桂林电子科技大学2023-2024学年 第2学期

**数据库系统原理综合 实验报告**

**课号：2320954 任课老师：唐麟**

|  |  |  |  |
| --- | --- | --- | --- |
| 项目名称 | **web端即时通信系统** | | |
| 院 系 | **计算机与信息安全学院** | 专业 | **软件工程** |
| 实验日期 | **2024年5月17日** | | |

团队任务分工表

|  |  |  |  |
| --- | --- | --- | --- |
| 题目 | web端即使通信系统 | | |
| 负责人 | 2200310717黄荣钊 | | |
| 合作  完成 |  | | |
| 序号 | 学号 | 姓名 | 个人负责模块 |
| 1 | 2200310717 | 黄荣钊 | 全栈设计与实现、数据库设计与管理 |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |
| 5 |  |  |  |

1. 需求分析

用户注册登陆后，可以通过web进行即时通信，选择一个房间或者与另一个用户，并且还可以搜索历史记录，还能更改密码。

1. 所用开发工具和技术

Phoenix 是一种用于 Elixir 编程语言的 Web 框架，基于 Erlang 虚拟机构建。它旨在具有高度可扩展性、容错性和易用性。Phoenix 提供了一组用于构建 Web 应用程序的工具和约定，包括路由系统、控制器层和用于生成 HTML 的模板引擎。

Phoenix LiveView 是基于 Phoenix 构建的一种技术，它允许开发人员使用最少的 JavaScript 构建丰富、交互式的 Web 应用程序。LiveView 使用一种称为“服务器呈现反应性”的技术，以实时更新 UI，而无需复杂的 JavaScript 代码。使用 LiveView，开发人员可以编写简单的 Elixir 代码来处理用户交互和更新 UI，而框架负责其余的工作。

Phoenix 和 Phoenix LiveView 的主要优势在于其可扩展性。Elixir 和 Erlang 被设计用于处理高度并发，并且可以轻松扩展以处理大量用户。这使得 Phoenix 和 LiveView 非常适合用于构建高流量 Web 应用程序。另一个优势是其易用性。Phoenix 提供了一套用于构建 Web 应用程序的明确约定，而 LiveView 则抽象了构建交互式 UI 的许多复杂性。这意味着开发人员可以在最短的时间内构建复杂的应用程序，并且付出的努力最少。Phoenix 和 LiveView 具有高度的容错性。Erlang 虚拟机被设计用于处理故障，并且可以自动重新启动已失败的进程。这意味着即使在发生错误或崩溃的情况下，Phoenix 和 LiveView 应用程序也可以继续运行。

数据管理系统，选择了postgresql配合Elixir的Ecto库使用。Ecto 是 Elixir 编程语言的数据库映射库，旨在提供简单、可扩展和高性能的数据库访问。Ecto 的一个主要优势是其查询接口，它允许开发人员使用 Elixir 代码构建复杂的数据库查询，而无需编写 SQL。Ecto 还提供了一套强大的迁移工具，用于管理数据库模式更改，并支持多种数据库，包括 PostgreSQL、MySQL 和 SQLite。PostgreSQL 是一种高级的开源关系数据库管理系统，具有强大的功能和可扩展性。PostgreSQL 的优势在于其支持复杂的查询和索引、数据类型丰富、可扩展性高以及其在安全性和可靠性方面的表现，并且易于使用和管理，能轻松满足我们的需求。

