最终项目 – 书评应用



预计所需时间: 2 小时

- 在这个最终项目中，我们将构建一个服务器端的在线书评应用，并将其与一个安全的 REST API 服务器集成，该服务器将在会话级别使用 JWT 进行身份验证。然后，您将使用 Promises 回调或 Async-Await 函数测试您的应用。

目标:

完成本实验后，您将能够:

1. 创建 API 并在 Express 服务器上执行 CRUD 操作，使用会话和 JWT 进行身份验证。
2. 在 Node.js 中使用 Axios 的 Async/Await 或 Promises。
3. 创建 REST API 端点并使用 Postman 测试它们。

设置：创建应用程序

1. 通过编辑器中的菜单打开终端窗口：终端 > 新建终端。
2. 如果您还不在项目文件夹中，请切换到您的项目文件夹。

   ```
   cd /home/project
   ```

3. 请复制包含本实验所需起始代码的 Git 仓库：

   ```
   https://github.com/ibm-developer-skills-network/expressBookReviews.git
   ```

4. 克隆你的分叉 Git 仓库，如果它尚不存在的话。

   ```
   [ ! -d 'expressBookReviews' ] && git clone https://github.com/<your Github username>/expressBookReviews.git
   ```

5. 切换到 **expressBookReviews/final_project/** 目录以开始实验。

   ```
   cd expressBookReviews/final_project/
   ```

6. 列出该目录的内容以查看本实验的工件。

   ```
   ls
   ```

理解服务器应用

1. 在文件资源管理器中打开 **expressBookReviews/final_project/** 文件夹，并查看 `index.js` 中的以下代码：

   ```
   const express = require('express');
   const jwt = require('jsonwebtoken');
   const session = require('express-session')
   const customer_routes = require('./router/auth_users.js').authenticated;
   const genl_routes = require('./router/general.js').general;
   const app = express();
   app.use(express.json());
   app.use("/customer",session({secret:"fingerprint_customer",resave: true, saveUninitialized: true}))
   app.use("/customer/auth/*", function auth(req,res,next){
   //Write the authenication mechanism here
   });

   const PORT =5000;
   app.use("/customer", customer_routes);
   app.use("/", genl_routes);
   ```

```
app.listen(PORT,()=>console.log("Server is running"));
```

2. 本实验所需的包在 `packages.json` 中定义为依赖项，如下所示：

```
  "dependencies": {
    "express": "^4.18.1",
    "express-session": "^1.17.3",
    "jsonwebtoken": "^8.5.1",
    "nodemon": "^2.0.19"
}
```

理解用户路由

导航到 `router` 目录，包含以下 3 个文件：

1. `booksdb.js` – 该文件包含本应用程序的预加载书籍信息。
2. `general.js` – 该文件包含普通用户可以访问的路由的基本实现。
3. `auth_users.js` – 该文件包含授权用户可以访问的路由的基本实现。

更新认证机制的代码：

- 导航到 `index.js`，并在 `app.use("/customer/auth/*", function auth(req,res,next){` 下更新认证代码：

提示：使用会话授权功能（在实践项目实验中实现）根据访问令牌对用户进行身份验证。

更新和测试 `general.js` 中的通用用户路由

注意：

- 请记得在完成每个任务后，或如果您没有在一个会话中完成实验室，请将您的工作推送到 GitHub 仓库。您可以在第 14 任务中找到将仓库推送到 GitHub 的步骤。这将帮助您避免丢失进度。
- 如果您在使用 Postman 时遇到困难，您可以选择使用 `curl` 命令来测试 API 端点，如 模块 3：动手实验：使用 Node.js 和 Express 的 CRUD 操作 中所示。请在提交的同行评审中提供显示您的 `curl` 命令 JSON 输出的截图。

任务 1：

- 完成代码以获取商店中可用书籍的列表，位置在 `public_users.get('/',function (req, res) {`。

提示：使用 JSON.stringify 方法以整齐的方式显示输出。

- 运行 `npm install` 安装所需模块并启动服务器。
- 在 Postman 中测试输出。
- 请截取相同内容的屏幕截图，并将其保存为 `1-getallbooks.png`，以便在同行评审作业的任务 1 中提交。

任务 2：

- 完成代码以根据 ISBN 获取书籍详细信息，位置在 `public_users.get('/isbn/:isbn',function (req, res) {`。

提示：从请求参数中检索 ISBN

- 在 Postman 中测试输出。
- 请截取相同内容的屏幕截图，并将其保存为 `2-gedetailsISBN.png`，以便在同行评审作业的任务 2 中提交。

任务 3：

- 完成代码以根据作者获取书籍详细信息，位置在 `public_users.get('/author/:author',function (req, res) {`。

提示：
1. 获取‘books’对象的所有键。
2. 遍历‘books’数组并检查作者是否与请求参数中提供的匹配。

- 在 Postman 中测试输出。
- 请截取相同内容的屏幕截图，并将其保存为 `3-getbooksbyauthor.png`，以便在同行评审作业的任务 3 中提交。

任务 4：

- 完成代码以根据标题获取书籍详细信息，位置在 `public_users.get('/title/:title',function (req, res) {`。

提示：这将类似于练习 3

- 在 Postman 中测试输出。
- 请截取相同内容的屏幕截图，并将其保存为 `4-getbooksbytitle.png`，以便在同行评审作业的任务 4 中提交。

