第三章作业-2020211502-王小龙

思考题

1.

a.

Web系统架构安全性主要包括以下几方面：

- 认证（Authentication）：系统如何正确分辨出操作用户的真实身份？
- 授权（Authorization）：系统如何控制一个用户该看到哪些数据、能操作哪些功能？
- 凭证（Credential）：系统如何保证它与用户之间的承诺是双方当时真实意图的体现，是准确、完整且不可抵赖的？
- 保密（Confidentiality）：系统如何保证敏感数据无法被包括系统管理员在内的内外部人员所窃取、滥用？
- 传输（Transport Security）：系统如何保证通过网络传输的信息无法被第三方窃听、篡改和冒充？
- 验证（Verification）：系统如何确保提交到每项服务中的数据是合乎规则的，不会对系统稳定性、数据一致性、正确性产生风险？

b.

OIDC：在OAuth2的基础上额外加一个JWT来传递用户信息。功能全面强大，是目前很流行的SSO方案。

c.

- 中心化存储：转移到中间件如Redis中去。利用Redis 极高的并发处理能力，也可以做到弹性横行扩容。不过可能会带来中间件高可用性维护难的问题，通过租赁云服务商的托管中间件是降低中间件单点故障（SPOF） 的一种方式；
- 会话复制（Session replication）：让各个节点之间采用复制式的Session，每一个节点中的Session变动都会发送到组播地址的其他服务器上，这样某个节点崩溃了，不会中断该节点用户的服务。但Session之间组播复制的同步代价高昂，节点越多时，同步成本越高。

- 会话粘滞（Sticky session）：通过负载均衡算法如Nginx的 IP Hash 算法将来自同一IP的请求转发至同一服务。每个服务节点都不重复地保存着一部分用户的状态，如果这个服务崩溃了，里面的用户状态便完全丢失。

d.

1.由于JWT的Payload并未做过多限制，所以很容易产生滥用的问题，并且带来很多误解。 比如下面的一些问题：

- 误把JWT当作Cookie-Session使用（把JWT当作引用令牌使用），会带来未知的隐患。遵循不重复造轮子和“创新”的指导原则，尽可能不要这么做；
- 认为JWT更安全。虽然JWT采用了一定的加密算法签名，使其具备了抗篡改的能力。但其Payload大部分都只是采用 `base64UrlEncode` 编码，数据并不是加密的。攻击者可以通过 会话劫持（Session hijacking） 技术拿到JWT会话信息，之后通过 会话重放攻击（Session Replay Attack） 获取用户资源，所以最佳实践是通过启用TLS/SSL来加密通信信道。
- 把JWT存储到浏览器的Local Storage中。此方式很容易受到 XSS 攻击导致JWT泄漏。可通过服务端启用 内容安全策略（CSP） 来防御这种攻击。
- 采用对称加密方式签名（Signature）。对称加密密钥一旦泄漏，会让整个服务的基础设施遭受安全威胁。JWT支持非对称加密算法，只有签名的服务需要私钥，其他验证JWT信息的服务只需要使用公钥即可。
- 不校验JWT的签名算法。这篇 Critical vulnerabilities in JSON Web Token libraries 文章提到JWT的一种漏洞，通过 `none` 算法规避令牌验证。所以最好每次都验证JWT header中的签名算法是否是期望的。

2.JWT还有以下一些Cookie-Session没有的问题：

- 令牌难以主动失效：JWT中虽然有 `exp` 、 `nbf` 与 `iat` 这些和时间相关的属性，但很难在令牌到期之前让令牌失效，比如很难在用户退出登录时立刻让签发的令牌全部失效。虽然可能通过一些“黑名单”的技术解决这个问题，不过相比Cookie-Session来说，引入了一定的复杂性；
- 令牌数据老旧：很难把签发的令牌全部更新成最新的数据。比如把用户的权限信息（Role）放在JWT Payload中，当用户的角色发生变化时，很难把之前签发的令牌信息更新成最新的数据；
- 令牌存储：存储在客户端意味着有多种选择：Cookie？Local Storage？如果放在Cookie中，为了安全，一般会给Cookie设置 `http-only` 和 `secure` 的属性。但这也会带来一定的不便性，比如客户端要读取JWT Payload的内容只能借助服务端API接口。如果将JWT存储至浏览器Local Storage，虽然方便了客户端读取，但可能会带来XSS攻击的威胁，又需要去设置CSP来防御这种威胁；
- 令牌大小：JWT相比Cookie-Session还是大不少，尤其是要在Payload中存储一些额外的权限信息。一般服务端都有对HTTP Header的大小限制；

- 网络开销：更大的文本意味着更高的网络开销，进一步会需要更复杂的基础设施，也会产生复杂的运维问题等；
- 难以统计：服务端无状态意味着很难做诸如统计用户在线数量的功能；

e.

