2019-6-18

JAVA EE 第四次作业

得分点	状态
Webflux	完成
Functional handler	完成
Reactive persist	完成
Security	完成

1 基于 Spring Webflux 的 Restful API

1.1 获取当前所有账户

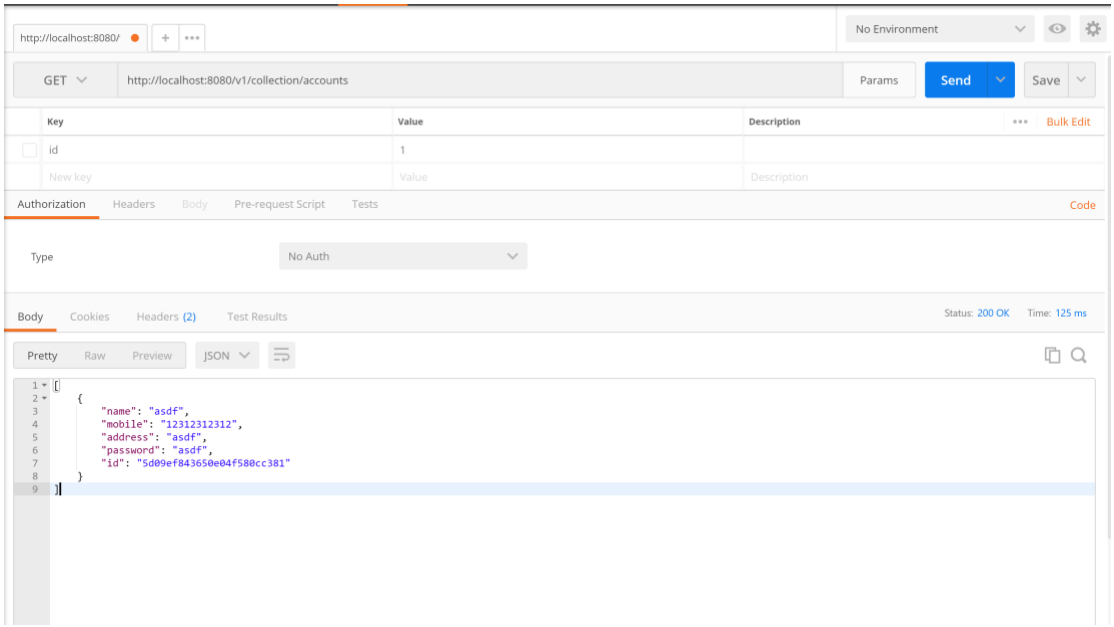
HTTP METHOD:

GET

URI:

/v1/collection/accounts

Demo:



1.2　增加一个账户

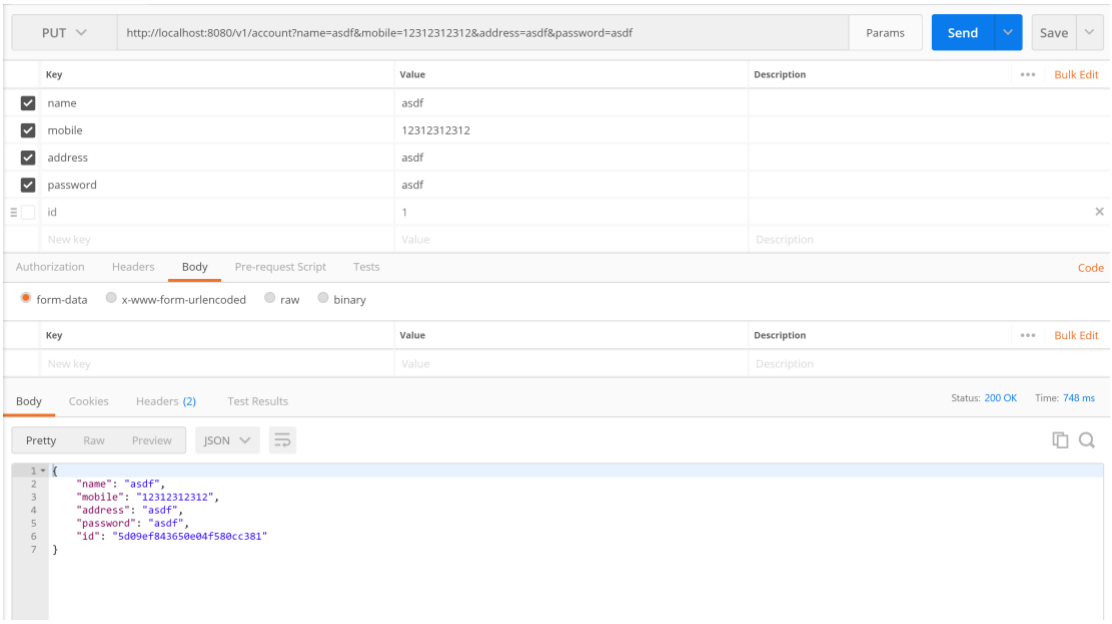
HTTP METHOD:

PUT

URI:

/v1/account

Demo:



1.3　根据用户名删除一个账户

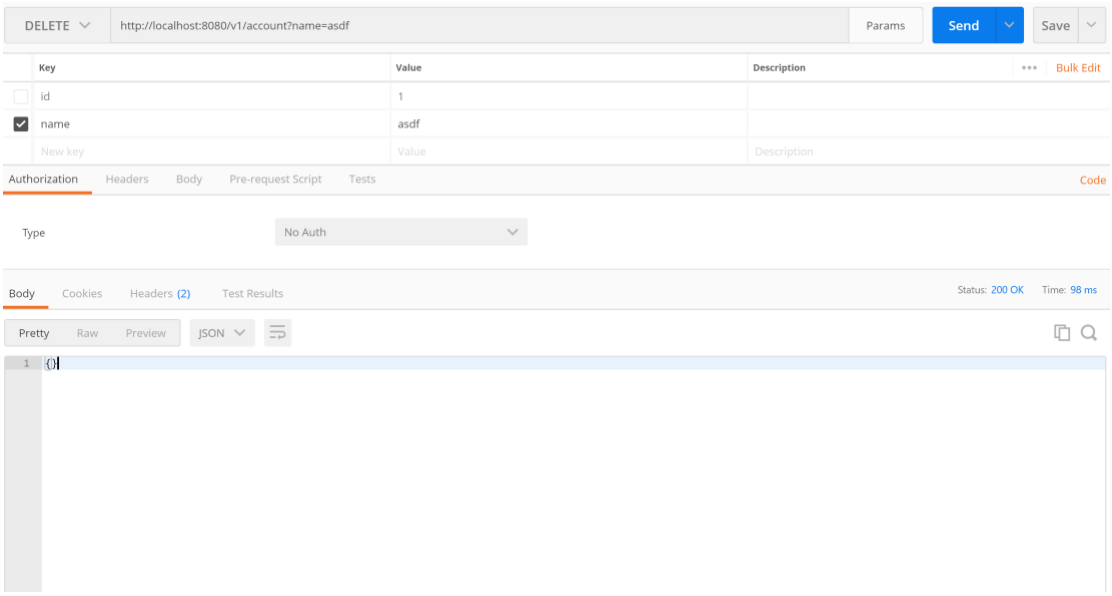
HTTP METHOD:

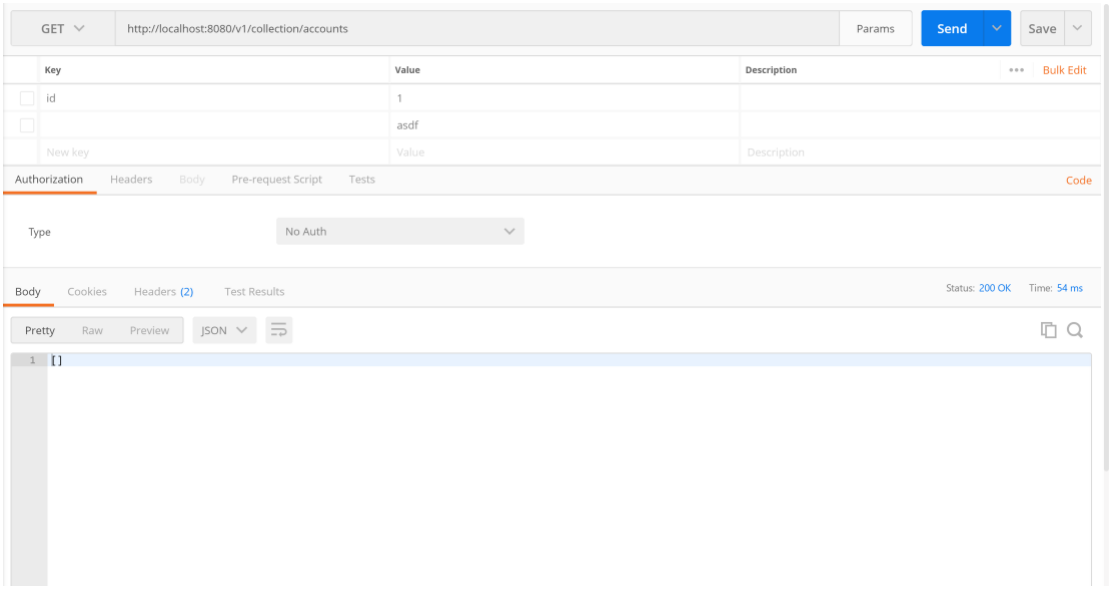
DELETE

URI:

/v1/account

Demo:





1.4　根据用户名寻找一个账户

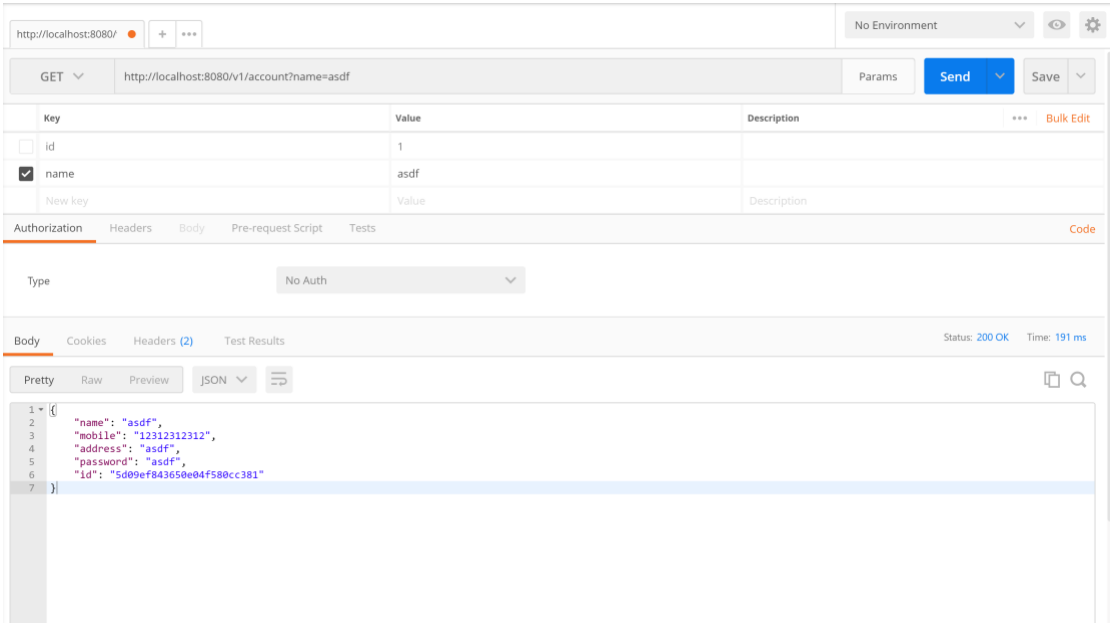
HTTP METHOD:

GET

URI:

/v1/account

Demo:



1.5　获取当前所有教练

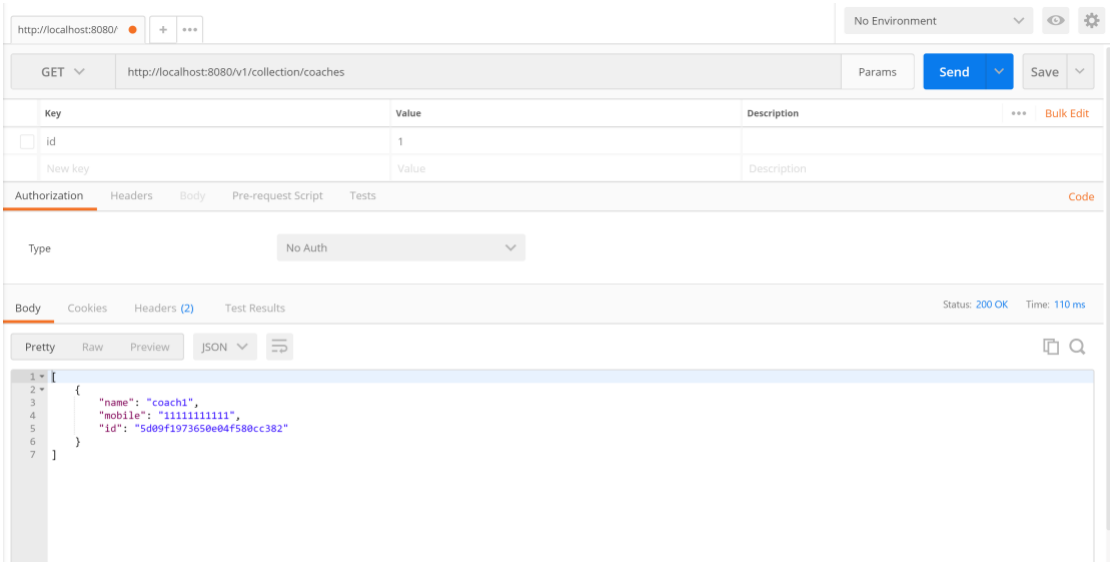
HTTP METHOD:

GET

URI:

/v1/collection/coaches

Demo:



1.6　增加一个教练

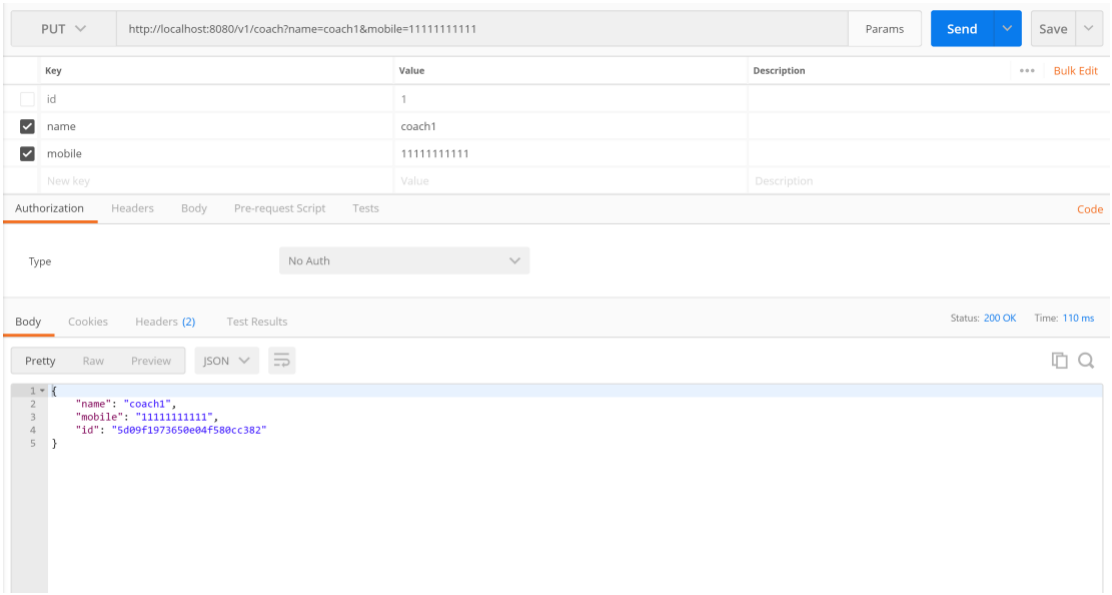
HTTP METHOD:

PUT

URI:

/v1/coach

Demo:



1.7　根据名字删除一个教练

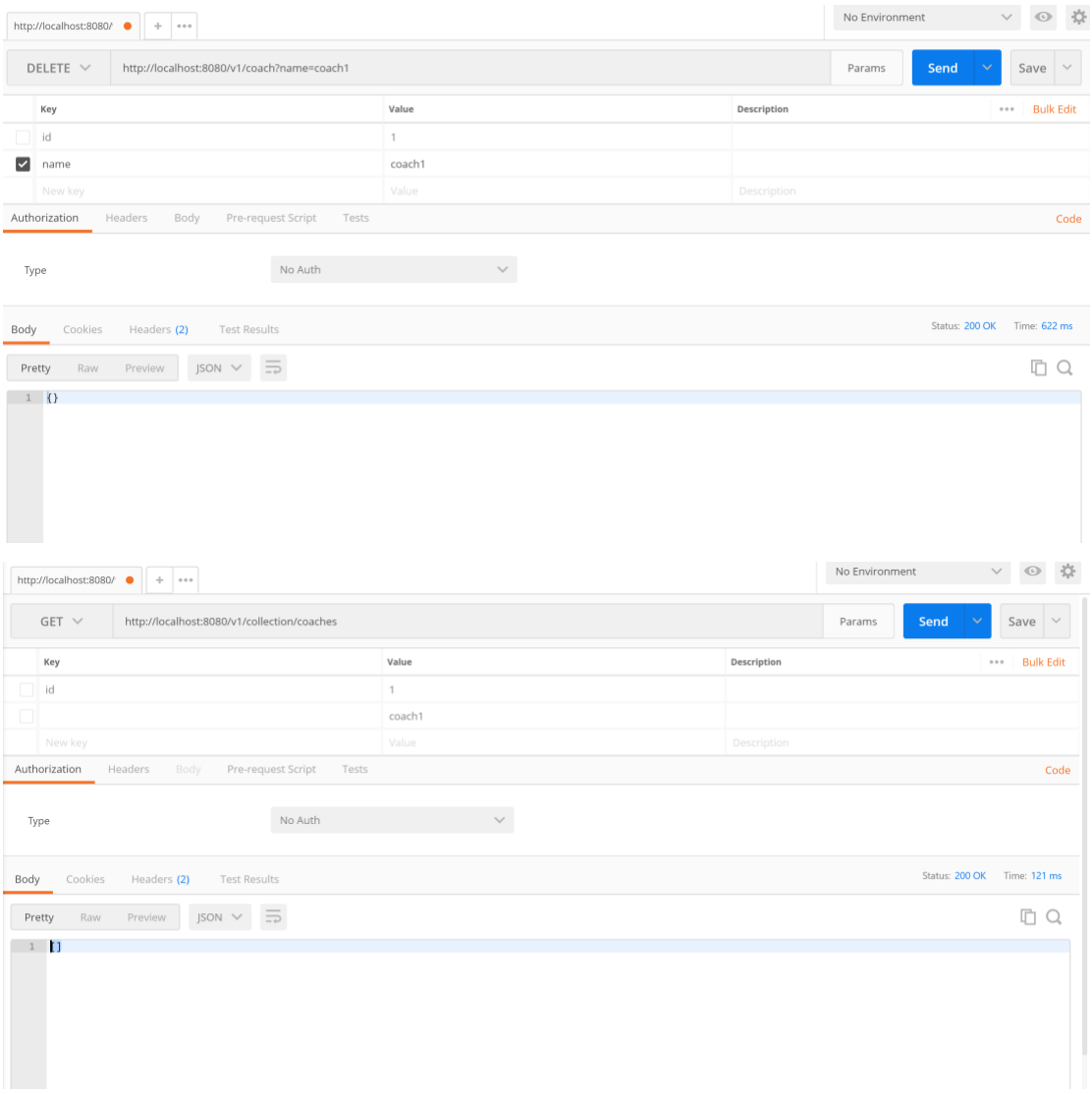
HTTP METHOD:

DELETE

URI:

/v1/coach

Demo:



1.8　根据名字寻找一个教练

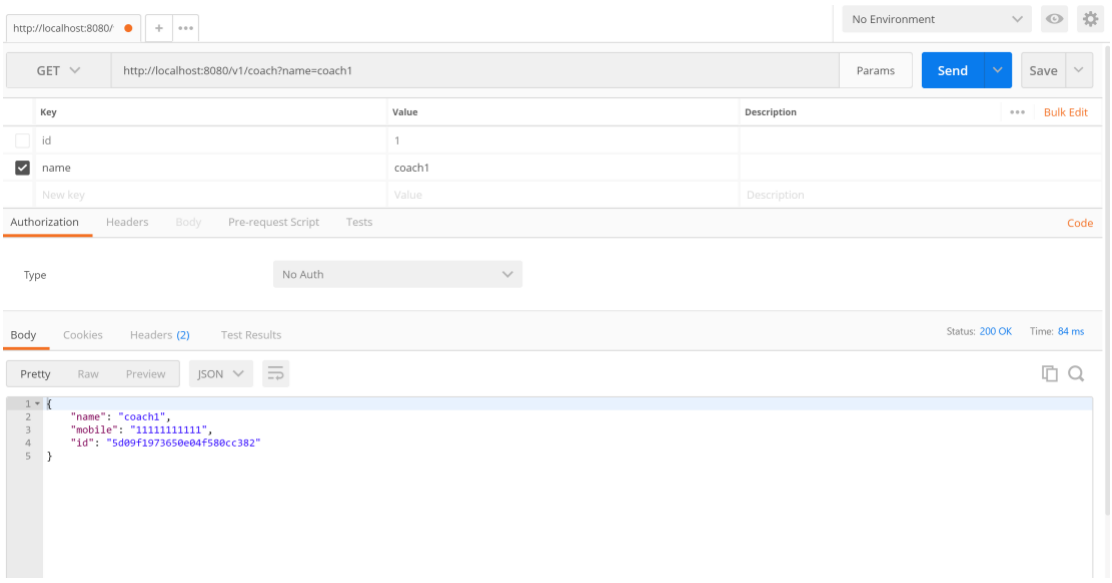
HTTP METHOD:

GET

URI:

/v1/coach

Demo:



2 基于 Handler 与 Route 的函数式编程

本项目主要使用了 Spring Webflux，并利用了函数式编程技巧。在 Routes 中，将特定的 Route 与对应的 Handler 完成绑定，处理相应的请求。

如图所示，为 CoachHandler 中的 findByName 方法，该方法负责处理 HTTP Method 为 GET，URI 为/v1/coach 的请求，将会根据传过来的 name 参数，寻找到相应的用户信息并发布。首先，在绑定 Handler 与 Route 时，已经进行了函数式编程，其次，该图中的返回结果，也采用了函数式编程进行处理。

```
public Mono<ServerResponse> findByName(ServerRequest request) {
    String name = request.queryParam( name: "name").orElse( other: "unknown");
    Mono<ServerResponse> notFound= ServerResponse.notFound().build();
    return coachRepository.findByName(name).flatMap(coach -> ServerResponse.ok()
            .contentType(MediaType.APPLICATION_JSON)
            .body(fromObject(coach))
            .switchIfEmpty(notFound));
}
```

3 基于 MongoDB 的 Reactive 数据持久化

本项目使用了 MongoDB，同时进行了 Reactive 类型的数据持久化操作，从图中可以看出，Reactive 操作主要依赖于框架 spring-boot-starter-data-mongodb-reactive。图中使用了 EnableReactiveMongoRepositories 注解，来启动框架功能。

```
import ...

@SpringBootApplication
@EnableReactiveMongoRepositories
@EnableWebFluxSecurity
public class Application {

    public static void main(String[] args) { SpringApplication.run(Application.class, args); }
}
```

于此同时，已有的 Repositories 继承了 ReactiveMongoRepository，并通过该方式完成持久化层的 Reactive 化。

```
import ...

public interface AccountRepository extends ReactiveMongoRepository<Account, String> {
    Mono<Account> findByName(String name);
    Mono<Long> deleteByName(String name);
}
```

4 基于 Spring Security 的访问控制

我们使用了 Spring Security 框架来对项目进行访问控制，避免非法用户登入并使用关键 API（如 PUT、POST、DELETE 等对数据造成影响的 API）。

我们通过 Configuration 注解，应用对 Security 框架的配置，然后细化 UserDetailsRepository，最终完成了项目的权限控制，保证了 API 的安全性。

```java
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.core.userdetails.MapUserDetailsRepository;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsRepository;

import static org.springframework.security.core.userdetails.User.withUsername;

@Configuration
public class SecurityConfiguration {

    @Bean
    UserDetailsRepository userDetailsRepository() {
        UserDetails tom = withUsername("admin").password("admin").roles("ADMIN").build();
        return new MapUserDetailsRepository(tom);
    }
}
```