任务 5：

- 完成代码以获取书籍评论，位置在 `public_users.get('/review/:isbn',function (req, res) {`。

提示：根据请求参数中提供的 ISBN 获取书籍评论。

- 请截取相同内容的屏幕截图，并将其保存为 `5-getbookreview.png`，以便在同行评审作业的任务 5 中提交。

任务 6：

- 完成代码以注册新用户

提示：代码应从请求的主体中获取用于注册的‘username’和‘password’。如果用户名已存在，必须提及相同内容，并且还必须显示其他错误，例如当未提供用户名和/或密码时。

- 在 Postman 中测试输出。
- 请截取相同内容的屏幕截图，并将其保存为 `6-register.png`，以便在同行评审作业的任务 6 中提交。

更新并测试 `auth_users.js` 中的认证用户路由。

任务 7：

- 完成作为注册用户登录的代码。

提示：代码必须根据第 6 题中创建的用户名和密码验证并登录客户。它还必须将用户凭据作为 JWT 保存到会话中。由于您需要作为客户登录，在 Postman 中测试输出时，使用端点 "customer/login"

- 在 Postman 中测试输出。
- 请截取该屏幕截图，并将其保存为 `7-login.png`，以便在同伴评审作业的任务 7 中提交。

任务 8：

- 完成添加或修改书评的代码。

提示：您必须将评论作为请求查询提供，并且必须与存储在会话中的用户名一起发布。如果同一用户在相同 ISBN 上发布不同的评论，则应修改现有评论。如果另一个用户登录并在相同 ISBN 上发布评论，则将其作为不同的评论添加到相同 ISBN 下。

- 在 Postman 中测试输出。
- 请截取该屏幕截图，并将其保存为 `8-reviewadded.png`，以便在同伴评审作业的任务 8 中提交。

任务 9：

完成在 `regd_users.delete("/auth/review/:isbn", (req, res) => {` 下删除书评的代码。

提示：根据会话用户名过滤并删除评论，以便用户只能删除自己的评论，而无法删除其他用户的评论。

- 在 Postman 中测试输出。
- 请截取该屏幕截图，并将其保存为 `9-deletereview.png`，以便在同伴评审作业的任务 9 中提交。

通过这些，您已实现并测试了通用和认证用户路由的代码。

使用 Promises 或 Async-Await 改进任务 1-4 的范围

您现在将使用 Promise 回调或 Async-Await 函数来实现我们在任务 1-4 中同步覆盖的相同功能。

> **注意:** 请截取您使用 async/await 或 Promises 实现任务 10-13 的代码的屏幕截图。该屏幕截图将用于同行评审提交。

任务 10：

- 添加获取商店中可用书籍列表的代码（在任务 1 中完成），使用 Promise 回调或 async-await 和 Axios。

提示：请参考此关于 Promises 和回调的实验。

- 请确保 `general.js` 文件中包含使用 Promise 回调或 async-await 和 Axios 获取商店中可用书籍列表的代码。
- 请截取相同内容的屏幕截图，并将其保存为 task10.png，以便在同行评审作业的任务 10 下提交。

任务 11：

- 添加根据 ISBN 获取书籍详细信息的代码（在任务 2 中完成），使用 Promise 回调或 async-await 和 Axios。

提示：请参考此关于 Promises 和回调的实验。

- 请确保 `general.js` 文件中包含使用 Promise 回调或 async-await 和 Axios 根据 ISBN 获取书籍详细信息的代码。
- 请截取相同内容的屏幕截图，并将其保存为 task11.png，以便在同行评审作业的任务 11 下提交。

任务 12：

- 添加根据作者获取书籍详细信息的代码（在任务 3 中完成），使用 Promise 回调或 async-await 和 Axios。

提示：请参考此关于 Promises 和回调的实验。

- 请确保 `general.js` 文件中包含使用 Promise 回调或 async-await 和 Axios 根据作者获取书籍详细信息的代码。
- 请截取相同内容的屏幕截图，并将其保存为 task12.png，以便在同行评审作业的任务 12 下提交。

任务 13：

- 添加根据标题获取书籍详细信息的代码（在任务 4 中完成），使用 Promise 回调或 async-await 和 Axios。

提示：请参考此关于 Promises 和回调的实验。

- 请确保 `general.js` 文件中包含使用 Promise 回调或 async-await 和 Axios 根据标题获取书籍详细信息的代码。
- 请截取相同内容的屏幕截图，并将其保存为 task13.png，以便在同行评审作业的任务 13 下提交。

Github 仓库更新以提交同行评审

任务 14:

- 请将所有更改提交并推送到您分叉的 Github 仓库。

  注意： 如果您在提交和推送更改到您的仓库时需要任何帮助，请参考此实验。

- 您的 Github 仓库链接将用于同行评审作业中任务 14 的评估。

恭喜您！您已完成最终项目！

摘要：

在本实验中，您构建了一个服务器端的在线书评应用程序，将其与使用基于 JWT 的会话级身份验证的安全 REST API 集成，并使用 Promises 回调或 Async-Await 函数测试了构建的应用程序。

作者

Lavanya T S

Sapthashree K S

K Sundararajan

© IBM Corporation. 保留所有权利。