心盾·智护后端开发技术文档

本项目是“心盾·智护”心理健康风险预警系统的后端应用，采用 **Python + Flask** 技术栈进行开发，数据库使用 **MySQL**。后端划分为多个功能模块，包括：**用户认证模块**、**心理测评模块**、**火场任务记录模块**、**风险预警模块**以及**管理员报告模块**。本文档详细描述项目的结构设计、数据库模式、各模块的接口说明、安全机制、风险预警规则以及系统数据流动等内容。

项目结构

后端项目遵循 Flask 框架的最佳实践进行组织，采用模块化结构，方便开发和维护。主要结构如下：

- **应用入口**：项目根目录下包含应用启动脚本（如 `app.py` 或 `run.py`），用于创建 Flask 应用实例并加载配置、初始化数据库连接等。
- **配置文件**：`config.py` 定义了配置参数，例如数据库连接URL、秘钥（`SECRET_KEY`）、调试模式开关等。通过不同的配置类可以支持开发、测试和生产环境。
- **应用包**：主应用代码位于 `app/` 包内，实现各功能模块。典型结构示例如下：

```
project/
├── app.py  # Flask应用入口（创建app，注册蓝图）
├── config.py  # 配置文件（数据库URI、Secret Key等）
├── requirements.txt  # 项目依赖库列表
└── app/  # 应用模块包
    ├── __init__.py  # 初始化Flask应用，配置数据库等
    ├── models.py  # 定义数据库模型（或按模块拆分为多个模型文件）
    ├── auth.py  # 用户认证模块的路由与逻辑
    ├── assessment.py  # 心理测评模块路由与逻辑
    ├── tasks.py  # 火场任务记录模块路由与逻辑
    ├── warning.py  # 风险预警模块路由与逻辑
    ├── admin.py  # 管理员报告模块路由与逻辑
    └── utils/  # 工具函数（如安全相关、辅助方法等）
```

上述结构中，每个模块（如认证、测评、任务、预警、管理员）都有各自的路由和业务处理代码，实现单一职责。`models.py` 定义了所有数据库实体模型（或者按模块拆分为多个文件）。各模块可以使用 **Flask Blueprints（蓝图）** 进一步组织，将不同模块的路由注册到应用实例上，从而实现清晰的代码分层。

在应用初始化（`app/__init__.py`）时，会完成以下工作：加载配置（例如从 `config.py` 读取数据库URI和Secret Key等）、初始化数据库连接（使用SQLAlchemy或其它ORM连接MySQL）、注册各模块的蓝图路由、设置全局错误处理和中间件（如跨域CORS支持、身份认证钩子函数等）。这样，项目结构清晰，方便团队协作开发和后期维护扩展。

数据库设计

系统使用 MySQL 数据库来存储和管理数据。数据库设计遵循第三范式，确保数据一致性和减少冗余。主要的数据表及其结构如下：

- **用户表（`users`）**：存储用户的基本账号信息。主要字段包括：
- `id`：主键，自增整数。
- `username`：用户名，唯一识别用户。
- `password_hash`：密码哈希值（采用安全哈希算法存储，不保存明文密码）。
- `role`：用户角色（如 `"user"` 表示普通用户，`"admin"` 表示管理员）。
- `email`：电子邮箱（可选，用于找回密码或通知）。
- `created_at`：账户创建时间。
  （**说明**：普通用户和管理员账户都存储在此表，通过 `role` 区分权限。）

- **心理测评表（`assessments`）**：记录用户每次心理测评的结果。主要字段包括：

- `id`：主键，自增。
- `user_id`：外键，关联到用户表 `users.id`，表示哪个用户的测评。
- `score`：测评得分（例如0~100分，分数越高表示心理压力或风险越高）。
- `level`：测评结果等级（如 `"低"`、`"中"`、`"高"` 风险，可根据score区间得出）。
- `details`：测评详情（可选，JSON或文本，保存用户在问卷中各题的答案或各维度分数）。
- `created_at`：测评完成时间。
  （**说明**：系统可预先定义不同心理测评问卷，例如压力指数评估等，用户作答提交后计算 `score` 及 `level`。）

- **测评问题表（`questions`）**：（若系统需要动态存储测评问卷，可设计此表。） 保存心理测评的试题和选项等。主要字段：

- `id`：主键。
- `assessment_type`：测评类型标识，例如 `"stress_test"` 表示压力测试问卷。
- `question_text`：问题文本。
- `options`：可选项（如果是选择题，则存储JSON数组；如果是问答题，可为空）。
- `score_weight`：该题目在总分计算中的权重或分值。
  （**说明**：如果题库固定且不需动态管理，可以不设计此表，将问卷题目直接写死在代码或配置中。）

