مهندسی اینترنت

تمرین شماره‌ی ۱

مدرس: **دکتر خامس‌پناه**

طراحان: علی اخگری، علی کرامتی

مهلت تحویل: دوشنبه ۷ اسفند، ساعت ۲۳:۵۹

MizDooni

مقدمه

سیستمی که در طول درس به توسعه آن خواهید پرداخت، یک سیستم رزرو میز از رستوران‌ها می‌باشد. در این سیستم کاربران می‌توانند لیستی از رستوران‌های موجود در شهرهای مختلف را مشاهده کنند، به فیلتر کردن رستوران‌ها بر اساس ویژگی‌های آنها بپردازند، میز مورد نظر خود را رزرو کنند و در نهایت نظرات خود را در مورد آن رستوران بدهند.

هدف تمرین

هدف این تمرین کسب آشنایی بیشتر با ابزارهای Git، Maven و Unit Testing است که در تمامی تمرین‌های بعدی به آن‌ها نیاز خواهید داشت. همچنین شما در این تمرین بخشی از منطق دامنه پروژه اصلی را نیز پیاده‌سازی می‌کنید. توجه داشته باشید که در این تمرین نیازی به پیاده‌سازی مفاهیم وب ندارید و در تمرین‌های بعدی به این مفاهیم پرداخته خواهد شد.

گام‌های پیاده‌سازی:

ایجاد پروژه Maven[1]

ابتدا یک پروژه‌ی Maven بسازید و با ساختار ایجاد شده توسط آن آشنا شوید. همچنین، فایل pom.xml و اطلاعات داخل آن را مشاهده کنید. پیشنهاد ما برای انجام پروژه‌های این درس، استفاده از ابزار IntelliJ IDEA می‌باشد. برای ایجاد پروژه Maven در محیط توسعه IntelliJ IDEA، می‌توانید از این لینک استفاده کنید.

اضافه کردن وابستگی‌های مورد نیاز

داده‌های مورد نیاز برای انجام این تمرین، در قالب JSON به شما داده می‌شود. برای خواندن و تجزیه JSON و تبدیل آن به فرمت مورد نیاز از پکیج‌های مخصوص کار با JSON در جاوا استفاده کنید. در اینترنت جستجو کرده و بهترین پکیج موجود را به وابستگی‌های پروژه خود اضافه کنید.

پیاده‌سازی منطق برنامه

در این تمرین نیاز است تا تعدادی از عملیات‌های پایه‌ای برنامه خود را پیاده‌سازی کنید. این عملیات در قالب دستوراتی در **خط فرمان**، به برنامه شما داده خواهند شد. برای سهولت پیاده‌سازی نیازی نیست که داده‌ای را در جایی ذخیره کنید و تمامی موجودیت‌های برنامه خود را در حافظه اصلی نگه داشته و متناسب با آن‌ها به دستورات پاسخ دهید. هر دستور، شکل کلی زیر را دارد:

```
command <JSON Data>
```

[1] برای اطلاعات بیشتر در مورد Maven می‌توانید به لینک‌های زیر مراجعه کنید.

Maven – Maven in 5 Minutes (apache.org)

Maven – Maven Getting Started Guide (apache.org)

command نشان‌دهنده نام دستور و JSONData داده‌ی مربوط به آن دستور به شکل serialize[2] شده و در قالب JSON است. برای استفاده از این داده، باید با استفاده از کتابخانه‌ای که در قسمت قبل به پروژه اضافه کردید، آن را deserialize کنید. همچنین، قابل ذکر است که JSONData برای همه دستورات وارد نمی‌شود.

ورودی‌ها به صورت JSON به شما داده می‌شوند که در هر بخش نمونه آن‌ها آمده است. پاسخ‌ها نیز باید به صورت JSON در خط فرمان چاپ شوند. فرمت پاسخ‌ها به دو حالت زیر می‌باشند:

```
{“success”: true, “data”: <ResponseData>}
```

```
{“success”: false, “data”: <ErrorMessage>}
```

دستورات منطق برنامه

در ادامه، دستوراتی که باید در این تمرین پیاده‌سازی کنید آمده است.