刷新令牌是用于在访问令牌过期时，获取新的访问令牌。这样可以避免用户需要重新授权，提高了用户体验。
当访问令牌过期时，客户端可以使用刷新令牌来获取新的访问令牌。刷新令牌通常比访问令牌长寿命，因此可以在访问令牌过期之后使用。

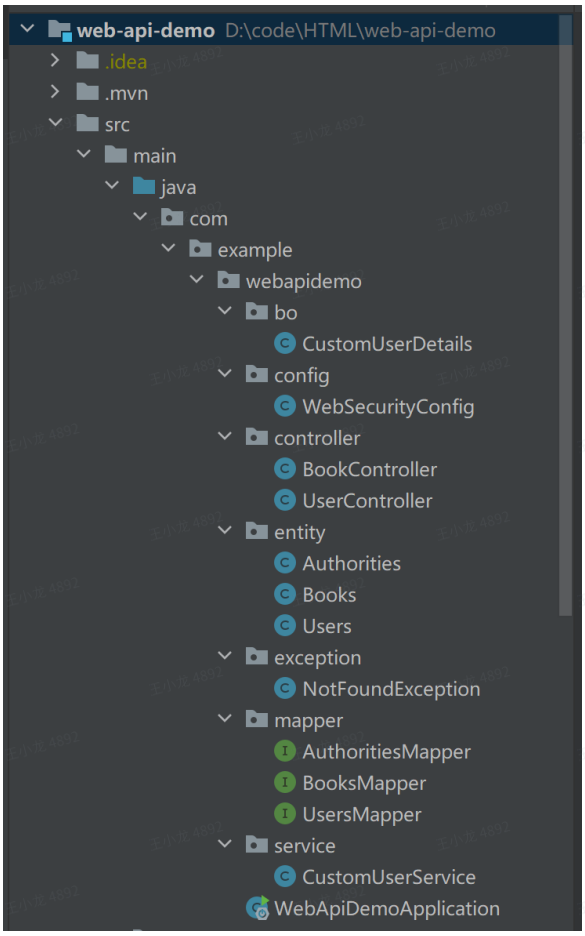