1. 数据库设计

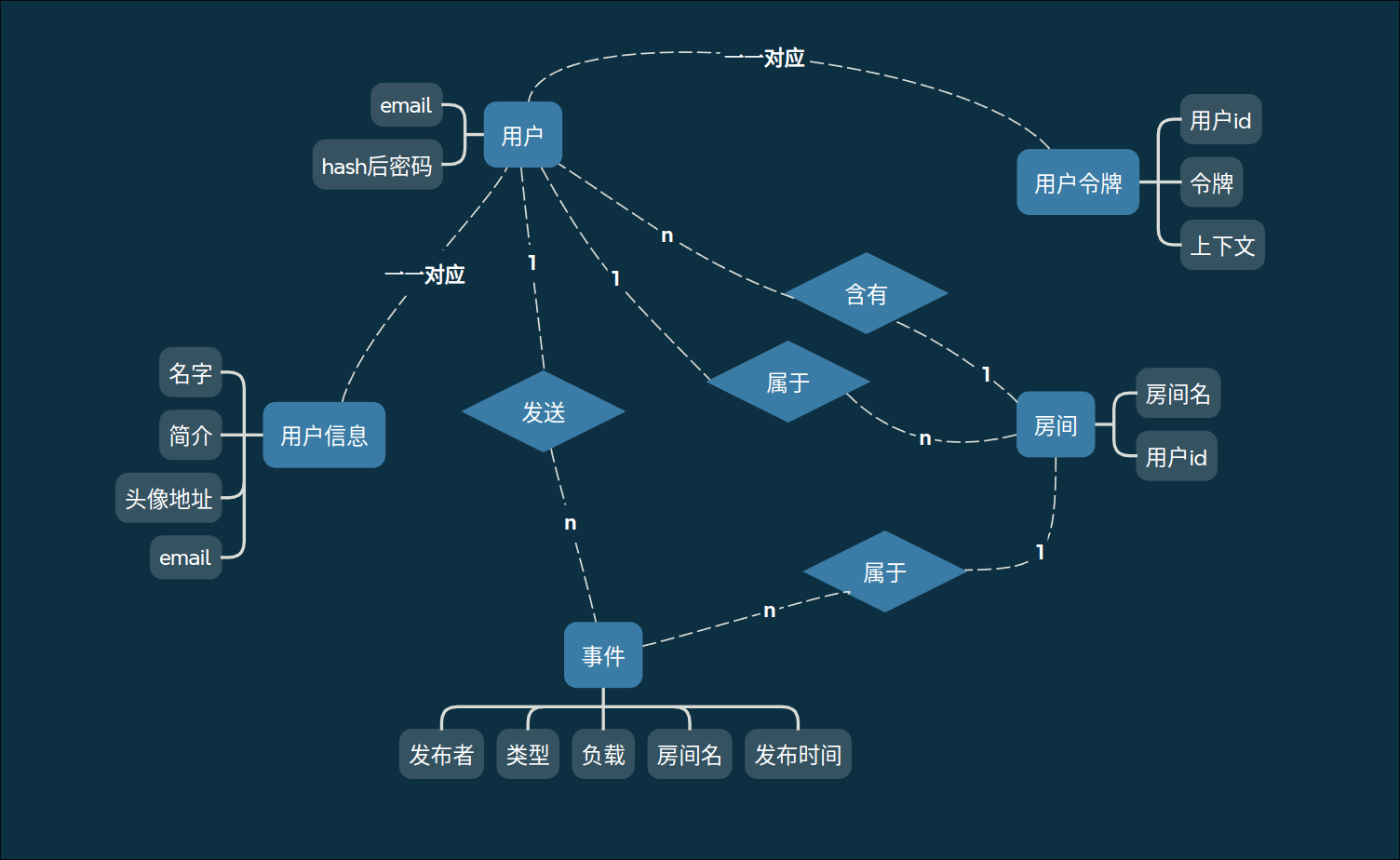
概念结构设计、逻辑结构设计、物理结构设计，别弄错！

**概念结构设计：**

1. 用户：代表用户实质身份的数据
2. 用户令牌：当用户登陆后，会生产一个唯一的token用于指代该会话，用于用户的身份信息验证
3. 用户信息：用于记录用户相关的信息，如名字、个人介绍、头像等等
4. 事件：用于记录房间中每个事件，如某某加入房间，某某发送了一条信息
5. 房间：用于记录每个成员与每个房间的关系

（任务体量有限，简单起见，不设房间信息表）

ER图如图所示：



逻辑结构设计：

1. 用户表：

id: bigint, 由Ecto自动创建的id, 用于其内部管理

email: citext, 用户的邮箱

hashed\_password: character varying(255), hash保存的用户密码

confirmed\_at: timestamp,

inserted\_at: timestamp, 用户创建时间

updated\_at: timestamp, 用户更新时间

1. 用户令牌表：

id: bigint,

user\_id: bigint,

token: bytea,

context: character varying(255),

sent\_to: character varying(255),

inserted\_at: timestamp,

1. 用户信息表：

id: bigint,

name: character varying(255),

bio: character varying(255),

email: character varying(255),

avatar: character varying(255),

inserted\_at: timestamp,

updated\_at: timestamp,

1. 房间表：

id: bigint,

user\_id: bigint,

room\_name: character varying(255),

inserted\_at: timestamp,

updated\_at: timestamp,

1. 事件表：

id：bigint,

sender: character varying(255)

