数据科学与计算机学院

数据库课程设计报告

姓名 杨麒平

学号 15336220

专业 信息与计算科学

完成日期： 2018 年 1 月 14 日

目 录

第 1 章

艇员管理系统

艇员管理系统是一个能够发布训练计划，统计训练记录的系统。并能通过统计得到的训练记录进行反馈，为队员提供参考。一个体育团队通过收集每天的训练情况来定期给队员提供积极科学的反馈，并以此能够修正训练计划，从而使训练更加高效有序。可是现实团队里面在即时通讯软件（wechat）里面发布和收集训练记录，或者说用纸来记录数据。这样的做法费时费力而且数据也不完整，即使通信软件里面有相当强大的搜索功能。但是如果每次进行数据收集都是对聊天记录进行搜索的话，数据难免丢失（数据格式不一致等原因）因此开发一个数据库应用来进行队内事务管理是相当有必要的。

§1.1 开发环境与开发工具

1. GNU Emacs 25.3.2
2. zsh 5.1.1
3. mysql Ver 14.14 Distrib 5.7.20, for Linux (x86_64) using EditLine wrapper
4. Python 3.5.2
5. Python Flask Framework
6. Python Main Packages: virtualenv, PyMySQL, SQLAlchemy, WTForms

§1.2 系统需求分析

1. 系统的用户管理
   包括用户的添加、删除、修改信息、修改密码等。

   一个用户对应一个成员，有唯一的用户名作为标识符，需要密码进行登录。

2. 艇员信息管理
   艇员信息的添加、删除、修改、查询。

   一个成员有自己的各种信息，对应唯一一个账户，有自己的训练级别，还有多条自己的训练记录，有多条对应自己的开支，由 (ID) 唯一标识。

3. 训练计划管理
   训练计划的添加，修改，删除，查询。

   一个训练计划包含训练时间，训练时长，训练项目，训练指标，还有训练对应的级别，由 (plan_ID) 唯一标识，需要和成员联系，作为计划制定者。

4. 训练记录管理
   训练记录的添加，修改，删除，查询。

   一个训练记录(与训练计划对称),有训练时间,训练时长,训练项目,训练结果,由 (record_ID) 唯一标识，需要和成员联系，作为训练者。

5. 船艇信息
   艇的信息添加，修改，删除，查询。

   一艘艇有名字，有类型，由 (ship_ID) 唯一标识。

6. 开支信息
   开支记录的添加，修改，删除，查询。

   一条开支，有时间，有备注，需要和成员联系，作为支出的对象，由 (fee_ID) 唯一标识。

§1.3 功能需求分析

艇员管理系统按照上面所述，管理功能主要在艇员信息、训练计划和训练记录的管理上面。下面还有一些主要的功能说明

1. 其中用户分为两个级别。普通用户级别和特权用户级别。特权用户级别的操作是普通用户级别的超集。

2. 其中添加、删除操作输入特权用户级别，AKA 只有特权用户能够添加新的成员。而每个用户需要自己的艇员信息被录入才能注册新的用户。每个用户能够对自己的信息进行查询，对部分信息能够进行修改。