实验题

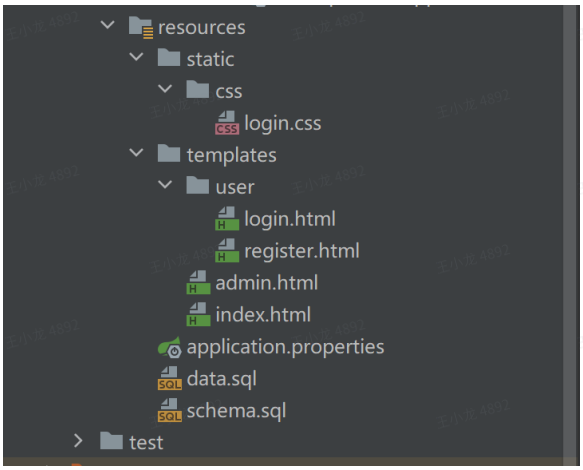
项目仓库地址如下：

（已添加老师为成员）

https://gitee.com/lhfhlhfl/web-api-demo.git

项目的文件如下：

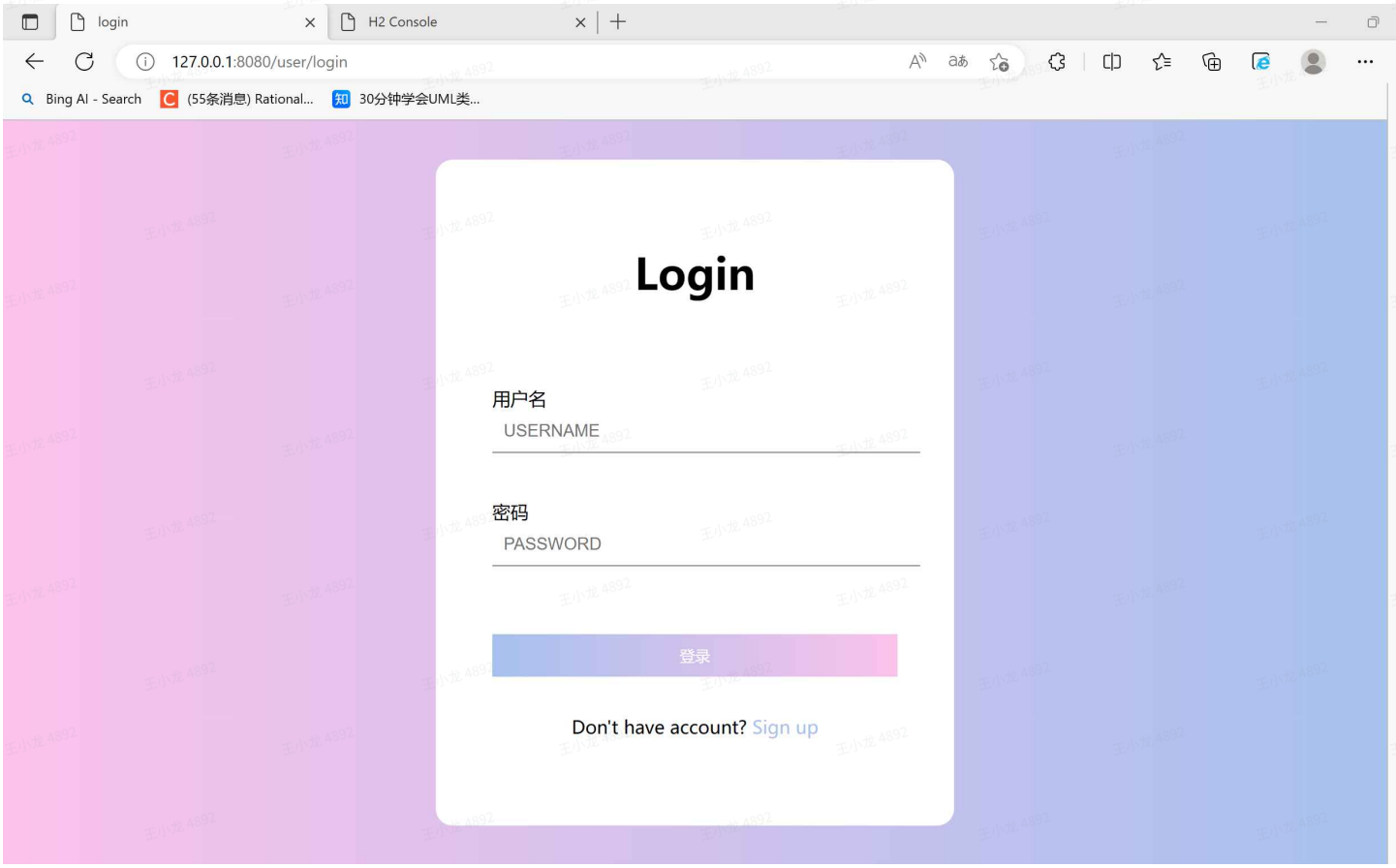




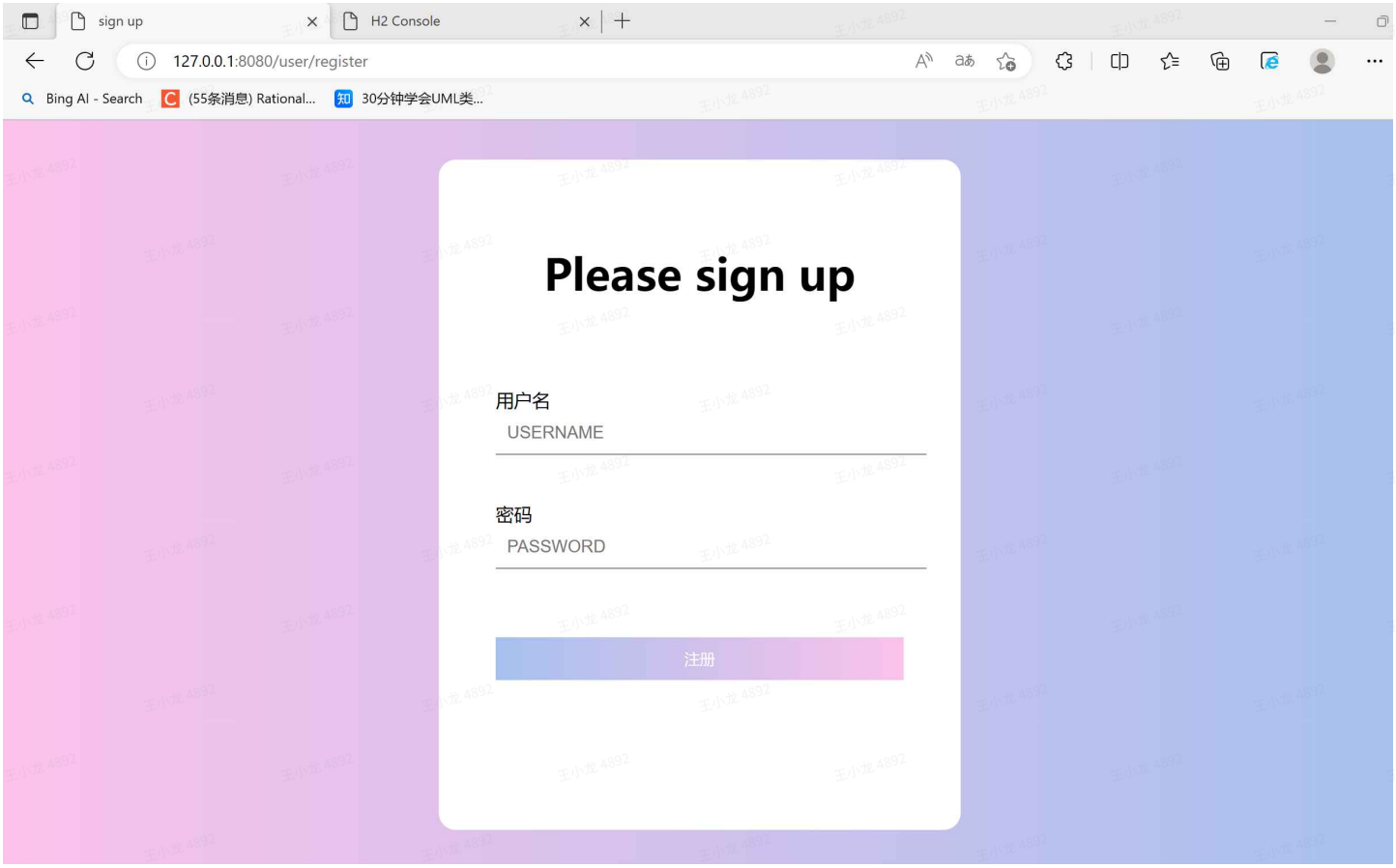
本次实验题，在上次作业的基础上，结合老师的演示项目，实现了引入Spring Security框架，进行用户身份认证和访问授权。