type: character varying(255), 事件类型，msg/join/leave/img

payload: character varying(255), 事件内容,

当type是msg时，这是消息内容

当type是join和leave时，这用不上

当type是img时，这是图片消息的url

room: character varying(255)**,**

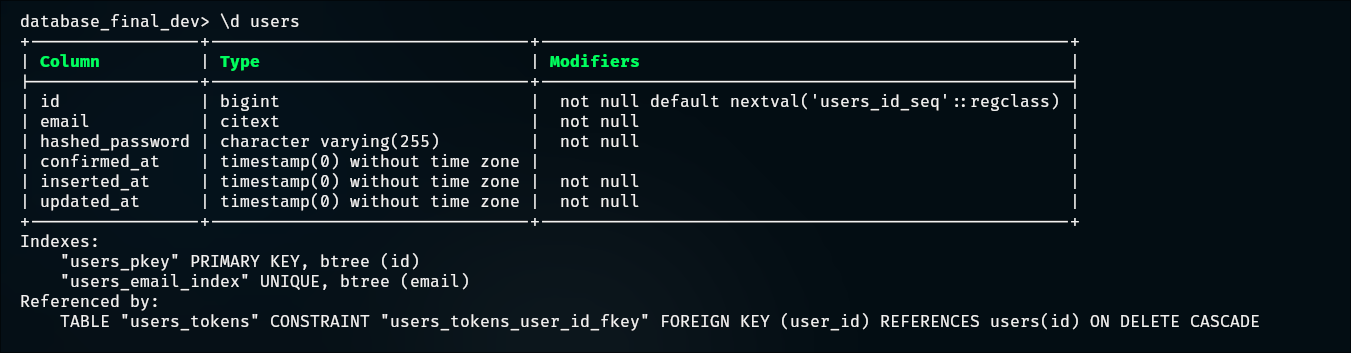
inserted\_at: timestamp,

updated\_at: timestamp

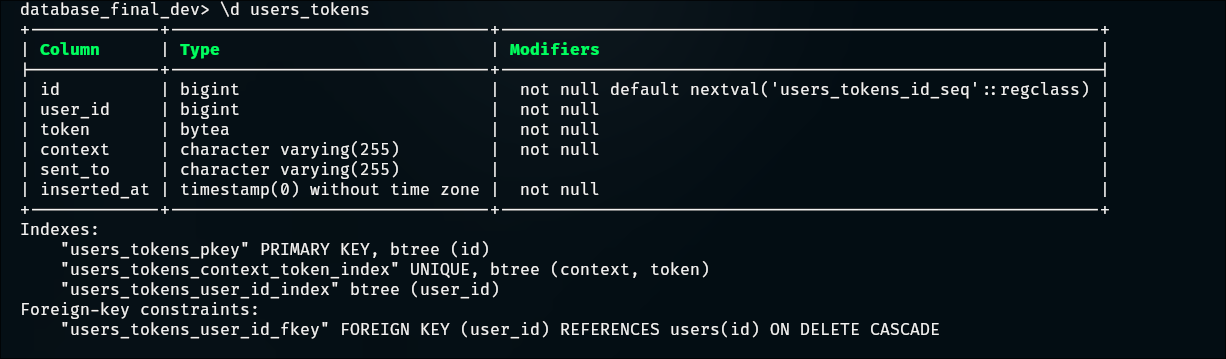
物理结构设计：

各表信息及其索引一览。

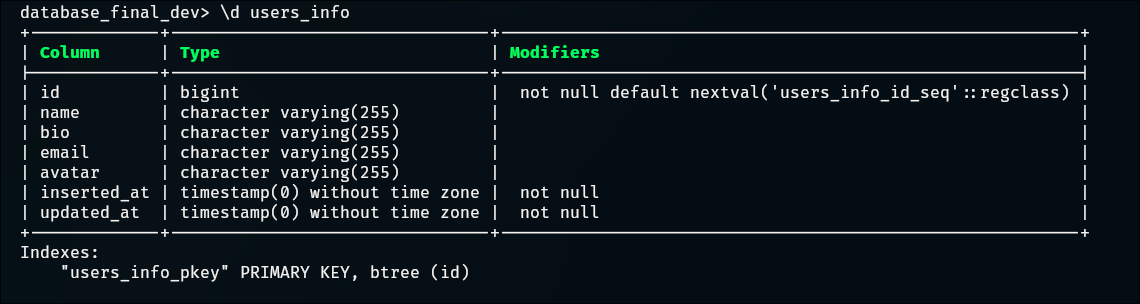
用户表：



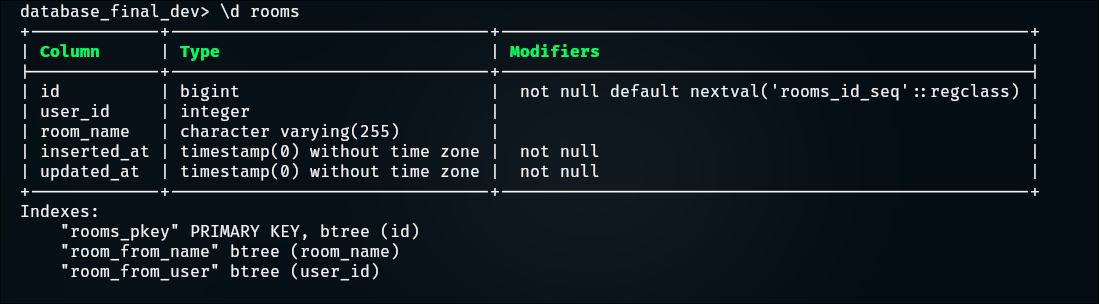
用户令牌表：



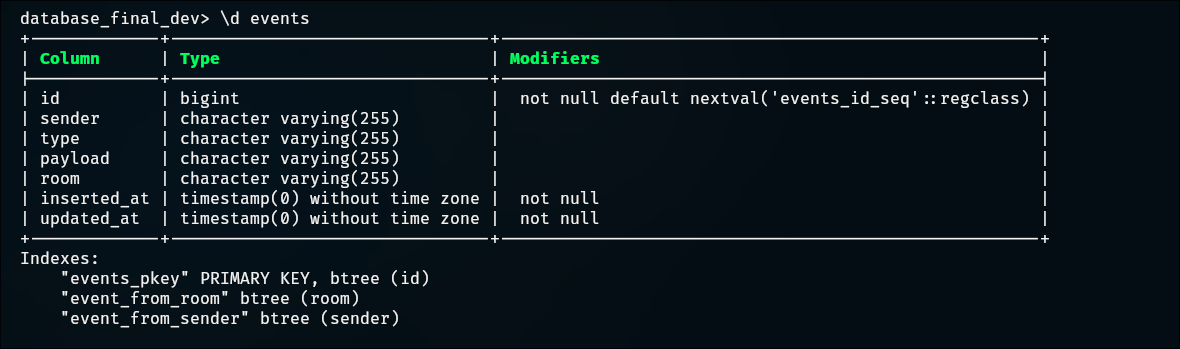
用户信息表：



房间表：



事件表：



**相关：**

1. 索引:
2. 在事件表建立了房间名索引
3. 在房间表建立了房间名索引和用户email索引
4. 在用户信息表建立了email索引
5. SQL语句的优化由ecto自动生成四元式进行依赖分析、优化
6. 为了拓展性、 可维护性、安全性和完整性考虑，全权使用ecto进行数据库管理与验证，因此用后端逻辑替代触发器（触发器各数据库管理系统的语法与实现都不同），防止因突发事件无法迁移其他数据库。
7. Ecto 通过使用参数化查询和模式化输入来杜绝数据库安全问题，如 SQL 注入攻击。

在参数化查询中，Ecto 将查询中的用户输入作为参数传递，而不是将其直接嵌入到查询字符串中。这样一来，即使用户输入恶意的 SQL 代码，也不会对查询产生影响。在模式化输入中，Ecto 强制执行严格的数据类型和验证规则，以确保用户输入的格式和内容符合预期。这可以有效地防止恶意用户通过输入不合法的数据来攻击数据库。此外，Ecto 还提供了一些其他的安全特性，如查询过滤和访问控制，以帮助开发人员更好地保护他们的数据库。