۱. اضافه کردن کاربر

این دستور یک کاربر را به لیست کاربران موجود در سامانه اضافه می‌کند. اطلاعات کاربر شامل نقش کاربر (role)، نام کاربری (username)، رمز عبور (password)، ایمیل (email) و آدرس (address) است.

- نقش کاربر فقط می‌تواند دو حالت client و manager باشد. در صورتی که نقش ورودی غیر از این دو حالت باشد، خطای مناسب برگردانده شود.
- نام کاربری و ایمیل برای هر کاربر یکتا می‌باشد. اگر کاربری با username یا email مشابه در سامانه موجود بود، خطای مناسب برگردانده شود.

[2] serialization - What is deserialize and serialize in JSON? - Stack Overflow

- نام کاربری نمی‌تواند شامل کاراکترهایی نظیر فاصله، نیم‌فاصله و یا کاراکترهای دیگر نظیر ! @ # و ... باشد.
- ساختار درست ایمیل به صورت روبه‌رو می‌باشد: abcdefg@example.com
- آدرس باید شامل country و city باشد.

نمونه دستور:

```
addUser {“role”: “client”, “username”: “user1”, “password”: “1234”, “email”:
“user1@gmail.com”, “address”: {“country”: “Iran”, “city”: “Tehran”}}
```

خروجی در حالت عدم بروز خطا:

```
{“success”: true, “data”: “User added successfully.”}
```

۲. اضافه کردن رستوران

این دستور یک رستوران را به مجموعه رستوران‌ها اضافه می‌کند. اطلاعات رستوران شامل نام (name)، نام کاربری مدیر (managerUserame)، نوع (type)، ساعت شروع (startTime) و پایان (endTime) کار، توضیحات (description) و آدرس (address) رستوران می‌باشد.

- نام رستوران یکتا می‌باشد. اگر رستورانی با نام مشابه در سامانه موجود بود، خطای مناسب برگردانده شود.
- در صورتی که نام کاربری مدیر رستوران موجود نباشد، باید خطای مناسب برگردانده شود. توجه داشته باشید که کاربری که به عنوان مدیر رستوران تعیین می‌شود باید role مدیر (manager) داشته باشد.
- ساعت شروع و پایان کار رستوران را ساعت‌های رند (00:00, 01:00, ... , 23:00) در نظر بگیرید. در غیر این‌صورت خطای مناسب برگردانده شود.
- آدرس باید شامل city، country و street باشد.

نمونه دستور:

```
addRestaurant {“name”: “restaurant1”, “managerUsername”: “user2”, “type”: “Iranian”, “startTime”: “08:00”, “endTime”: “23:00”, “description”: “Open seven days a week”, “address”: {“country”: “Iran”, “city”: “Tehran”, “street”: “North Kargar”}}
```

خروجی در حالت عدم بروز خطا:

```
{“success”: true, “data”: “Restaurant added successfully.}
```

۳. اضافه کردن میز

این دستور یک میز را به لیست میزهای موجود در یک رستوران اضافه می‌کند. اطلاعات میز شامل شماره میز (tableNumber)، نام رستوران متعلقه (restaurantName)، نام کاربری مدیر رستوران (managerUseranme) و تعداد صندلی‌ها (seatsNumber) است.

- شماره میز در هر رستوران یکتا می‌باشد. اگر میز با شماره مشابه در آن رستوران موجود بود، خطای مناسب برگردانده شود.
- در صورتی که نام کاربری مدیر رستوران موجود نباشد، باید خطای مناسب برگردانده شود. توجه داشته باشید که کاربری که به عنوان مدیر رستوران می‌خواهد به رستوران خود میز اضافه کند، باید role مدیر داشته باشد.
- برای رزرو هر میز، یک بازه زمانی یک ساعته لحاظ شود که بازه‌های آن از ساعت شروع و پایان کار رستوران مشخص می‌شوند.
- در صورت عدم وجود نام یک رستوران، خطای مناسب برگردانده شود.
- تعداد صندلی‌ها (seatsNumber)، باید عدد طبیعی باشد. در غیر این صورت، خطای مناسب برگردانده شود.