- **火场任务记录表（`task_records`）**：记录消防员（用户）执行火场任务的情况。主要字段包括：

- `id`：主键，自增。
- `user_id`：外键，关联用户表，表示执行任务的用户。
- `task_date`：任务日期时间。
- `location`：任务发生地点。
- `description`：任务描述（例如火情简要、任务内容）。
- `severity`：任务严重程度等级（例如 1-5 级，或 `"一般"`、`"重大"` 等）。
- `casualties`：伤亡人数（如果有的话，没有则为0）。
- `duration`：任务持续时间（以分钟为单位，或其他合适单位）。

- `created_at`：记录创建时间。
  （**说明**：该表用于量化用户所经历的火场任务压力。如果一个任务特别严重（高`severity`或有伤亡），可用于后续风险评估。）

- **风险预警表（`risk_warnings`）**：保存系统生成的心理风险预警信息。主要字段包括：

- `id`：主键。
- `user_id`：外键，关联到用户。
- `warning_level`：预警等级，例如`"高风险"`、`"中风险"`等。
- `message`：预警信息描述（例如“近期心理压力过高，请关注心理健康”）。
- `trigger_time`：预警生成时间。
- `resolved`：布尔值，表示该预警是否已被处理或标记（管理员查看后可标记为已处理）。
  （**说明**：当用户的心理测评结果或任务记录达到预警条件时，系统将插入一条预警记录。管理员可查看所有预警记录并跟进处理。）

  关系说明：以上表之间通过外键关联：`assessments.user_id`和`task_records.user_id`都参考`users.id`实现一对多关系（一个用户对应多次测评和多条任务记录）。`risk_warnings.user_id`也参考`users.id`，用于标记哪个用户触发了预警。一条预警通常由最近的测评或任务触发，其`message`可以包含相关说明。为了高效查询，在外键列上均建立了索引。例如，在`assessments.user_id`、`task_records.user_id`和`risk_warnings.user_id`上创建索引，以便快速检索某用户的相关记录。

数据库采用 InnoDB 引擎以支持事务和外键约束，确保数据完整性。所有时间戳（如`created_at`、`task_date`等）使用 DATETIME 类型记录，时间统一为 UTC 或本地时间并注明时区，以方便日后分析。字符编码使用 UTF8MB4 以支持中英文及特殊符号存储。

模块与接口说明

后端通过RESTful风格的API接口向前端提供服务。各模块划分清晰，每个模块提供一组相关的接口，具体说明如下。

用户认证模块

模块职责：用户认证模块负责用户的注册、登录，以及提供身份验证机制。该模块确保只有合法用户才能访问受保护的接口，并为后续请求提供令牌（Token）用于认证。主要功能包括：新用户注册、用户登录获取令牌、获取当前登录用户信息等。

相关数据表：`users`（用户表）。注册时向该表插入新用户记录；登录时验证`username`和`password_hash`。

主要接口：

- **用户注册接口** – **`POST /api/auth/register`**：接受用户提交的注册信息（如用户名、密码、邮箱等），创建新用户账户。成功注册后返回操作结果（一般返回简要信息或新用户ID）。密码在后端进行哈希后存储，确保安全。
- **用户登录接口** – **`POST /api/auth/login`**：接受用户名和密码，验证凭据。若验证通过，生成会话令牌（JWT 等）并返回给客户端。令牌将在随后的请求中用于认证用户身份。若验证失败，返回认证错误（HTTP 401）。
- **获取用户信息接口** – **`GET /api/auth/profile`**：（需登录）返回当前登录用户的基本信息。客户端需在请求头中附加授权令牌，例如：`Authorization: Bearer <token>`。后端验证令牌有效后，返回用户的资料（如用户名、角色、注册日期等）。

接口请求及响应示例：

- **登录请求示例**：用户通过登录接口获取JWT令牌。

请求：

```
POST /api/auth/login
Content-Type: application/json

{
  "username": "alice",
  "password": "mypassword"
}
```

响应：

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "user_id": 1,
  "role": "user"
}
```

上述响应中，`token`为JWT格式的身份令牌，`user_id`是用户标识，`role`表明用户角色。客户端应保存此令牌，并在后续请求的HTTP头部加入`Authorization: Bearer token`。

- **注册请求示例**：

请求：

```
POST /api/auth/register
Content-Type: application/json

{
  "username": "bob",
  "password": "securePass123",
  "email": "bob@example.com"
}
```

响应：