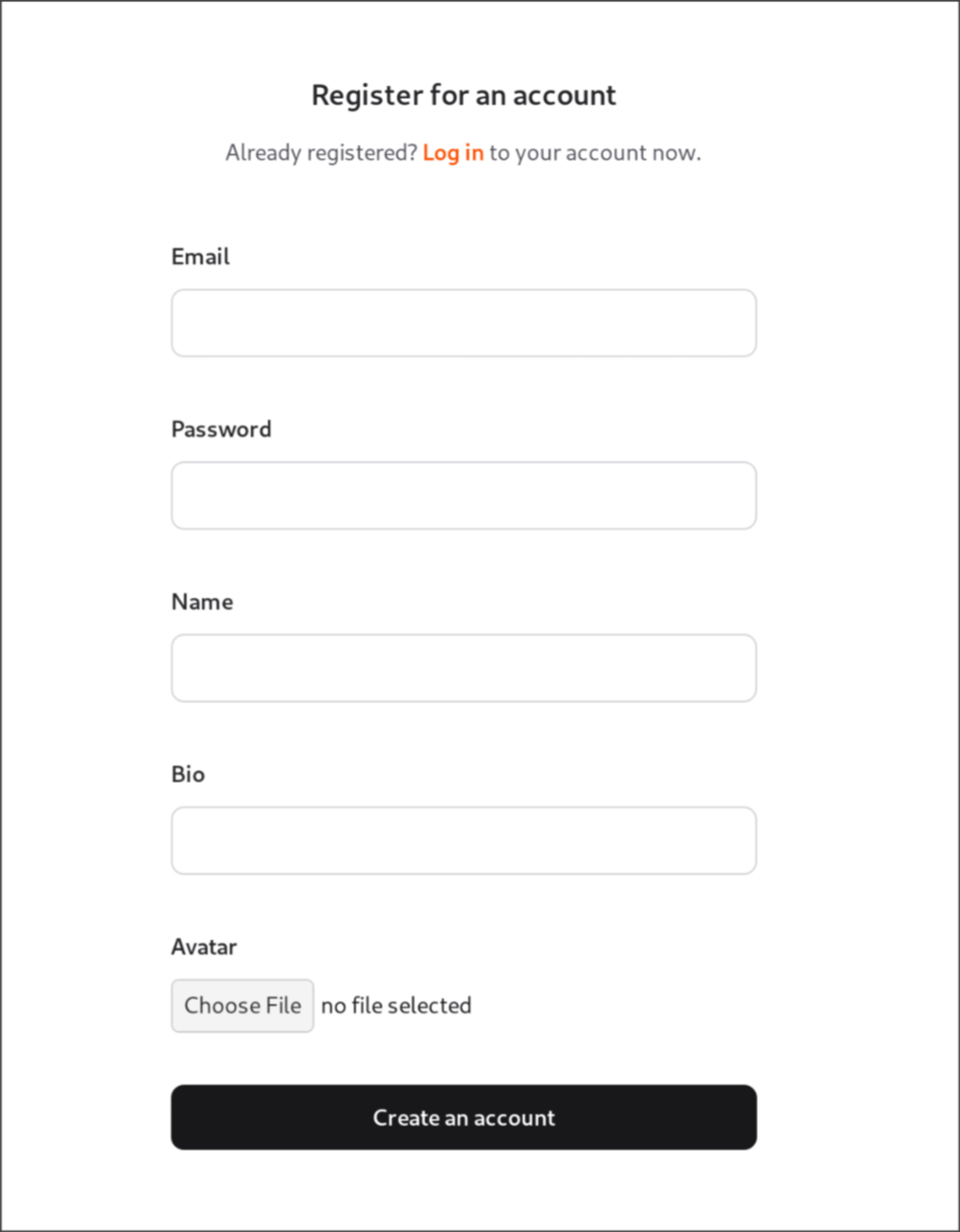
1. 完整性约束，当用户被删除时，对应的用户信息表、用户令牌表、房间表中与该用户对应的条目会被一同删除，但事件表（聊天记录表）不会受影响。由于我们使用了Ecto,方便起见，不对任何表设置外键。
2. 详细设计与实现