نمونه دستور:

```
addTable {“tableNumber”: 1, “restaurantName”: “restaurant1”, “managerUseranme”: “user2”, “seatsNumber”: 4}
```

خروجی در حالت عدم بروز خطا:

```
{“success”: true, “data”: “Table added successfully.}
```

۰۴ رزرو میز

با این دستور، کاربر می‌تواند با انتخاب یک میز از رستوران مورد نظر در یک زمان مشخص، آن میز را برای آن ساعت رزرو کند. اطلاعات رزرو شامل شما نام کاربری (userName)، نام رستوران (restaurantName)، شماره میز (tableNumber) و زمان رزرو (datetime) می‌باشد.

- در صورتی که نام کاربری موجود نباشد، باید خطای مناسب برگردانده شود. توجه داشته باشید که کاربر با نقش (role) مدیر نمی‌تواند اقدام به رزرو نماید.
- زمان انتخابی کاربر باید ساعت رند باشد. در غیر این‌صورت خطای مناسب برگردانده شود.
- فرمت datetime به صورت yyyy-MM-dd HH:mm می‌باشد.
- در صورت عدم وجود نام یک رستوران، خطای مناسب برگردانده شود.
- در صورت عدم وجود شماره میز در یک رستوران، خطای مناسب برگردانده شود.
- در صورتی که ساعت انتخابی از قبل رزرو شده باشد، خطای مناسب برگردانده شود.
- در صورتی که datetime از زمان فعلی عقب‌تر باشد، خطای مناسب برگردانده شود.
- در صورتی که زمان انتخابی کاربر خارج از بازه کاری رستوران باشد، باید خطای مناسب برگردانده شود.

نمونه دستور:

```
reserveTable {“username”: “user1”, “restaurantName”: “restaurant1”, “tableNumber”: 1,
“datetime”: “2024-02-14 21:00”}
```

خروجی در حالت عدم بروز خطا:

```
{“success”: true, “data”: {“reservationNumber”: 123}
```

(قسمت data از خروجی این دستور، یک شناسه رزرو می‌باشد که برای هر کاربر، یکتا است)

۵. لغو رزرو میز

کاربر می‌تواند با وارد کردن شناسه رزرو، رزرو خود را لغو کند.

- در صورتی که شناسه رزرو برای کاربر موجود نباشد، خطای مناسب برگردانده شود.
- در صورتی که زمان رزرو از زمان فعلی عقب‌تر باشد، امکان لغو رزرو وجود ندارد و باید خطای مناسب برگردانده شود.

نمونه دستور:

```
cancelReservation {“username”: “user1”, “reservationNumber”: 123}
```

خروجی در حالت عدم بروز خطا:

```
{“success”: true, “data”: “Reservation cancelled successfully.}
```

۶. مشاهده تاریخچه رزروهای کاربر

با این دستور، کاربر می‌تواند تاریخچه رزروهای خود را مشاهده کند. توجه داشته باشید که رزروهای کنسل شده توسط کاربر نیازی نیست نمایش داده شود.

نمونه دستور:

```
showReservationHistory { “username”: “user1”}
```

نمونه خروجی:

```
{“success”: true, “data”: {“reservationHistory”: [{“reservationNumber”: 123,
“restaurantName”: “restaurant1”, “tableNumber”: 1, “datetime”: “2024-02-14 21:00”}]}
```

۷. جستجوی رستوران بر اساس نام

با این دستور، کاربر می‌تواند رستوران‌ها را بر اساس نام آن‌ها جستجو کند.

- در صورت عدم وجود نام یک رستوران، خطای مناسب برگردانده شود.

نمونه دستور:

```
searchRestaurantsByName {“name”: “restaurant1”}
```

نمونه خروجی در حالت عدم بروز خطا:

```
{“success”: true, “data”: {“restaurants”: [{“name”: “restaurant1”, “type”: “Iranian”,
“startTime”: “08:00”, “endTime”: “23:00”, “description”: “Open seven days a week”,
“address”: {“country”: “Iran”, “city”: “Tehran”, “street”: “North Kargar”}]}
```

۸. جستجوی رستوران بر اساس نوع

با این دستور، کاربر می‌تواند رستوران‌ها را بر اساس نوع آن‌ها جستجو کند.

```
searchRestaurantsByType {“type”: “Iranian”}
```