演示如下：

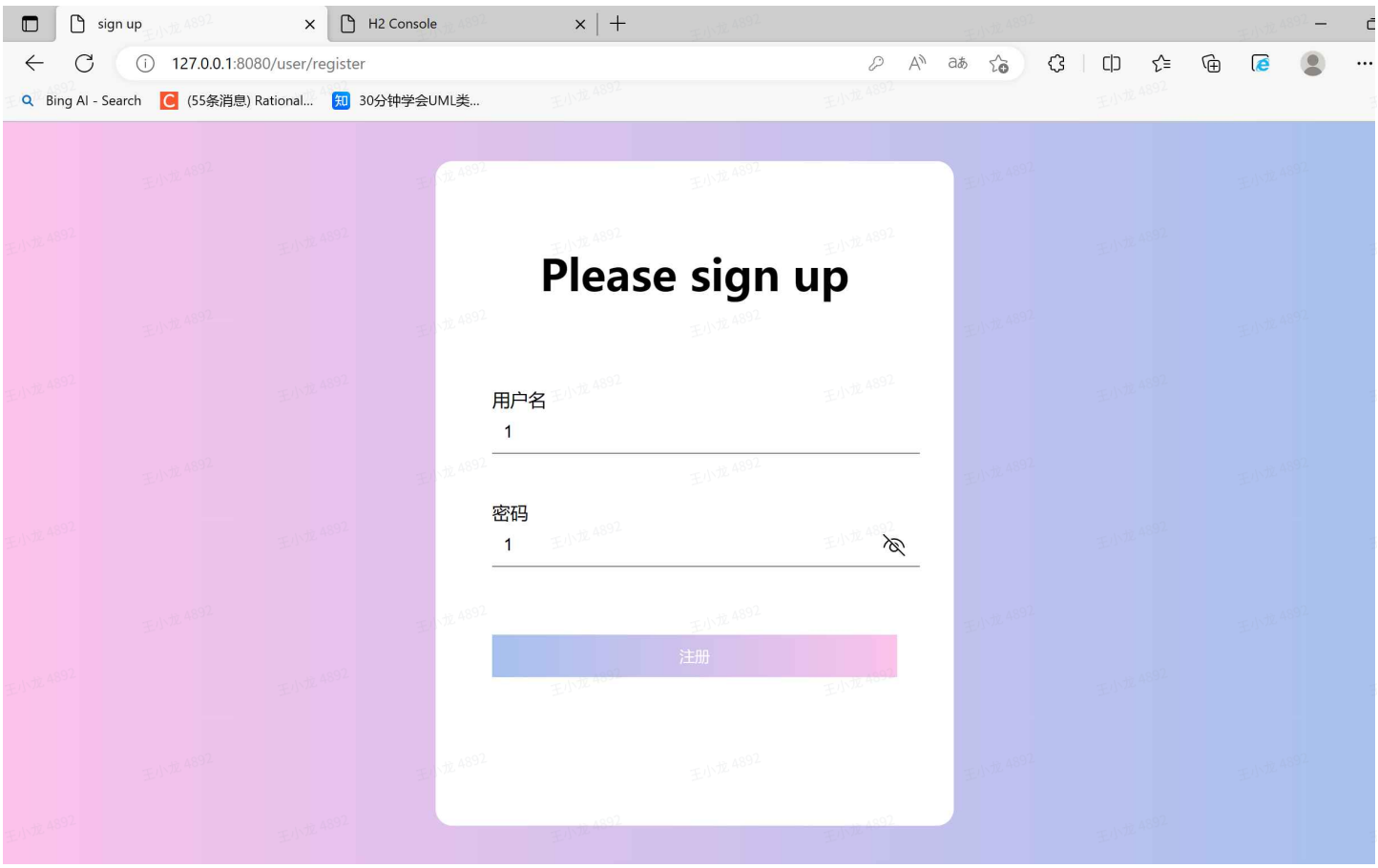
登录界面：



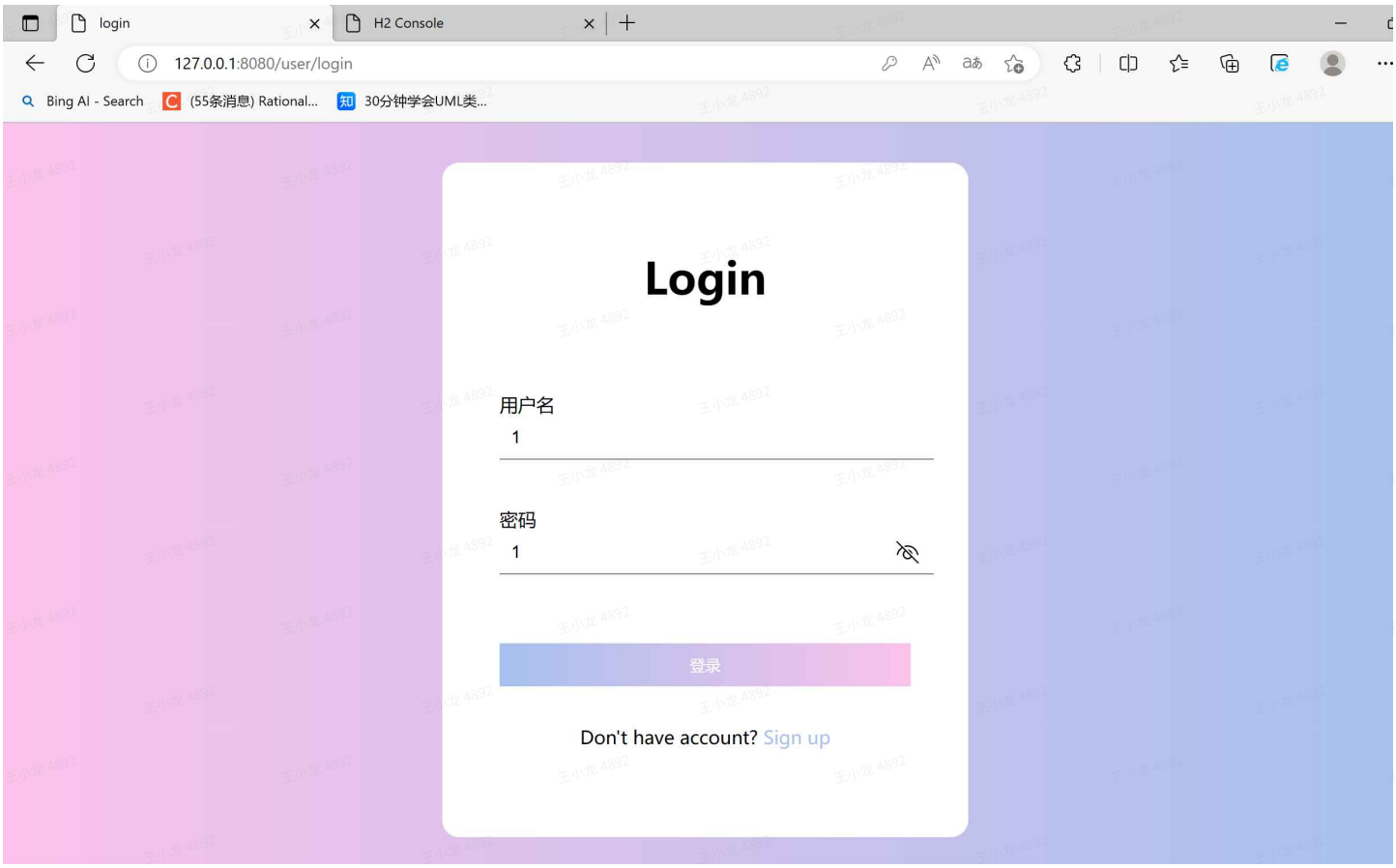
点击“sign up”进入注册界面：



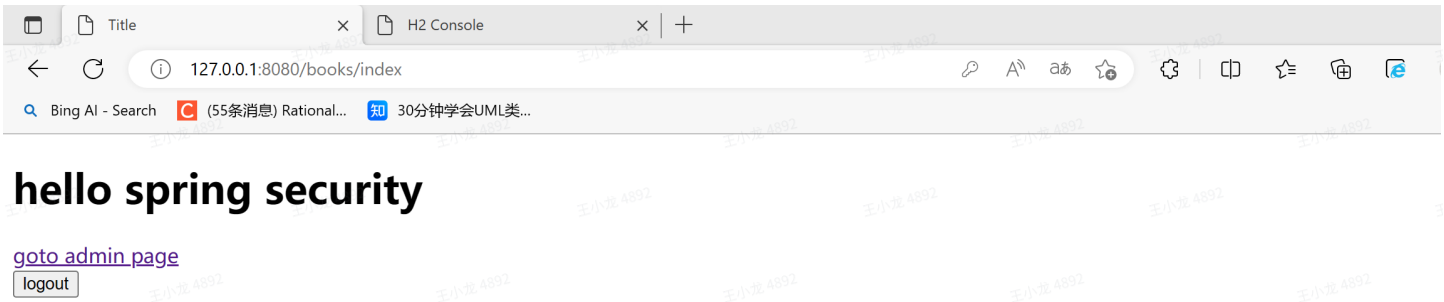
输入用户名和密码，点击注册按钮，跳转回登录界面：