整个应用遵循Domain Specificed Design, 并且在界面实现上遵循MVC模式。

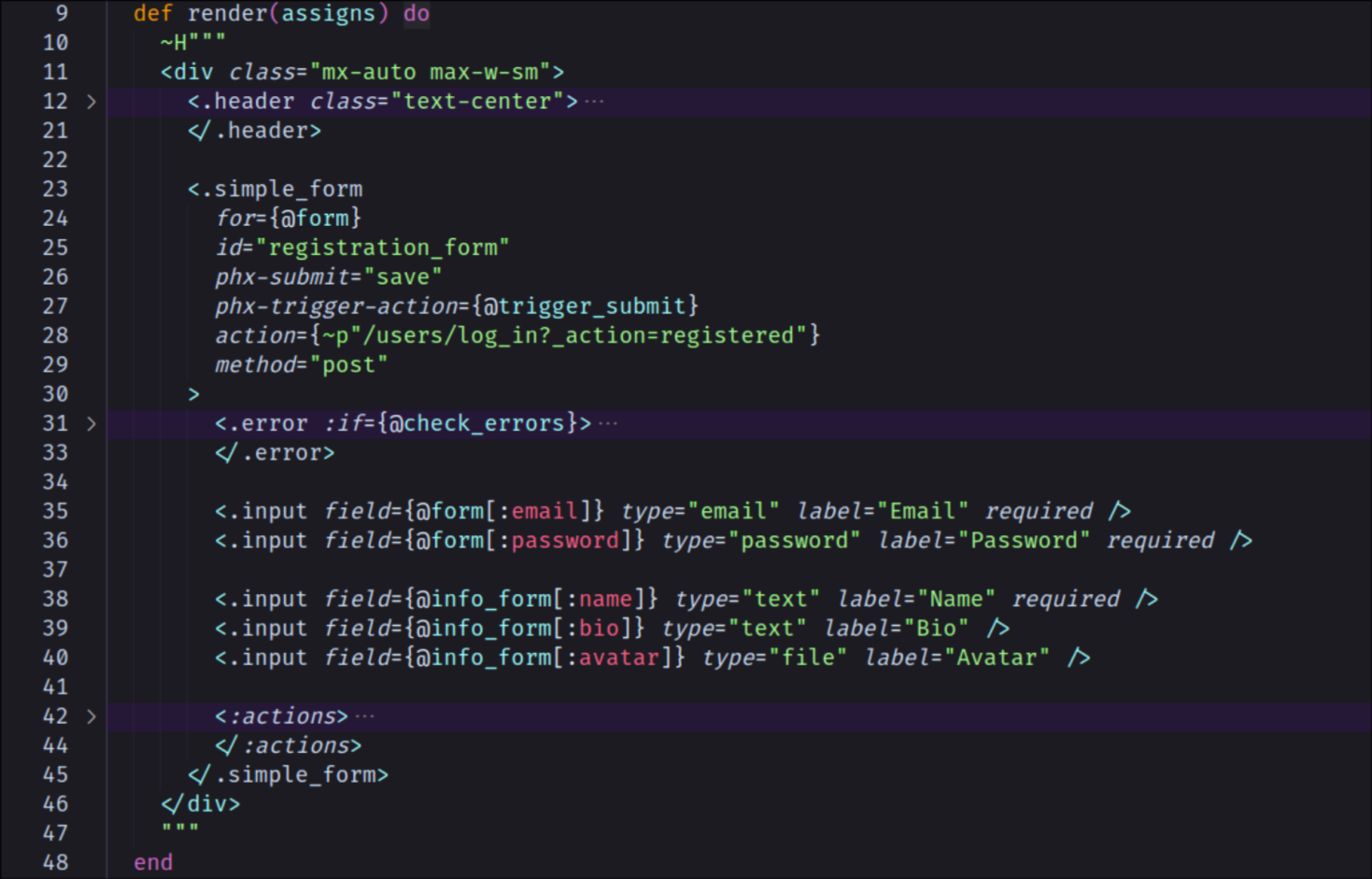
**注册页面、修改页面、登陆页面：**

这三个页面的样式与逻辑大体相似，放置几个表单，当输入发生变化时，进行数据验证，提交时尝试执行相应的增删差改操作。

如注册界面：



相关代码在database\_final\_web/live/user\_registration.ex中，

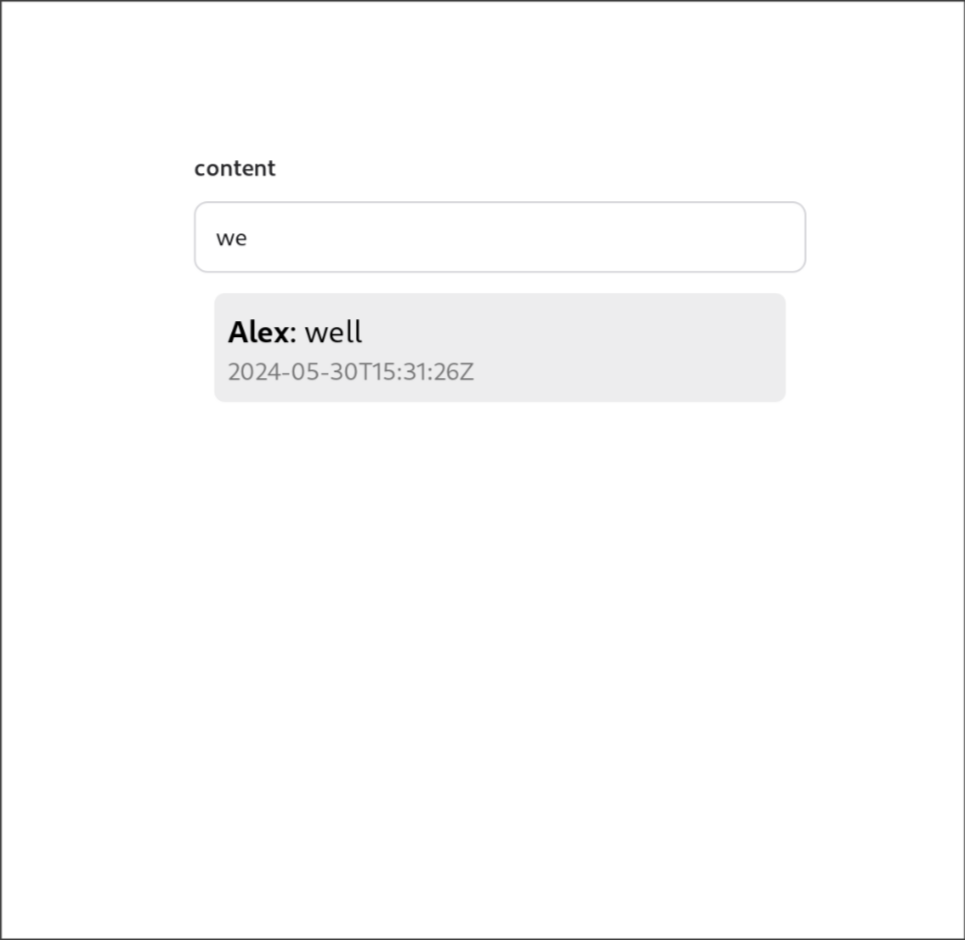




如代码所示，当表单提交时，首先尝试创建用户，成功后可创建用户信息，并对用户comfirm，而数据验证的代码在源文件中相邻的事件处理函数中，不做过多展示；另外两个页面逻辑类似，不作过多展示。