نمونه خروجی:

```
{“success”: true, “data”: {“restaurants”: [{“name”: “restaurant1”, “type”: “Iranian”,
“startTime”: “08:00”, “endTime”: “23:00”, “description”: “Open seven days a week”,
“address”: {“country”: “Iran”, “city”: “Tehran”, “street”: “North Kargar”}]}
```

۹. مشاهده میزهای یک رستوران به همراه ساعت‌های خالی

با این دستور، کاربر می‌تواند میزهای یک رستوران مشخص را به همراه ساعت‌هایی که آن میز رزرو نشده است، مشاهده کند.

- در صورت عدم وجود رستوران، خطای مناسب برگردانده شود.

نمونه دستور:

```
showAvailableTables {“restaurantName”: “retaurant1”}
```

نمونه خروجی در حالت عدم بروز خطا:

```
{“success”: true, “data”: {“availableTables”: [{“tableNumber”: 1, “seatsNumber”: 4,
“availableTimes”: [“2024-02-14 08:00”, “2024-02-14 09:00”, “2024-02-15 18:00”,
“2024-02-15 22:00”}, {“tableNumber”: 2, “seatsNumber”: 8, “availableTimes”: [“2024-02-14
```

```
09:00", "2024-02-14 12:00", "2024-02-15 18:00", "2024-02-15 23:00"}]}
```

۱۰. اضافه کردن بازخورد

با این دستور، کاربر می‌تواند بازخورد خود از یک رستوران را ثبت کند. هر بازخورد متشکل از شش بخش می‌باشد:

۱- کیفیت غذا: یک عدد اعشاری بین صفر تا پنج (Food)

۲- سرویس دهی: یک عدد اعشاری بین صفر تا پنج (Service)

۳- محیط رستوران: یک عدد اعشاری بین صفر تا پنج (Ambiance)

۴- مجموع: یک عدد اعشاری بین صفر تا پنج (Overall)

۵- کامنت: یک متن

۶- تاریخ: تاریخ و ساعتی که کاربر نظر خود را ثبت می‌کند.

- در این فاز فرض کنید که کاربر می‌تواند به همه رستوران‌ها بازخورد دهد.
- در صورتی که نام کاربری موجود نباشد، باید خطای مناسب برگردانده شود. توجه داشته باشید که کاربر با نقش (role) مدیر نمی‌تواند اقدام به بازخورد دادن نماید.
- در صورت عدم وجود نام یک رستوران، خطای مناسب برگردانده شود.
- بخش‌های مختلف امتیاز دهی باید صحت سنجی شوند و در صورتی که نوع یا بازه آن‌ها خارج از محدوده تعریف شده باشد، باید خطای مناسب برگردانده شود.

نمونه دستور:

```
addReview {"username": "user1", "restaurantName": "retaurant1", "foodRate": 4.5,
"serviceRate": 3, "ambianceRate": 4.5, "overallRate": 4, "comment": "Not bad!"}
```

خروجی در حالت عدم بروز خطا:

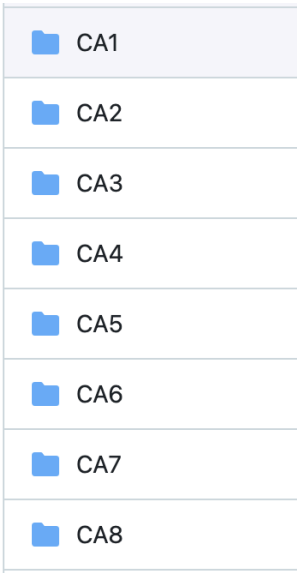
```
{“success”: true, “data”: “Review added successfully.}
```

آزمون واحد

در این قسمت باید با استفاده از چارچوب JUnit برای سناریوهای مختلف رزرو میز و اضافه کردن بازخورد، آزمون واحد بنویسید. آزمون‌های شما باید ساختار مناسب Setup، Test و Teardown را رعایت کنند.