```
{
  "message": "User registered successfully",
  "user_id": 2
}
```

服务器会返回简单的成功消息和新用户的ID。密码经过哈希处理后存储在数据库中。后续可以用注册的凭据进行登录获取token。

实现要点：用户认证模块在实现时，使用Flask提供的请求上下文获取 JSON 请求数据，使用数据库模型验证用户。密码验证通过对用户提交的密码进行同样的哈希后与数据库中的 `password_hash` 比对。生成JWT令牌可使用诸如 `PyJWT` 或 `flask-jwt-extended` 库，令牌中至少包含用户ID和有效期等信息。所有涉及身份的接口均需验证令牌有效性，未登录或令牌失效时返回403或401错误。

心理测评模块

模块职责：心理测评模块提供心理问卷测试相关的功能接口。用户可以通过该模块获取心理测评问卷（如果前端未写死问卷题目），提交心理测评答案，查看测评结果历史等。本模块通过量表测试来评估用户的心理压力或心理健康状态。

相关数据表：`assessments`（心理测评表）以及可能的 `questions`（测评问题表）。每次用户提交测评，将在 `assessments` 表中新建记录保存结果。

主要接口：

- **获取测评问卷接口** – **`GET /api/assessment/questions`**：返回心理测评问卷的题目列表及选项。（如果问卷内容固定且前端内置，则无需此接口。）响应包含一组问题，每个问题含题干、题号、可能的选项等。
- **提交测评结果接口** – **`POST /api/assessment/submit`**：提交用户对测评问卷的作答。请求中包含用户对应每一道题的答案。后端根据预设的算法计算得分和结果等级，将记录保存到 `assessments` 表。响应返回本次测评的结果摘要，例如得分和风险等级。此接口需要用户已登录（令牌验证），系统会根据令牌识别用户身份关联 `user_id`。
- **获取测评历史接口** – **`GET /api/assessment/history`**：（需登录）返回当前用户过往的心理测评记录列表。可支持分页或限制返回最近的N次结果。每条记录一般包含测评时间、分数、等级等摘要信息。通过此接口用户可以查看自身心理状态的变化趋势。

接口请求及响应示例：

- **提交测评结果示例**：用户完成问卷后提交答案获取评估结果。

请求:

```
POST /api/assessment/submit
Authorization: Bearer <token>
Content-Type: application/json

{
  "assessment_type": "stress_test",
  "answers": {
    "q1": 4,
    "q2": 3,
    "q3": 5,
    "...": "..."}
}
```

（上述请求中，`assessment_type`标识测评类型或量表名称，`answers`对象包含题目编号到作答的映射。例如`q1:4`表示第1题选择4分的选项。）

响应:

```
{
  "score": 78,
  "level": "中等风险",
  "advice": "您的压力指数中等偏高，请注意适当纾压和休息。",
  "assessment_id": 10
}
```

（响应给出本次测评的总体得分、风险等级评估`level`，以及简单的建议`advice`。`assessment_id`是该记录在数据库中的ID。）

- **获取测评历史示例**:

请求:

```
GET /api/assessment/history
Authorization: Bearer <token>
```

响应:

```
{
  "assessments": [
    {
      "id": 8,
      "date": "2025-10-01 15:30:00",
      "score": 65,
      "level": "中等风险"
    },
    {
      "id": 10,
      "date": "2025-10-15 10:20:00",
      "score": 78,
      "level": "中等风险"
    }
  ]
}
```

（响应返回一个测评记录列表，每项包含测评ID、日期时间、分数及等级。）

实现要点：测评题目的评分规则和结果计算逻辑在后端实现。例如，可以为每道选择题预设一个分值，汇总计算总分，并按照阈值划分风险等级。提交测评时，在后端完成分数计算和评语生成，并将结果存储。为提高性能，题目列表（questions）可以缓存在内存或者前端，以减少频繁从数据库读取。重要的是在提交接口对用户答案进行校验（例如题目数量是否匹配、答案选项是否有效）以确保数据可靠。

火场任务记录模块

模块职责：火场任务记录模块用于记录消防员（用户）执行过的救火任务信息。这有助于量化用户所经历的事件强度，用于后续风险评估（例如累积高强度任务可能提高心理风险）。该模块允许用户（或管理员）添加任务记录、查看个人任务记录列表，管理员也可查看所有人员的任务记录。

相关数据表：`task_records`（火场任务记录表）。每条记录关联一个用户以及具体任务细节。

主要接口：

- **新增任务记录接口** – `POST /api/tasks`：添加一条新的火场任务记录。需要提供任务的各项细节（日期、地点、描述、严重程度等）。普通用户一般只能添加属于自己的任务记录；管理员可以为任何用户添加或导入历史任务记录。请求体为JSON格式，包括必要字段。成功时返回创建的记录ID或成功标识。
- **获取任务记录列表接口** – `GET /api/tasks`：（需登录）获取当前用户的所有任务记录列表。普通用户调用时，返回该用户自己的任务记录；管理员调用时，可选择查看所有用户的任务记录（可通过查询参数过滤某个用户或按日期筛选）。列表可分页返回。
- **获取任务记录详情接口** – `GET /api/tasks/<id>`：（需登录）根据任务记录ID获取详细信息。普通用户仅能查看自己的记录详情，管理员可以查看任意用户的记录详情。返回内容包括任务的完整信息（地点、描述、参与人员、时长等详细字段）。
- **编辑/删除任务记录接口** – `PUT /api/tasks/<id>` 或 `DELETE /api/tasks/<id>`：对已有任务记录进行修改或删除。**（注意：** 可以限定只有管理员可以删除或修改记录，普通用户若误报或记录错误时可联系管理员修改，以保持数据可信。）

接口请求及响应示例：

- **新增任务记录示例**：消防员完成一次任务后添加记录。

请求:

```
POST /api/tasks
Authorization: Bearer <token>
Content-Type: application/json

{
  "task_date": "2025-10-20 14:30:00",
  "location": "江南区仓库火灾现场",
  "description": "参与仓库火灾扑灭，现场有危化品，无人员伤亡。",
  "severity": "较高",
  "casualties": 0,
  "duration": 120
}
```

响应:

```
{
  "message": "Task record created successfully",
```

```
  "task_id": 25
}
```

（上述请求提供了任务时间、地点、描述以及严重程度等信息。响应返回确认消息和新记录的ID。）

- **获取任务记录列表示例（普通用户）：**

请求:

```
GET /api/tasks
Authorization: Bearer <token>
```

响应:

```
{
  "tasks": [
    {
      "id": 21,
      "task_date": "2025-09-10 09:00:00",
      "location": "宁江区居民楼火灾",
      "severity": "一般",
      "duration": 45
    },
    {
      "id": 25,
      "task_date": "2025-10-20 14:30:00",
      "location": "江南区仓库火灾现场",
      "severity": "较高",
      "duration": 120
    }
  ]
}
```

（列表中每条任务包含基本信息字段，详细描述等可在获取详情时获取。）

实现要点：任务记录接口需要根据用户角色控制权限，例如普通用户只能访问和操作属于自己的记录，而管理员可以查看所有记录。可以通过查询参数支持筛选，如 `GET /api/tasks?user_id=5` 供管理员查询特定用户的任务。每次新增任务记录时，可在后台检查该记录是否需要触发风险评估（例如严重程度为“高”且有伤亡的任务可能立即触发高风险预警）。实现上，使用ORM保存记录非常直接，将JSON解析为任务对象后 `db.session.add()` 并提交。注意验证提交的数据格式和完整性，如日期格式是否正确、枚举值（严重程度）是否在允许范围等。

风险预警模块

模块职责：风险预警模块是系统的核心，用于综合分析用户的心理测评结果和任务记录数据，自动判别潜在的心理风险并产生预警信息。该模块的功能包括：根据预定规则计算风险等级、生成预警记录，提供接口让用户或管理员查看风险预警信息等。

相关数据表：`risk_warnings`（风险预警表），以及利用`assessments`和`task_records`作为输入数据源。预警的生成可能不直接通过接口触发，而是在后台逻辑中自动插入`risk_warnings`记录，但也提供只读接口查看预警内容。

主要接口：

- **获取当前用户预警接口** – `GET /api/risk/status`：（需登录）返回当前登录用户最近的风险预警情况。例如，返回用户当前的风险等级、最近一次预警消息及时间等。如果用户目前无任何预警，则返回安全或空状态。
- **管理员获取预警列表接口** – `GET /api/risk/warnings`：（需管理员权限）获取系统内所有用户的未处理风险预警列表。支持按风险等级或用户筛选。例如可以使用查询参数`?level=高`来只获取高风险预警，或`?user_id=5`获取指定用户的预警。管理员据此可以了解哪些用户需要关注。
- **标记预警为已处理接口** – `PUT /api/risk/warnings/<id>`：（管理员）更新指定ID的预警记录状态，将其标记为已处理（即将`resolved`字段置为true），并可附加备注（例如处理措施）。这样可以避免重复提醒，并记录处理情况。

接口请求及响应示例：

- **获取当前用户预警示例**：用户查询自己的心理风险状态。

请求：

```
GET /api/risk/status
Authorization: Bearer <token>
```

响应：

```
{
  "has_warning": true,
  "latest_warning": {
    "level": "高风险",
    "message": "您近期心理压力过高，请及时联系心理辅导员。",
    "trigger_time": "2025-10-21 08:30:00"
  }
}
```

（表示该用户目前有未处理的预警，等级高风险，包含一条预警信息和产生时间。如果`has_warning`为false，则表示当前没有预警。）

- **管理员获取预警列表示例**：

请求：