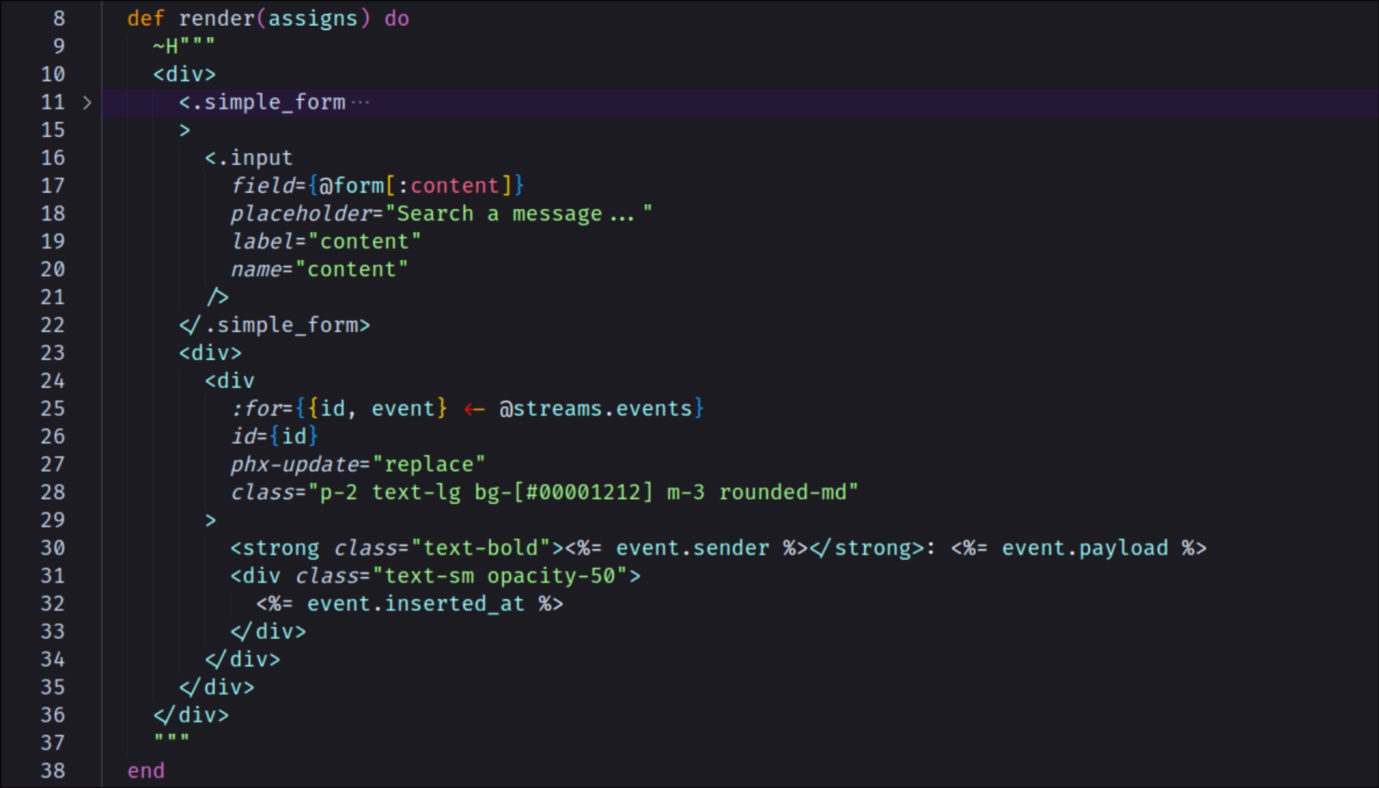
搜索页面：

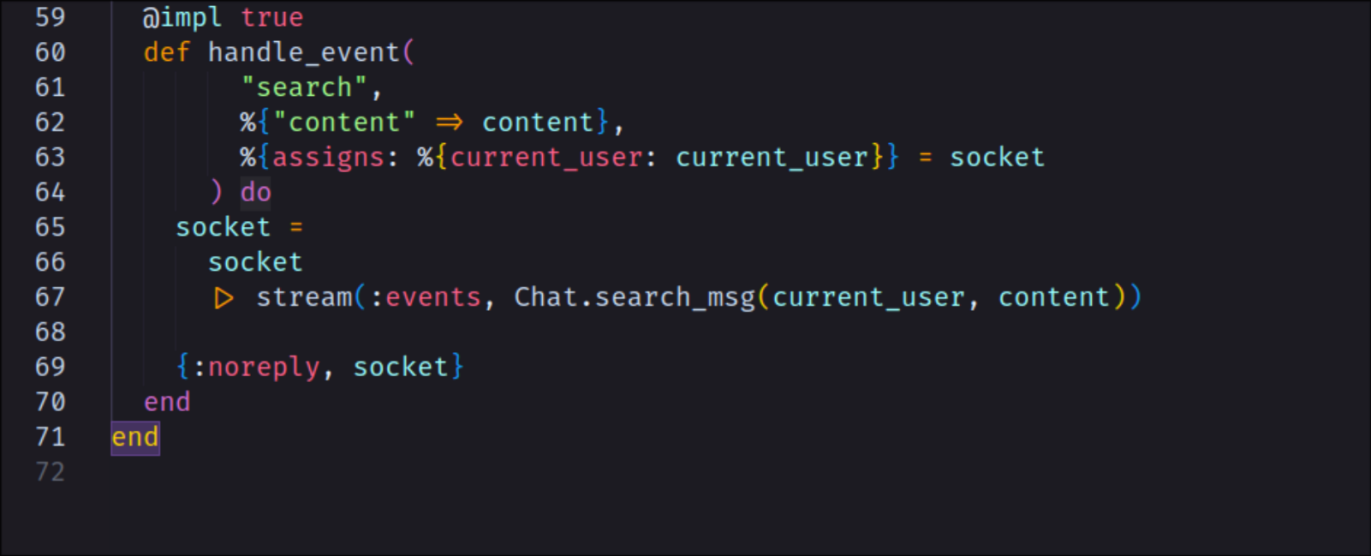
搜索页面在/search和根目录下



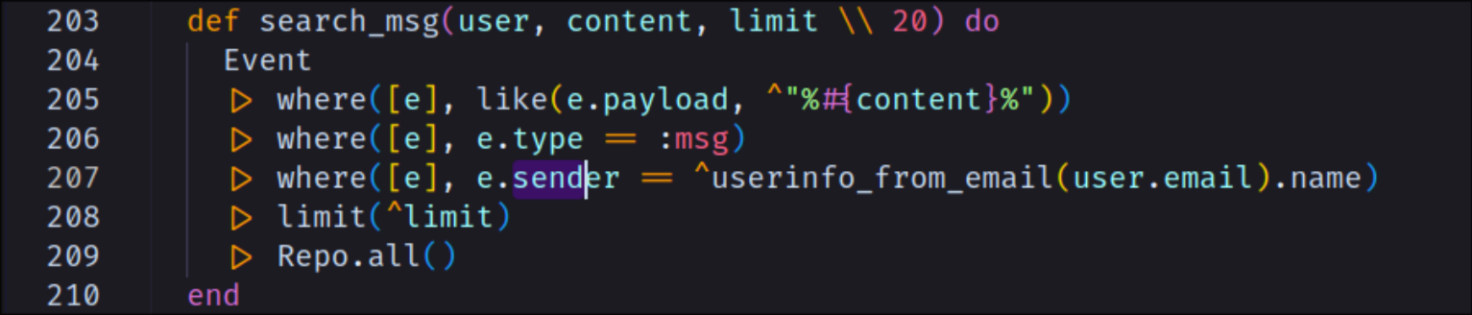