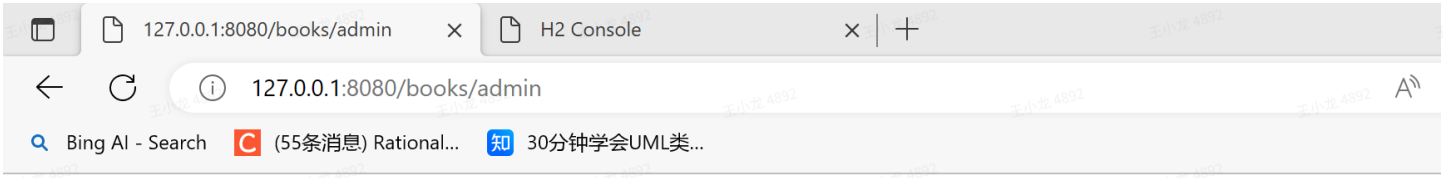
跳转回登录界面后，输入刚才的注册信息，点击“登录”，进入index界面：



在index界面可以看到有两个选项，一个是登出，另一个是前往管理员界面，点击goto admin page,跳转到如下界面：



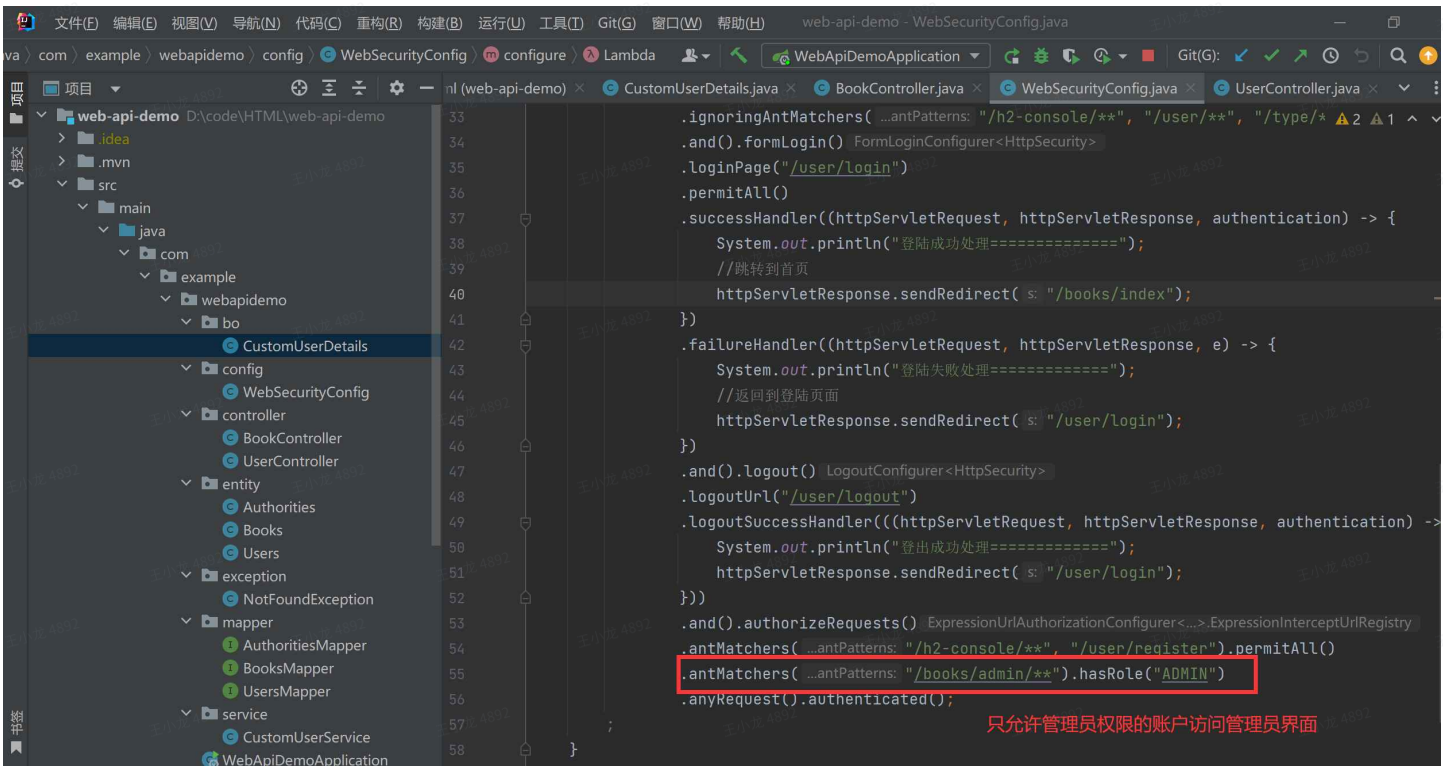