```
GET /api/risk/warnings?level=高
Authorization: Bearer <admin-token>
```

响应：

```
{
  "warnings": [
    {
      "id": 5,
      "user_id": 7,
      "level": "高风险",
      "message": "近期连续参加多次高强度任务，心理压力过大。",
      "trigger_time": "2025-10-20 17:00:00",
      "resolved": false
    },
    {
      "id": 6,
      "user_id": 12,
      "level": "高风险",
      "message": "最近一次心理测评得分过高。",
      "trigger_time": "2025-10-21 09:40:00",
      "resolved": false
    }
  ]
}
```

实现要点：风险预警的核心在于**规则引擎**或判断逻辑。系统可以在以下两种场景触发预警计算： 1. **实时触发**：当用户提交一次心理测评，或新增一条任务记录后，后台立即根据当前用户最新的所有数据计算其风险。如果满足预警条件则插入一条新预警记录。 2. **批处理触发**：每日定时任务扫描所有用户的数据，找出满足条件者生成预警（避免遗漏，作为实时触发的补充）。

预警判定逻辑可封装在一个函数中，例如 `evaluate_risk(user_id)`。伪代码示例如下：

```
def evaluate_risk(user_id):
    user = User.query.get(user_id)
    latest_assess = get_latest_assessment(user_id)
    recent_tasks = get_recent_tasks(user_id, within_days=30)
    risk_level = "低风险"
    reason = ""

    # 规则示例：根据测评得分判定风险等级
    if latest_assess.score >= 85:
        risk_level = "高风险"
        reason = "最近一次心理测评得分过高。"
    elif latest_assess.score >= 70:
        risk_level = "中风险"

    # 规则示例：高强度任务频繁
    high_severity_count = sum(1 for t in recent_tasks if t.severity == "高" or t.severity == "重大")
    if high_severity_count >= 3:
        risk_level = "高风险"
        reason += "近期连续参加多次高强度任务，心理压力过大。"
```

```
    # 如判定为高风险且没有重复预警，则记录预警
    if risk_level == "高风险":
        create_risk_warning(user_id, level=risk_level, message=reason)
```

上述逻辑只是示例，具体规则在实际应用中应更严谨和全面（可参考心理学专业建议）。一旦判定生成预警，`create_risk_warning` 会在数据库中插入记录。为避免重复预警，可以在插入前检查最近是否已有相同等级的未处理预警。

风险预警模块保证了当用户出现心理健康隐患时，系统能够及时记录并提示相关人员。通过接口，用户可自查自身风险，管理员可以统一查看所有预警并进行干预处理。

管理员报告模块

模块职责：管理员报告模块提供系统管理和数据汇总的接口，只允许具有管理员角色的账户访问。此模块帮助管理员了解系统整体状况，包括用户概览、测评统计、风险预警统计等，并提供必要的管理操作接口。

相关数据表：涉及所有前述表的数据汇总。例如从 `users` 获取用户数量，从 `assessments` 汇总平均得分，从 `risk_warnings` 获取预警数量等。本模块主要执行统计查询，因此通常不新增专门的表。

主要接口：

- **用户概览接口** – `GET /api/admin/users`：（管理员）获取所有用户列表及概要信息。可以返回用户ID、用户名、注册时间、最近一次测评结果、当前风险状态等汇总信息。用于管理员快速浏览所有用户的基本情况。
- **统计报表接口** – `GET /api/admin/statistics`：（管理员）返回系统整体数据统计报告，例如：总用户数、总测评次数、当前高风险用户数、中风险用户数、已处理预警数/未处理预警数、任务总数等。也可以细分统计，如按照日期统计最近30天内每日的测评次数、预警次数等。此接口便于在前端生成图表报表。
- **获取用户详细信息接口** – `GET /api/admin/users/<id>`：（管理员）查看特定用户的详细信息，包括该用户的注册资料、所有心理测评记录、所有任务记录及未处理的预警等综合数据。便于管理员全面评估某个用户的状况。
- **管理操作接口**：（按需设计）例如，`DELETE /api/admin/users/<id>` 删除用户账户，`POST /api/admin/notifications` 群发通知给某类用户等。如果需要，这些接口也放在管理员模块下，均需严格权限控制。

接口请求及响应示例：

- **统计报表示例**：管理员获取系统整体统计数据。

请求:

```
GET /api/admin/statistics
Authorization: Bearer <admin-token>
```

响应:

```
{
  "total_users": 50,
```

```
  "total_assessments": 120,
  "total_tasks": 200,
  "risk_summary": {
    "high_risk_users": 5,
    "medium_risk_users": 8,
    "low_risk_users": 37,
    "unresolved_warnings": 6,
    "resolved_warnings": 10
  },
  "recent_activity": {
    "assessments_last_7_days": [5, 8, 10, 7, 9, 4, 6],
    "warnings_last_7_days": [0, 1, 2, 1, 0, 0, 1]
  }
}
```


（上例中，`risk_summary` 汇总了不同风险级别的用户数量和预警处理情况，`recent_activity` 可用于绘制最近一周每日测评和预警发生数量的折线图。）

- **用户详细信息示例**：

请求：

