Unit 3 16

사용자 구독 저장

UT-NodeJS / 04.28.2023

ut-nodejs.github.io

16

캡스톤 프로젝트 3

사용자 구독 저장

p. 227-234

사용자 구독 저장

이 애플리케이션을 만들기 위해 다음과 같은 단계를 거친다.

1. 데이터베이스의 설정 (p. 228)
2. 데이터 모델링 (p. 228-231)
3. 구독자 뷰와 라우터 추가 (p. 231-234)

① 데이터베이스의 설정

Listing 16.1 main.js에서 Node.js 애플리케이션의 Mongoose 설정

```
const mongoose = require("mongoose");
mongoose.connect(
  "mongodb:// localhost:27017/confetti_cuisine",
  {useNewUrlParser: true}
);
```

mongoose의 요청

데이터베이스 연결 설정

```
npm install
mongodb
mongoose

code .
```

만일 데이터베이스가 존재하지 않는다면 애플리케이션을 실행 시 자동으로 생성된다.

② 데이터 모델링

Listing 16.2 subscriber.js에서의 구독자 스키마 정의

```
const mongoose = require("mongoose"),        ← Mongoose 요청
  subscriberSchema = mongoose.Schema({
    name: String,
    email: String,
    zipCode: Number   phone: String
  });                 ← 스키마 특성 정의
```

구독자에 대한 정보 필드를 3가지로 요청했기 때문에 이 필드들을 정의하고 있는 Mongoose 스키마를 만들 것이다.

하지만 우편번호 대신 전화번호를 문자열로 저장합니다.
(MongoDB는 010으로 시작하는 번호를 허용하지 않으므로 문자열로 저장합니다.)

② 데이터 모델링

Listing 16.3 subscriber.js에서 export된 구독자 모델 생성

```
module.exports = mongoose.model("Subscriber",
  subscriberSchema);
```

← 모델의 익스포트

이제 스키마가 정의됐고 이 스키마를 사용하는 모델을 정의해야 한다.
또한, 새로운 구독자들을 데이터베이스에 저장하기 위한 리우트와 로직을 만들 것이다.

② 데이터 모델링

이제 Subscribers 모듈의 인스턴스를 생성하거나 메인 애플리케이션 파일 내에서 이 모델을 호출할 수 있게 됐다.

Listing 16.4 subscribersController.js에서 구독자를 위한 컨트롤러 액션

```
const Subscriber = require("../models/subscriber");        ← 구독자 모델 요청

① exports.getAllSubscribers = (req, res) => {              ← 구독자 가져오기
    Subscriber.find({})
      .exec()
      .then((subscribers) => {
        res.render("subscribers", {
          subscribers: subscribers
        });
      })
      .catch((error) => {
        console.log(error.message);
        return [];
      })
      .then(() => {
        console.log("promise complete");
      });
};
```

① 첫 번째 액션은 데이터베이스에서 모든 구독지를 찾고 프라미스를 돌려주기 위한 쿼리를 실행시키기 위해 find를 사용한다.

② 두 번째 액션은 다음 쿼리 체인을 이어가고 성공적으로 데이터를 받았는지, 또는 catch 에서 에러가 발견됐는지에 대한 뷰를 렌더링하기 위해 then을 사용한다.

```
② exports.getSubscriptionPage = (req, res) => {
    res.render("contact");
};                                                         ← 구독 페이지 렌더링
```

```
③ exports.saveSubscriber = (req, res) => {                 ← 구독자 저장
    let newSubscriber = new Subscriber({
      name: req.body.name,
      email: req.body.email,
      zipCode: req.body.zipCode
    });

    newSubscriber.save()
      .then( () => {
        res.render("thanks");
      })
      .catch(error => {
        res.send(error);
      });
};
```

③ 세 번째 액션은 Subscriber의 인스턴스를 생성하고 데이터베이스에 저장한다. 이 동작은 자동으로 Mongoose를 통해 프라미스를 돌려주고 체인에서 다음 기능을 실행하거나 에러를 캐치할 수 있게 한다.

③ 구독자 뷰와 라우터 추가

Listing 16.5 subscribers.ejs에서의 구독자 체크

```
<% subscribers.forEach(s => {%>
  <p><%= s.name %></p>
  <p><%= s.email %></p>
<% })%>
```

구독자 배열 체크

맞춰야 할 마지막 조각은 뷰와 방문지들이 자신들의 정보를 전송하기 위한 폼을 추가하는 것이다. subscribers.ejs 뷰는 Listing 16.5와 같이 데이터베이스 내의 모든 구독자 정보를 루프 구문으로 순환하면서 HTML 태그와 함께 보여준다.

