이

정승욱

필수 과제

1 - 1학기와 2학기의 차이

2 - Database 설계 및 SQL 작성

5 - 비동기 프로그래밍 이해

6 - node.js와 socket.io를 활용한 채팅 server/client 구현

비동기 프로그래밍 이해

Promise vs Async/await

Promise : 성공 시 then, 실패 시 catch를 통해 기능을 구현할 수 있음.
하지만 성공 시 이어지는 기능의 구현에 대해서는 계속해서 then으로
이어가야 하는 불편함이 있음.

Async/await : then과 catch를 통해 쉽게 예외처리를 할 수는 없지만,
동기식 코드를 짜는 것처럼 깔끔하게 코드를 쓸 수 있음.

상황에 따라, 선택할 필요 있음 :
성공 시에만 계속 이어지는 기능의 경우 Promise로 하면 코드가 길어질
수 있으니 Async/await로 구현하는 게 편하고,
중간 중간에 예외처리가 많이 필요한 경우 Async/await로 할 시 try/catch문의
반복이 많아질 수 있으니 Promise로 구현하는 게 좋다.

비동기 프로그래밍 이해

```
request("a").then((resolve) => {
  console.log(resolve);

  request("b").then((resolve) => {
    console.log(resolve);

    request("c").then((resolve) => {
      console.log(resolve);
```

Promise_hard_code

```
async function test() {
  const resolve_0 = await request("a");
  console.log(resolve_0);
  const resolve_1 = await request("b");
  console.log(resolve_1);
  const resolve_2 = await request("c");
  console.log(resolve_2);
```

Await_hard_code

```
array.reduce((prev, item) => {
  return prev.then(() =>
    request(item.word).then((promise) => {
      console.log(promise);
    })
  );
}, Promise.resolve());
```

Promise_soft_code

```
async function test() {
  for (const item of array) {
    const resolve = await request(item.word);

    console.log(resolve);
  }
}
```

Await_soft_code

비동기 프로그래밍 이해

```
array.forEach(async (item) => {
  request(item.word).then((resolve) => {
    console.log(resolve);
  });
});
```

Promise_all_non_sequence

```
array.forEach(async (item) => {
  const resolve = await request(item.word);

  console.log(resolve);
});
```

Await_all_non_sequence

```
array.forEach((item) => {
  const promise = request(item.word);

  promise_list.push(promise);
});

Promise.all(promise_list).then((values) => {
  values.forEach((resolve) => {
    console.log(resolve);
  });
});
```

Promise_all_sequence

```
async function test() {
  const async_fun_list = [];

  for (item of array) {
    const async_fun = request(item.word);

    async_fun_list.push(async_fun);
  }

  for (async_fun of async_fun_list) {
    const resolve = await async_fun;

    console.log(resolve);
  }
}
```

Await_all_sequence

node.js와 socket.io를 활용한 채팅 server/client 구현

Java, Spring으로만 서버를 구성 했었는데, node.js와 express.js로도 서버를 구성할 수 있음을 경험했음.

우선, 서버와 클라이언트의 언어가 Javascript로 같기 때문에 호환이 좋다는 장점이 있다. 또한 Socket.io 라이브러리를 통해 쉽게 데이터 교환이 이루어진다는 것도 큰 장점이다.

Socket.io 라이브러리를 사용하면, 서버에서 socket.emit으로 이벤트를 발생시켜 socket.on으로 같은 이름의 이벤트를 수신하여 이벤트를 처리할 수 있게 된다. 반대로 클라이언트에서 똑같이 socket.emit으로 송신한 이벤트는 서버에서 socket.on으로 수신하여 처리할 수 있다.

Socket.io 라이브러리를 통해 실시간, 양방향, 이벤트 기반 통신이 쉽게 구현될 수 있고, 이를 통해 채팅 애플리케이션을 쉽게 완성할 수 있었다.

node.js와 socket.io를 활용한 채팅 server/client 구현

```html
<body>
  <section id="before">
    <p>닉네임을 입력하세요</p>
    <input id="nickname" />
    <button id="joinBtn">들어가기</button>
  </section>
```

```js
// 이벤트 : join 클릭
$("#joinBtn").click(function (e) {
  fnNickname(e);
});
```

```js
// 송신 : 닉네임
function fnNickname(e) {
  if ($("#nickname").val().trim() == "") {
    alert("Input your nickname!");
    return false;
  }
  nickname = $("#nickname").val().trim();
  socket.emit("join", nickname); // 접속 이벤트
}
```

Emit!!

```js
// 유저 입장
socket.on("join", function (data) {
  console.log(data);
  // 이미 입장했다면 중단
  if (joinedUser) {
```

```js
    socket.broadcast.emit("join", {
      nickname: nickname
```

```js
  socket.emit("welcome", {
    nickname: nickname
```

Emit!! *2

```js
// 수신 : 신규자 접속
socket.on("join", function (data) {
  // 입장 알림
  $("#messages").append(
    $('<li class="noti">').text(data.nickname + "님이 입장하셨습니다.")
  );

  // 유저리스트 업데이트
  fnUpdateUserList(data.userList);
});
```

```js
// 수신 : 환영인사
socket.on("welcome", function (data) {
  // 유저리스트 업데이트
  fnUpdateUserList(data.userList);

  $("#before").hide();
  $("#after").show();
  $("#messages").append(
    $('<li class="noti">').text(nickname + "님 환영합니다.")
  );
});
```

감사합니다