界面代码：

界面由只有一个输入的表单和消息列表组成





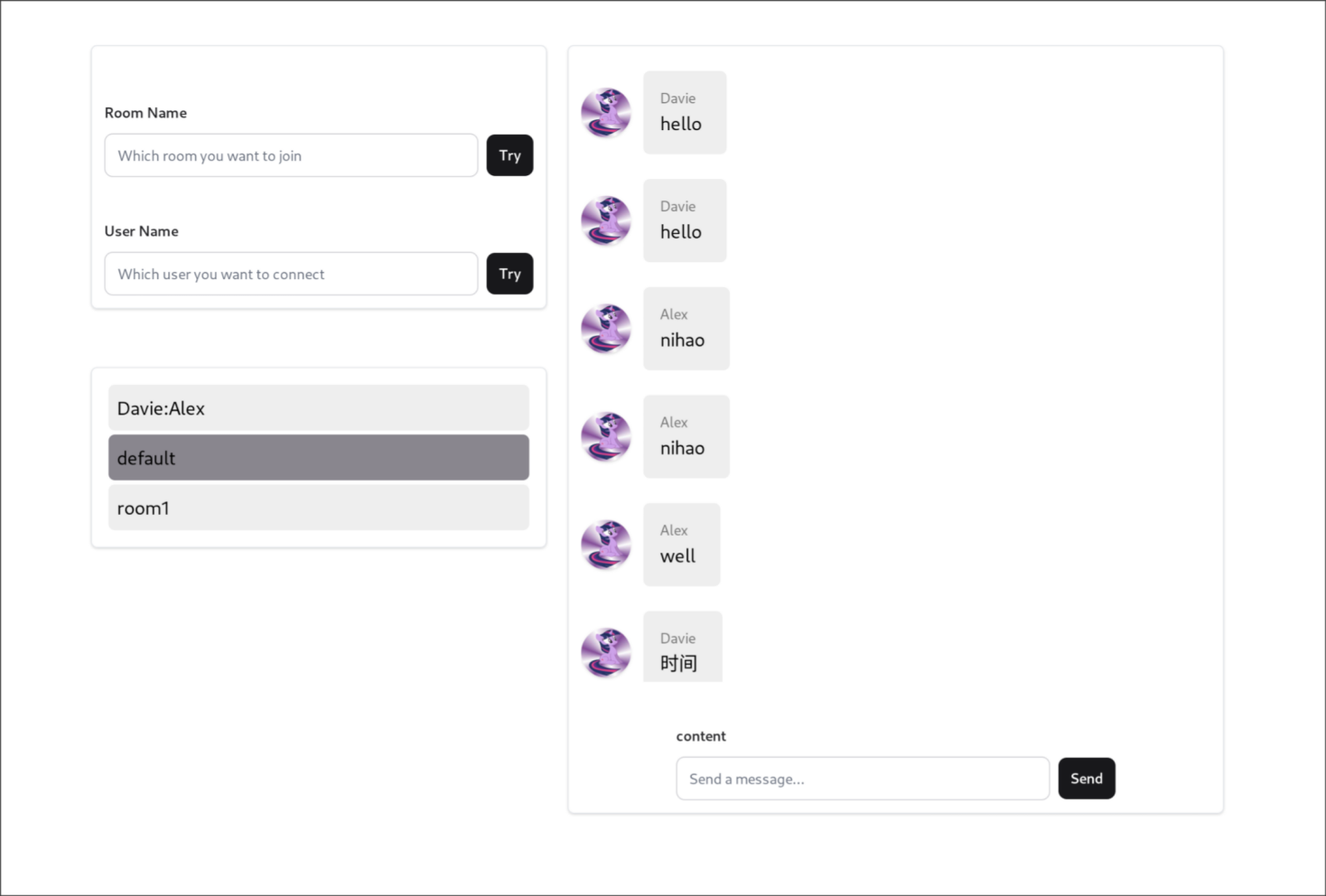
当输入内容更改时，上面的事件处理器将会被触发。该处理器首先从输入中读取搜索内容，然后将搜索到的相关的内容更新到事件列表中。搜索逻辑如下：



聊天室页面:

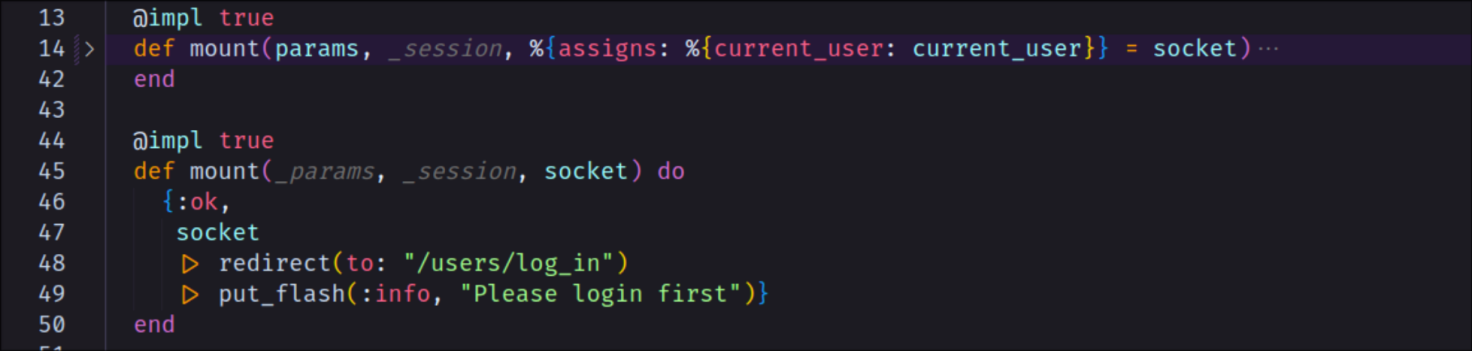
聊天室在/room路由下。

聊天室由三大板块组成，左上用于加入新的房间，左下显示房间列表，右边就是消息列表。



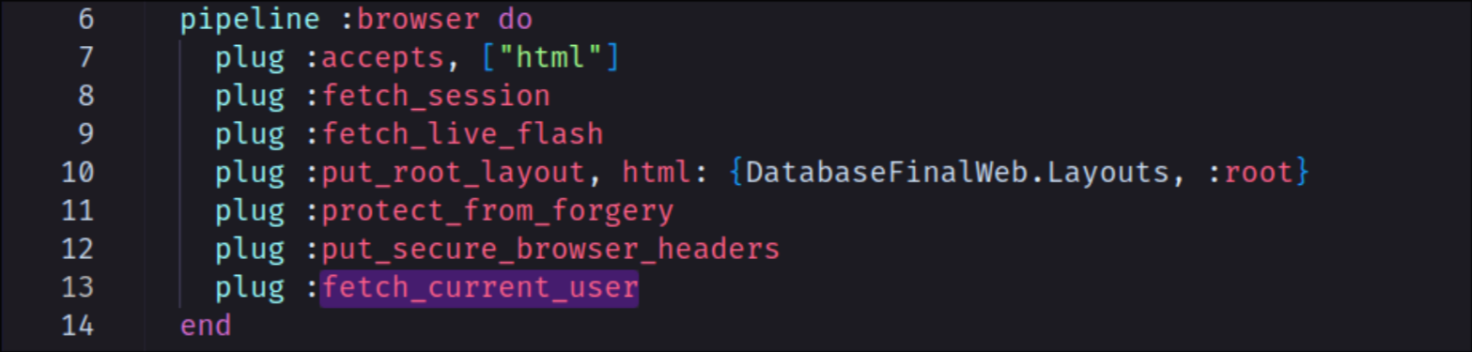
身份验证：

在路由到/ /room /search时，如果没有当前用户的登陆token, 则会被重定向到登陆页面，防止意外发生，如在路由进入/room时，

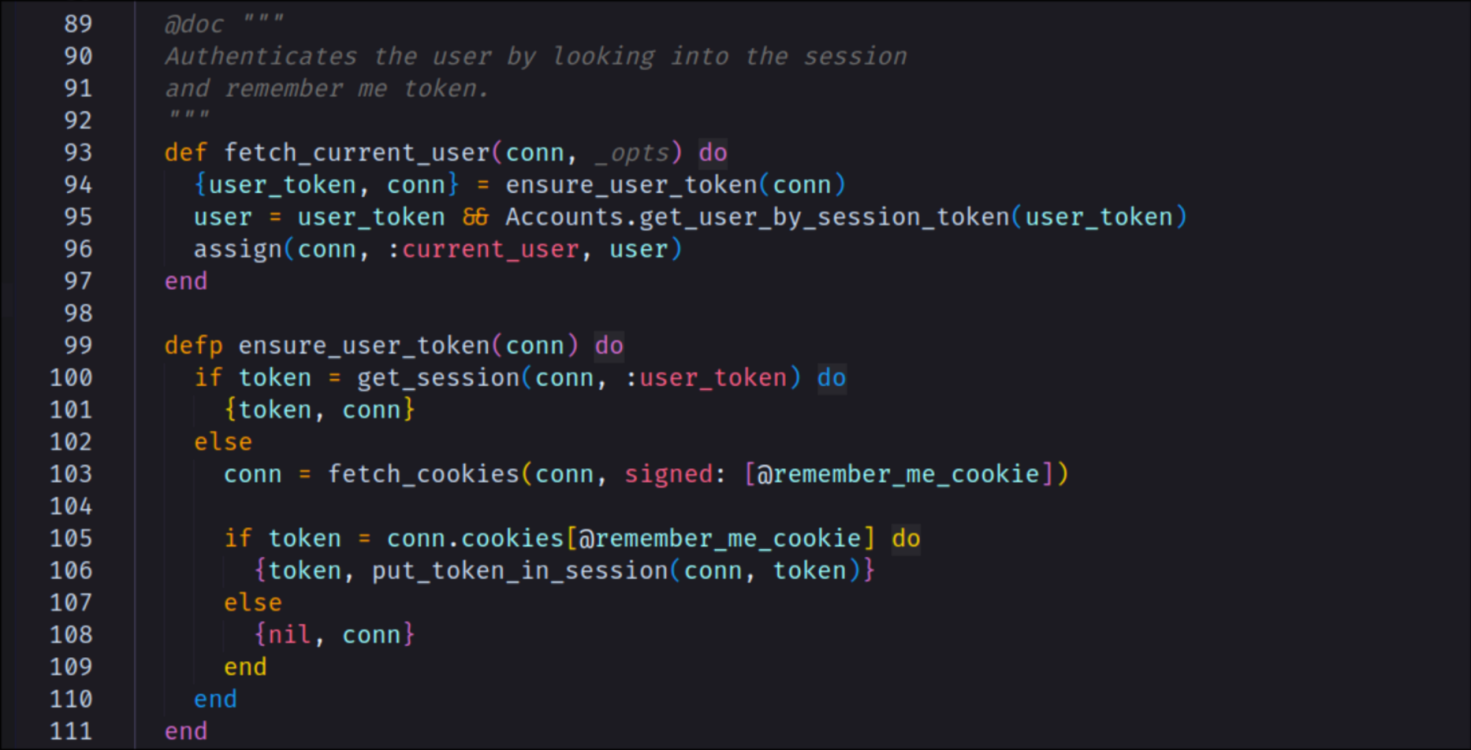


phoenix会执行房间模块中的mount函数，如图所示，第一个折叠的mount函数即是正常的聊天室初始化逻辑，而当current\_user为空时，elixir的模式匹配功能就会进入第二个mount函数中，从而对页面进行重定向。

current\_user由路由管道中的fetch\_current\_user Plug提供，



该plug会尝试从浏览器请求中读取cookie中的token信息，然后找到对应的用户身份，将该身份写入响应中的current\_user中。也就是说，若没有找到对应的用户信息，页面响应中，就不会存在current\_user。



分模块进行设计，小组成员分着写这一部分，然后合在一起。

这一个部分就是详细的将自己的系统详细设计与实现介绍清楚。设计就是画流程图之类，并配上文字说明。实现就是实现效果（截图）配上关键代码、文字说明是怎么做出来这样的结果的。

结构可以灵活些，比如根据流程顺序分块介绍，比如根据用户分类的功能介绍。

但是需要注意：（1）不能只有图和代码，必须有文字说明介绍（2）不能大段大段代码，**只能是关键的代码**。而且如果出现代码，那么必须有文字说明（不是代码上的注释）。（3）如果有用到设计模式的同学，这里可以好好的写写。。。。。。。。。。。

1. 测试

主要写功能测试：根据需求，将系统分为多个场景，以场景为主线，结合等价类划分、边界值分析等等设计测试用例，进行测试。测试发现问题，进行修改，修改后进行回归测试。。。。。。。。

如有web开发，可以写写兼容性测试和性能测试。

1. 存在的问题及改进思路
2. 附录

系统中所用的SQL语句