افزودن پروژه به گیت‌هاب

ابتدا در سایت گیت‌هاب عضو شوید و یک مخزن خصوصی ایجاد کنید. سپس کاربر IE-S03 را به پروژه خود اضافه کنید. تمامی تغییرات خود را به گیت اضافه کنید و در نهایت، در مخزن خود بارگذاری کنید. ساختار مخزنی که می‌سازید به این صورت باشد که یک مخزن کلی برای فازهای مختلف پروژه ایجاد کنید (به نام MizDooni) و برای هر فاز یک دایرکتوری جداگانه اختصاص دهید. به طور مثال:



Software Engineering Best Practices

این بخش از پروژه به معرفی برخی از best practiceها در حوزه مهندسی نرم‌افزار می‌پردازد. در هر فاز از پروژه‌های این درس، اصولی بر اساس ماهیت پروژه بیان می‌شوند که رعایت آن‌ها حائز اهمیت می‌باشد.

Git Commit[3]

کامیت‌ها بخشی جدایی ناپذیر از سیستم‌های ورژن کنترل مانند Git هستند و نقش مهمی در توسعه پروژه‌های نرم‌افزاری دارند. به عبارتی کامیت‌ها snapshotهایی از repository شما در زمان‌های خاص هستند که بر اساس یک واحد تغییر منطقی در کد می‌باشند. با گذشت زمان، کامیت‌ها باید داستانی از تاریخچه repository شما و نحوه تبدیل آن به ورژن فعلی را بیان کنند. در ادامه به دو ویژگی مهم یک کامیت خوب می‌پردازیم:

- **هر کامیت باید Atomic[4] باشد.** به این معنی که هر کامیت باید نماینده یک مجموعه تغییر باشد که کمترین سایز ممکن را دارد. هر کامیت یک کار ساده و فقط یک کار ساده انجام می‌دهد که می‌تواند در یک جمله ساده خلاصه شود. توجه داشته باشید که مقدار تغییر کد مهم نیست. می تواند یک حرف یا صد هزار خط باشد، اما شما باید بتوانید تغییر را با یک جمله کوتاه ساده توصیف کنید. همچنین این تغییر باید کامل نیز باشد.

  As small as possible, but complete: this is an atomic git commit

- **هر کامیت باید message معناداری داشته باشد.** commit messageها راهی برای ارتباط بین اعضای تیم هستند و نوشتن commit message معنادار می‌تواند در زمان پاسخگویی به بسیاری از "چرا؟"ها صرفه‌جویی کند. فرض کنید یک اشکال در برنامه وجود دارد که قبلا وجود نداشت؛ برای اینکه بفهمید چه چیزی باعث این مشکل شده است، خواندن commit messageها می‌تواند بسیار مفید باشد. راه‌های زیادی برای نوشتن یک commit message خوب وجود دارد. **در طول پروژه‌های این درس از این استاندارد برای نوشتن commit message های خود استفاده کنید.**

---

[3] در ارزشیابی این پروژه، این قسمت ۲۰ نمره از ۱۰۰ نمره را شامل می‌شود.

[4] Reference

نکات پایانی

- این تمرین در گروه‌های حداکثر دو نفره انجام می‌شود و کافی است که یکی از اعضای گروه Hash مربوط به آخرین کامیت پروژه را در سایت درس آپلود کند. در هنگام تحویل، پروژه روی این کامیت مورد ارزیابی قرار می‌گیرد.
- حتما کاربر IE-S03 را به پروژه خود اضافه کنید.
- ساختار صحیح و تمیزی کد برنامه، بخشی از نمره‌ی این فاز پروژه‌ی شما خواهد بود. بنابراین در طراحی ساختار برنامه دقت به خرج دهید.
- هدف این تمرین یادگیری شماست. لطفاً تمرین را خودتان انجام دهید. در صورت مشاهده‌ی مشابهت بین کدهای دو گروه، از نمره هر دو گروه مطابق سیاستی که در کلاس گفته شده است کسر خواهد شد.
- سوالات خود را تا حد ممکن در گروه درس مطرح کنید تا سایر دانشجویان نیز از پاسخ آنها بهره‌مند شوند. در صورتی که قصد مطرح کردن سوال خاص‌تری داشتید، از طریق ایمیل با طراحان این تمرین ارتباط برقرار کنید.