因为我们注册登录的账户不具有管理员权限，故访问被拦截：



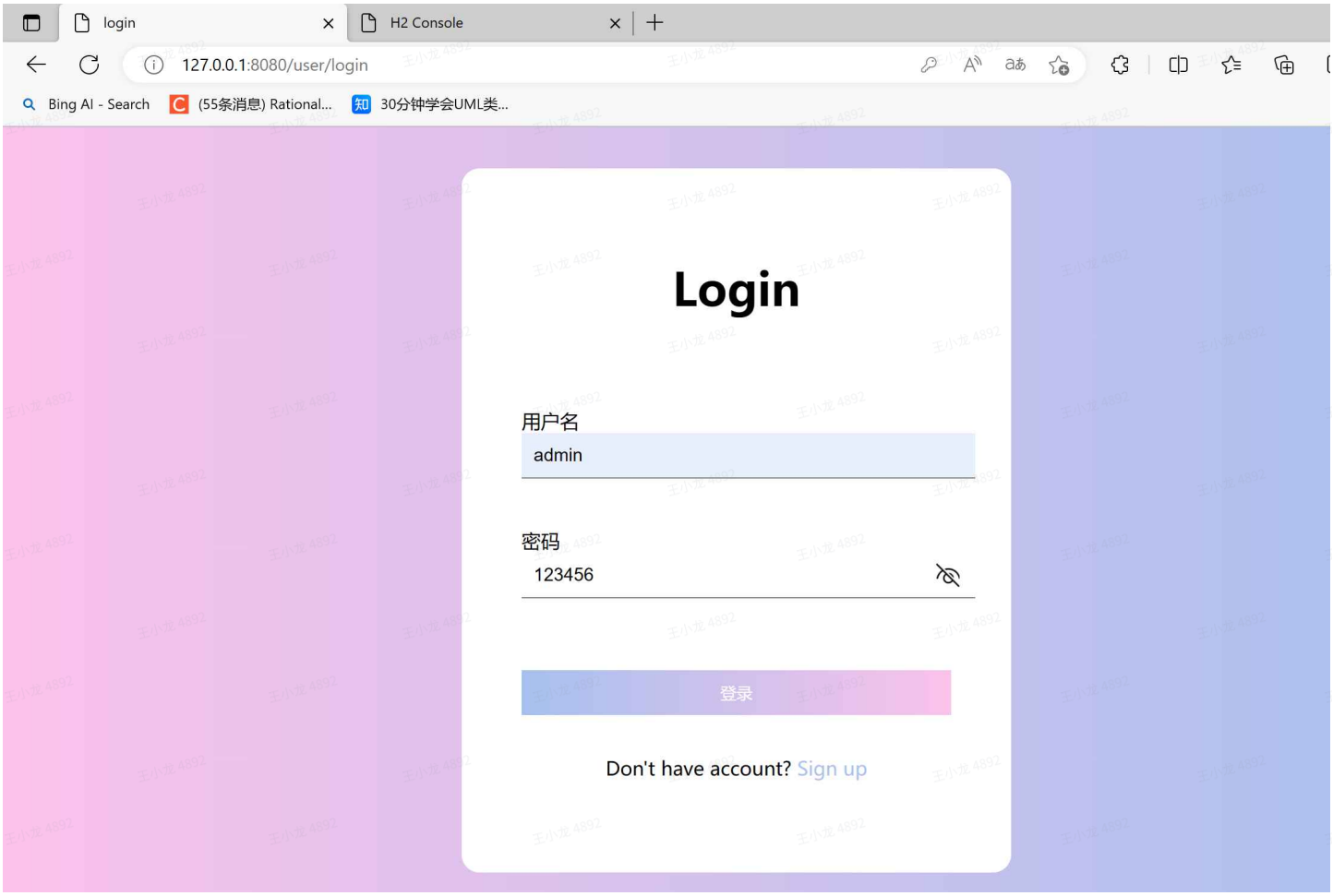
Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

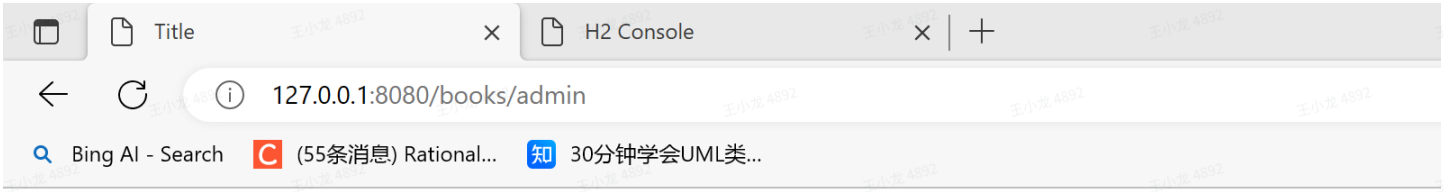
Fri Apr 07 14:48:25 CST 2023
There was an unexpected error (type=Forbidden, status=403).
Forbidden



返回到上一界面，点击layout，回到登录界面，然后以预置的管理员账户登录



进入index页面后，点击goto admin page，进入管理员界面，发现成功进入：



hello admin!

goto index page

logout