**提交/评论功能加入输入攻击拦截，提交/评论功能加入输入攻击拦截。**

下面列出代码中已有的输入攻击拦截措施，以及它们分别在哪些部分实现了防护：

1. SQL 注入防护

代码中防御 SQL 注入的关键在于参数化查询，即在拼接 SQL 语句时，使用问号占位符而不是直接拼接用户输入字符串，从而保证用户输入始终被视为数据，而非 SQL 代码的一部分。具体如下：

登录接口

在 login() 路由中，查询用户时使用了参数化查询：

1. user = conn.execute("SELECT \* FROM users WHERE username = ?", (username,)).fetchone()

这里，username 的值是通过参数传入的，避免了直接拼接字符串带来的 SQL 注入风险。

注册接口

在 register() 路由中，同样采用了参数化查询防护：

1. existing\_user = conn.execute("SELECT \* FROM users WHERE username = ?", (username,)).fetchone()

和插入新用户时：

1. conn.execute("INSERT INTO users (username, password) VALUES (?, ?)", (username, hashed\_password))

通过问号占位符传入变量，确保数据库对用户输入仅作为数据处理。

搜索接口

在 search() 路由中，当传入查询参数 loan\_id 时：

1. **if** loan\_id:
2. data = conn.execute("SELECT \* FROM loans WHERE id = ?", (loan\_id,)).fetchall()
3. **else**:
4. data = conn.execute("SELECT \* FROM loans").fetchall()

同样采用参数化查询，防止使用者在 URL 参数中输入恶意字符串进行 SQL 注入。

2. XSS 防护

针对 XSS 攻击，代码中没有使用专门的输入过滤库（如 bleach）来清洗用户输入。但有以下防护措施：

模板渲染自动转义

所有页面均使用 render\_template 渲染 HTML 模板，Flask 默认使用的 Jinja2 模板引擎会对输出的变量自动进行 HTML 转义。例如，在登录、注册、查询、捐款等页面中，若模板采用默认方式输出变量（如 {{ variable }}），用户输入中的特殊字符（如 <、>）将被转义，防止恶意脚本在浏览器中执行。

**SQL注入和XSS攻击效果：**

如在登录页面的用户名输入框中，输入一个 SQL 注入 payload，比如：' OR '1'='1

这可能会返回数据库中所有用户，从而绕过登录验证，我们输入任何一个数据库中有的密码就可以成功登录网站。

**自动刷单**