```
GET /api/admin/users/7
Authorization: Bearer <admin-token>
```

响应：（部分内容示例）


```
{
  "user": {
    "id": 7,
    "username": "fireman_zhang",
    "role": "user",
    "email": "zhang@example.com",
    "created_at": "2025-07-15 10:00:00"
  },
  "assessments": [
    {"id": 15, "date": "2025-08-01", "score": 82, "level": "高风险"},
    {"id": 18, "date": "2025-09-10", "score": 76, "level": "中等风险"}
  ],
  "task_records": [
    {"id": 20, "date": "2025-09-12", "location": "XX工厂火灾", "severity": "重大"},
    {"id": 25, "date": "2025-10-05", "location": "YY仓库火灾", "severity": "较高"}
  ],
  "active_warnings": [
    {"id": 6, "level": "高风险", "message": "近期心理测评得分过高", "trigger_time": "2025-10-06",
"resolved": false}
  ]
}
```

实现要点：管理员模块的接口都必须严格验证管理员权限（例如在JWT的payload中检查 `role` 是否为admin）。这些接口主要执行只读操作（除了可能的管理操作如删除用户），应注意效率，可以利用SQL的聚合函数或ORM的聚合查询获取统计数据，避免在Python层面处理过多数据。如果数据量大，还需考虑加入缓存机制。例如统计报表可以每隔一定时间生成缓存，而非每次请求都实时计算。

安全机制

系统在设计和实现过程中采用多种安全机制，保障数据和接口的安全性：

- **身份验证与授权**：使用基于 **JWT（JSON Web Token）** 的身份验证机制。用户登录成功后，后端签发包含用户身份信息的JWT令牌，客户端需在后续请求中附加该令牌。后端在每次需要保护的接口请求时验证JWT的有效性和签名，确定用户身份和权限。对于管理员接口，还会验证令牌中的 `role` 是否为 `admin`，以授权访问。JWT通常设置一定的有效期（例如24小时），过期后需要用户重新登录获取新的令牌。

- **密码安全存储**：用户密码绝不以明文形式保存。注册时，后端使用安全哈希算法对密码进行散列（例如使用Python的Werkzeug库的 `generate_password_hash`，默认采用PBKDF2-HMAC-SHA256算法）。在登录验证时，使用对应的 `check_password_hash` 将用户输入的密码与存储的哈希进行比较。这样即使数据库泄露，攻击者也无法直接获得明文密码。

- **输入验证与错误处理**：对所有客户端输入的数据进行严格校验。例如，用户名、密码长度校验，邮箱格式校验；测评答案的格式校验（题目数量是否匹配、答案值是否在合法范围）；任务记录各字段类型和取值校验（日期格式、枚举值是否正确）等。一旦检测到非法或异常输入，后端会返回适当的错误码和信息，不盲目将异常堆栈返回给客户端，防止信息泄露。

- **防SQL注入**：由于本项目采用参数化查询或ORM（如SQLAlchemy），大部分情况下天然防止了SQL注入攻击。如果某些地方需要手写SQL语句，也必须使用占位符绑定参数的方式，杜绝将未经处理的用户输入直接拼接进SQL查询字符串。数据库层也可以通过最小权限原则，给予应用账号有限的数据库权限。

- **跨域与CSRF防护**：如果前端与后端部署在不同域名/端口，需要配置 **CORS（跨域资源共享）**。后端可使用Flask的 `flask-cors` 库来允许来自前端域名的请求。鉴于本项目主要提供API接口，且采用JWT防护，跨站请求伪造（CSRF）的风险相对较低，但仍需确保敏感操作只接受可信来源。对于纯后端API，可以在每次请求中要求使用令牌，以避免CSRF。

- **HTTPS 与传输安全**：部署时，强制使用 **HTTPS** 协议，以保证客户端与服务器通信加密，防止中间人窃听敏感信息（如登录密码、JWT令牌等）。同时，可以在JWT令牌中使用 `secure` 和 `httponly` 属性（如果令牌存放在cookie时）以提高安全性。

- **日志与审计**：系统记录关键操作日志，例如用户登录失败/成功日志、管理员操作日志等。通过日志，管理员可以审计可疑行为，如某账户短时间多次失败登录（可能存在密码爆破攻击）。可以针对这些行为启用**速率限制**（Rate Limiting），比如连续多次登录失败则暂时锁定账户或IP，防止暴力破解。

综上，这些安全机制共同保障了系统的安全性和稳定性，从用户认证到数据存储各环节尽可能减少漏洞风险。