Listing 16.6 contact.ejs에서의 새로운 구독자

```
<form action="/subscribe" method="post">
  <label for="name">Name</label>
  <input type="text" name="name" placeholder="Name">
  <label for="email">Email</label>
  <input type="email" name="email" placeholder="Email">
  <label for="zipCode">Zip Code</label>
  <input type="text" pattern="[0-9]{5}" name="zipCode" placeholder="Zip Code">
  <input type="submit" name="submit">
</form>
```

구독 폼 추가

또 필요한 폼은 구독 정보 전송 폼이며 여기서 contact.ejs 내의 폼을 대체한다.

[노트] 이제 홈 컨트롤러의 postedContactForm을 더 이상 사용하지 않는다. 오래된 라우트와 액션은 삭제할 수 있다.

③구독자 뷰와 라우터 추가

이 뷰를 출력하려면 Listing 16.7과 같이 main.js 에 몇 가지 라우트를 추가하거나 변경해야 한다. 구독 뷰를 위한 라우트를 새로 만드는 대신 /contact 라우트를 수정해 getSubscriptionPage 함수를 쓰게 했다.

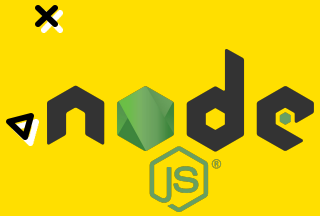
Listing 16.7 main.js에서의 구독자 추가 라우트

```
구독자 컨트롤러의 요청
const subscribersController = require(
 "./controllers/subscribersController");

                                        구독자 목록 뷰를 위한
                                        라우트 추가
app.get("/subscribers", subscribersController.getAllSubscribers);
app.get("/contact", subscribersController.getSubscriptionPage);  //
app.post("/subscribe", subscribersController.saveSubscriber);

구독 페이지 뷰를 위한
라우트 추가
                                        포스팅된 폼 데이터를
                                        위한 라우트 추가
```

④ 요약



이 프로젝트에서 대부분 정적인 부분으로만 이루어진 Express.js 애플리케이션을 데이터의 저장과 동적 표현을 위해 수정하기 시작했다.

프라미스의 도움으로 코드는 간결성을 유지하고 에러 발생에 대한 대응도 용이하게 됐다. 4부에서는 사용자 모델 구축을 통해 다른 레벨에서 어떻게 Mongoose를 사용하는지 배울 것이다. 이 모델을 통해 생성(Create), 조회(Read), 수정(Update), 삭제(Delete) 단계에서의 인증과 보안을 알아본다.

④ 실행

index.ejs

CONFETTI CUISINE

Home Courses Contact

Welcome!

Please check out all the cool new courses by our innovative and creative chefs.

If you want to see a new course, click the courses page and explore!

When you see a course you like you can contact us to join! Soon you'll be cooking in your own home.

개인 정보

④ 실행

courses.ejs

CONFETTI CUISINE

Home | Courses | Contact

Learn to cook cutting edge food

Our Courses

Event Driven Cakes

$ 50

Asynchronous Artichoke

$ 25

Object Oriented Orange Juice

$ 10

개인 정보

④ 실행

contact.ejs

localhost

CONFETTI CUISINE

Home Courses Contact

Reach out to us!

Enter your email if you're interested to learn more:

Name

Email

Zip Code

Submit

We'll get back to you soon!

④ 실행

thanks.ejs

CONFETTI CUISINE

Home Courses Contact

Thank you for submitting!

Go to home!

개인 정보

④실행

error.ejs

CONFETTI CUISINE

Home | Courses | Contact

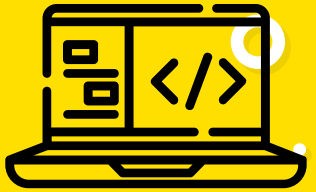
Oops! Something went wrong, or the page you are looking for is not available.

Go to home!

개인 정보

④ 실행

subscribers.ejs



Coding 과제!

캡스톤 프로젝트

사용자 구독 저장

p. 227-234

과제 타임!

한번 해 보자!~