风险预警规则

心理风险预警规则是系统的核心算法部分，它依据用户的心理测评结果和火场任务经历来判定用户的心理健康风险等级。制定规则时参考了专业经验和实际需求，力求及早发现高风险个体。主要的预警判定规则如下：

- **心理测评高危值**：如果用户最近一次心理测评得分非常高（接近满分，表示心理压力或症状严重，如得分 ≥ 85/100），则判定为**高风险**，立即触发预警。例如压力测试得分85分可能意味着严重压力，需要预警提醒【这一阈值可根据具体测评问卷的评分标准调整】。

- **连续中度异常**：如果用户连续多次测评结果均显示中等风险偏上（例如最近3次测评得分都在70分左右或以上），即使未达最高阈值，也认为累积效应显著，判定为**高风险**。此规则用于捕捉那些多次中等压力但未缓解的情况。

- **任务高负荷**：针对火场任务记录，若用户在短期内参与多次高强度任务，也可能导致心理风险升高。具体规则如：在过去30天内参与的任务中，有超过 **X 次** 严重等级（`severity="高"` 或 `"重大"`）的任务，则判定为**高风险**。例如规则设定 X=3，即一月内有3次重大火灾任务经历则触发预警。这个规则反映出频繁处置严重事故会累积压力。

- **重大单次事件**：如果某次任务记录特别重大（例如有人员伤亡，或事故造成重大影响），即使用户之前没有其他高风险迹象，也应触发预警。比如任务记录的 `casualties` 字段>0（有人员伤亡）或 `severity` 标记为最高等级，则直接产生**高风险**预警，提醒对该事件进行心理疏导跟进。

- **综合评分模型**：系统可设一个综合风险评分，将心理测评得分和任务负荷结合计算。例如：定义**综合风险分=最近测评得分 * 60% + 最近30天任务严重程度平均值 * 40%**。如果综合风险分超过某阈值（如80分），则视为高风险。这样能平衡考虑心理测评和现实任务因素。 (注：此模型视项目需要实现，可简单加权或使用更复杂的算法)。

- **风险等级划分**：根据规则结果，将风险划分为三级：低、中、高。低风险：各项指标均正常或轻微；中风险：有一定异常但未达阈值；高风险：符合预警条件，需要关注。对于中风险的用户，可在用户界面给予提示（例如提示注意休息），但不会在管理员预警列表中大张旗鼓出现，仅当高风险时才正式生成 `risk_warnings` 记录供管理员处理。

上述规则可以根据实际使用中的反馈不断调整优化。例如，引入更多维度（同事评价、睡眠数据等）可以丰富模型，但在本系统范围内主要聚焦于**心理测评**和**火场任务**两大来源的数据。规则阈值在初始设定时可以略偏严格（宁愿多报假阳性，也不漏报真风险），同时由专业心理辅导老师定期评估调整，以降低误报率。

当任一规则条件满足且判定为高风险时，系统将生成一条风险预警记录，并可以立即通知相关人员（例如通过应用内消息或邮件通知用户本人或管理员）。如果多个规则同时满足，也只生成一条预警但可能在 `message` 中罗列多项原因。管理员在查看预警时，应根据原因判断采取何种干预措施（如联系用户谈话、建议专业咨询等）。

数据流动

本节描述系统中数据在各模块之间流转的过程，包括普通用户使用功能时的数据交互流程，以及管理员使用后台功能时的数据交互流程。

普通用户流程

1. **用户注册**：新用户通过前端提交注册表单 -> 前端调用 `POST /api/auth/register` 将用户名、密码等发送至后端 -> 后端验证数据格式、创建用户记录于数据库（`users`表），返回注册成功消息 -> 前端通知用户注册成功（可能直接引导登录）。

2. **用户登录**：用户提交登录表单 -> 前端调用 `POST /api/auth/login` 将凭据发送 -> 后端验证用户名和密码：查询`users`表核对密码哈希 -> 验证成功则生成JWT令牌，返回给前端；失败则返回错误提示 -> 前端保存令牌（例如保存在内存、localStorage或cookie）。

3. **获取测评问卷**（可选）：用户准备开始一次心理测评 -> 前端调用 `GET /api/assessment/questions` 获取题目 -> 后端查询测评题库（或读取预定义题目）返回问卷JSON -> 前端渲染题目让用户作答。 (如果问卷已在前端固定，则无需此步网络请求)。

4. **提交心理测评**：用户完成问卷 -> 前端调用 `POST /api/assessment/submit` 提交答案（附上步骤2获得的JWT令牌用于认证） -> 后端收到请求，根据`user_id`（从令牌解码）识别用户，将答案解析计算得分和等级 -> 后端在`assessments`表插入新记录，并根据**风险预警规则**评估该用户当前风险：

   - 若满足高风险条件，则在`risk_warnings`表插入预警记录；
   - 否则无进一步操作。

   后端将测评结果（分数、建议等）返回给前端 -> 前端显示结果给用户，并可能给予简要建议。

5. **查看历史测评**：用户需要查看自己以往的心理状态变化 -> 前端调用 `GET /api/assessment/history`（附令牌）-> 后端验证令牌后查询`assessments`表筛选该用户的记录列表 -> 返回给前端 -> 前端列表展示每次测评的分数和等级，供用户参考。

6. **记录火场任务**：用户参与一次新的消防任务后，希望记录该经历 -> 前端调用 `POST /api/tasks` 提交任务信息（时间、描述等，附令牌）-> 后端验证身份，将任务数据写入`task_records`表 -> 写入完成后，后端根据此任务数据再次触发**风险评估**（例如如果任务严重程度高，调用`evaluate_risk`检查该用户是否因此达到预警条件）：

   - 若是，这时插入新的`risk_warnings`记录（并可实时通知管理员有新预警产生）；
   - 若否，则不产生新预警。

   后端返回记录成功消息 -> 前端提示用户记录已保存。

7. **查看个人预警状态**：用户可以随时查询自己的风险状态 -> 前端调用 `GET /api/risk/status`（附令牌）-> 后端从`risk_warnings`表检索该用户最新的未处理预警 -> 将结果返回给前端 -> 前端如检测到高风险，可以提醒用户联系管理员或心理辅导员。如果没有预警则显示“状态正常”之类的信息。

上述流程中，普通用户始终在前端与后端通过API进行交互，后端执行相应的数据库读写操作并应用业务逻辑。所有与用户相关的请求都带有JWT身份令牌确保安全；同时，每当有新的测评或任务输入时，风险评估逻辑自动运行，保证预警信息及时更新。数据在 **前端 -> 后端 -> 数据库** 之间流动，并通过后端处理后 **数据库 -> 后端 -> 前端** 返回结果，实现闭环。

管理员流程

1. **管理员登录**：管理员账户通过与普通用户相同的登录接口登录 -> 后端返回JWT令牌，其中包含管理员角色信息。管理员前端保存该令牌用于后续管理接口调用。

2. **查看用户列表**：管理员进入后台管理界面 -> 前端调用 `GET /api/admin/users`（附管理员令牌）-> 后端验证令牌和角色后，查询`users`表获取所有用户基本信息，可能还进一步汇总每个用户最新测评或预

警状态以一并返回 -> 前端收到数据，在管理界面生成用户列表（例如表格形式展示用户名、注册时间、当前风险等级等）。

3. **查看总体统计**：管理员调用 `GET /api/admin/statistics` 接口获取系统全局统计数据 -> 后端执行多项统计查询：如COUNT用户数、COUNT测评数、COUNT预警等，以及按需的时间序列统计 -> 返回JSON数据 -> 前端将数据以图表或报告形式展示给管理员（如折线图、饼图等）。

4. **查看具体预警**：管理员在后台查看风险预警 -> 前端调用 `GET /api/risk/warnings` （附管理员令牌）-> 后端返回当前所有未处理的预警列表（可能已经按高风险优先排序）-> 管理员浏览列表，针对每条预警可以选择查看详情或标记处理。

5. 如果管理员点击某条预警想了解详情，前端可进一步调用 `GET /api/admin/users/<id>` 获取该用户详细信息（包括历史测评、任务列表）以供深入分析。

6. 如果管理员对预警采取了干预措施（比如联系该用户谈话），可通过 `PUT /api/risk/warnings/<id>` 将此预警标记为已处理并附加备注 -> 后端收到请求更新数据库中的预警记录（将 `resolved` 设为true并保存备注）-> 前端界面相应更新，不再将该预警显示为待处理。

7. **管理操作**：管理员可能执行其他管理功能，例如删除不良用户、导出报告、发送通知等。

8. **删除用户**：前端调用 `DELETE /api/admin/users/<id>` -> 后端验证管理员权限后，删除 `users` 表中对应记录，并连带删除或标记该用户相关的测评、任务数据（根据业务需求决定是物理删除还是逻辑删除）。操作完成后返回成功消息 -> 前端刷新用户列表。
9. **发送通知**：管理员编写通知内容并选择目标用户范围 -> 前端调用 `POST /api/admin/notifications` 提交通知 -> 后端验证权限后，将通知内容存储并推送给相应用户（这可能涉及额外的通知机制，不在此详述）。

在上述管理员交互中，数据主要在 **后端 -> 前端** 单向流动（管理员获取数据查看），少部分操作涉及 **前端 -> 后端**（如标记预警、删除用户），这些都会引起数据库状态的改变。管理员的操作往往是审阅和统计，因此后端侧重提供聚合后的数据，减少前端的处理负担。由于管理员权限高，系统对这些接口的安全要求也更严格，每次请求都会验证令牌确保请求者确实是管理员，并防范普通用户伪造请求访问。

综上，“心盾·智护”后端系统通过清晰的模块划分和接口设计，确保了用户与管理员各自所需功能的实现。数据在系统中的流动是合理且安全的：用户数据从采集（注册、测评、任务）到存储，再到分析（风险评估）和反馈（预警通知），形成闭环；管理员可以从宏观和微观两方面监控系统数据，及时发现问题并采取行动。此技术文档为开发和维护人员提供了全面的参考，有助于在实际部署和后续迭代中保持系统的一致性和可靠性。