3. 其中添加，修改，删除属于特权用户级别。

4. 每个用户只能对自己的训练记录进行 (CUD–Create, Update, Delete)。每个用户可以查询其他用户的训练记录。

5. 普通用户只能部分修改，能够查询。

§1.4 系统设计

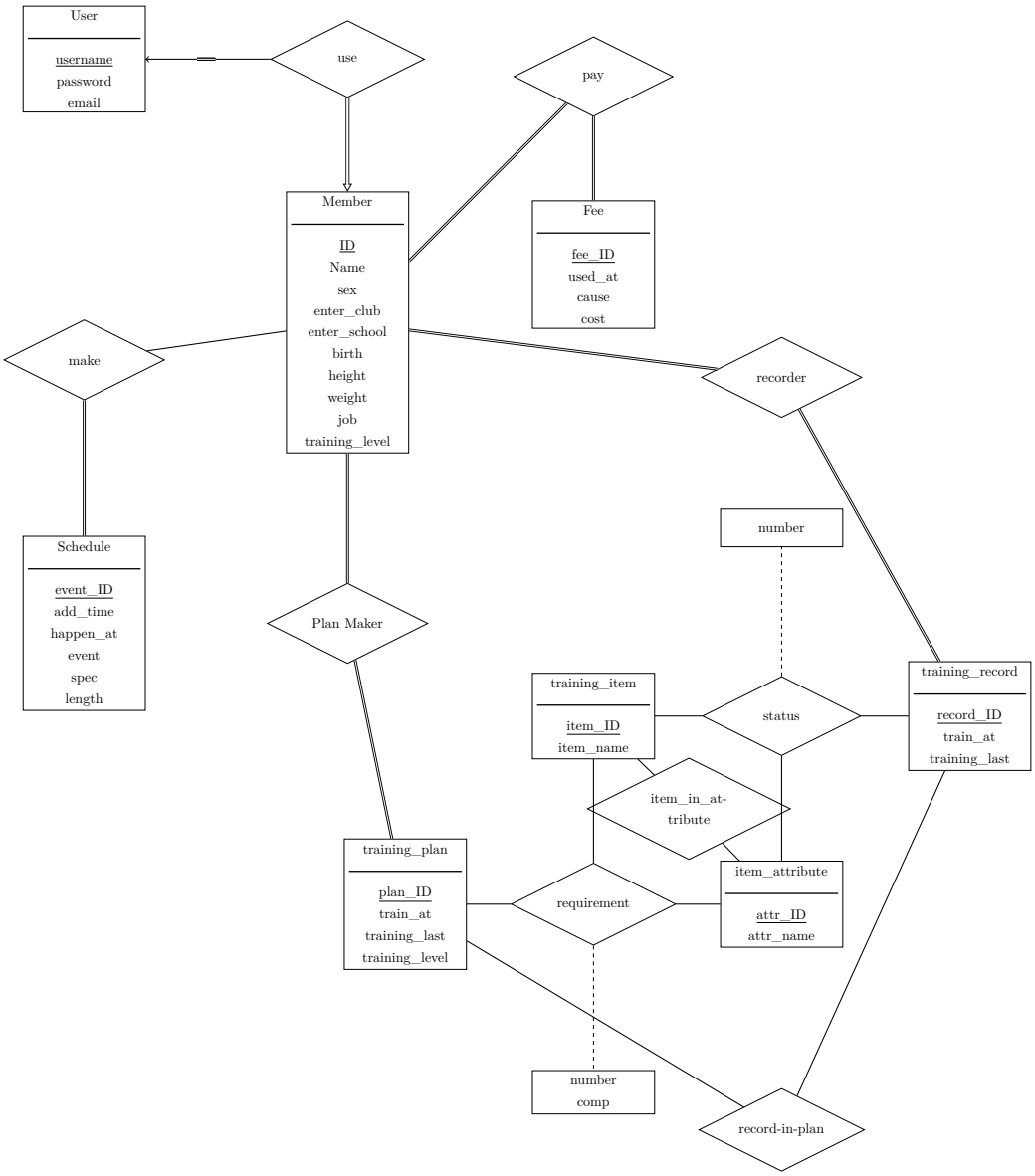
§1.4.1 ER 图的说明

1. 每个队员都只能拥有对应 ID 的一个账号，其实这里为什么不把 User 表和 Member 表合并：因为管理员在添加一个新的队员的时候需要先把 ID 输入，然后才能让用户注册，管理员能添加队员，也让队员能自己创建账号。

2. 另一个主要的设计就在于训练计划以及训练记录如何表达。一个训练计划有一个训练时间，一个训练时长，一个训练对应的级别，多个训练项目（多对多），一个训练项目有多个不同的指标（多对多）。因此，建立三个实体集，分别为 training_plan, training_item, item_attribute 以及一个三元联系，其中 requirement 还有联系集属性。项目和指标之间存在一个多对多的关系。在设计中我发现训练计划和训练记录实际上是对称的，因此在 ER 图中，我也是按着对称来画的。

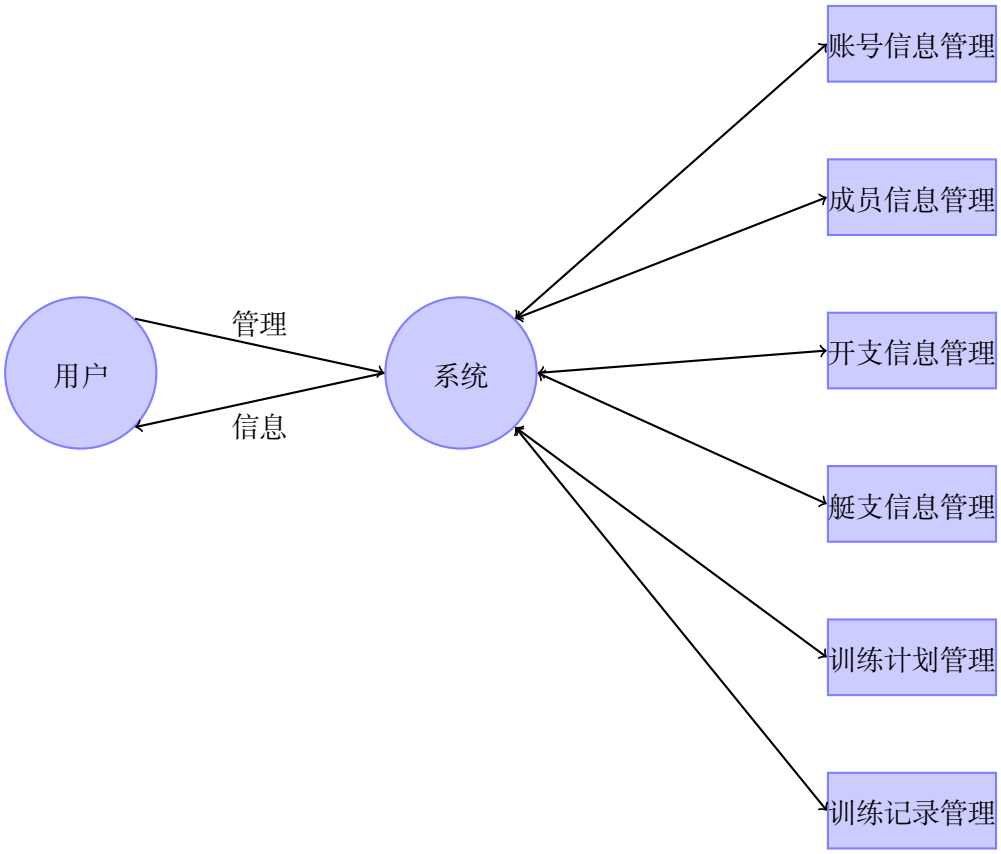
3. 训练和记录之间还有一个联系，而且记录不是全参与，因为一个训练记录可以是不按照训练计划的

§1.4.2 系统 ER 图



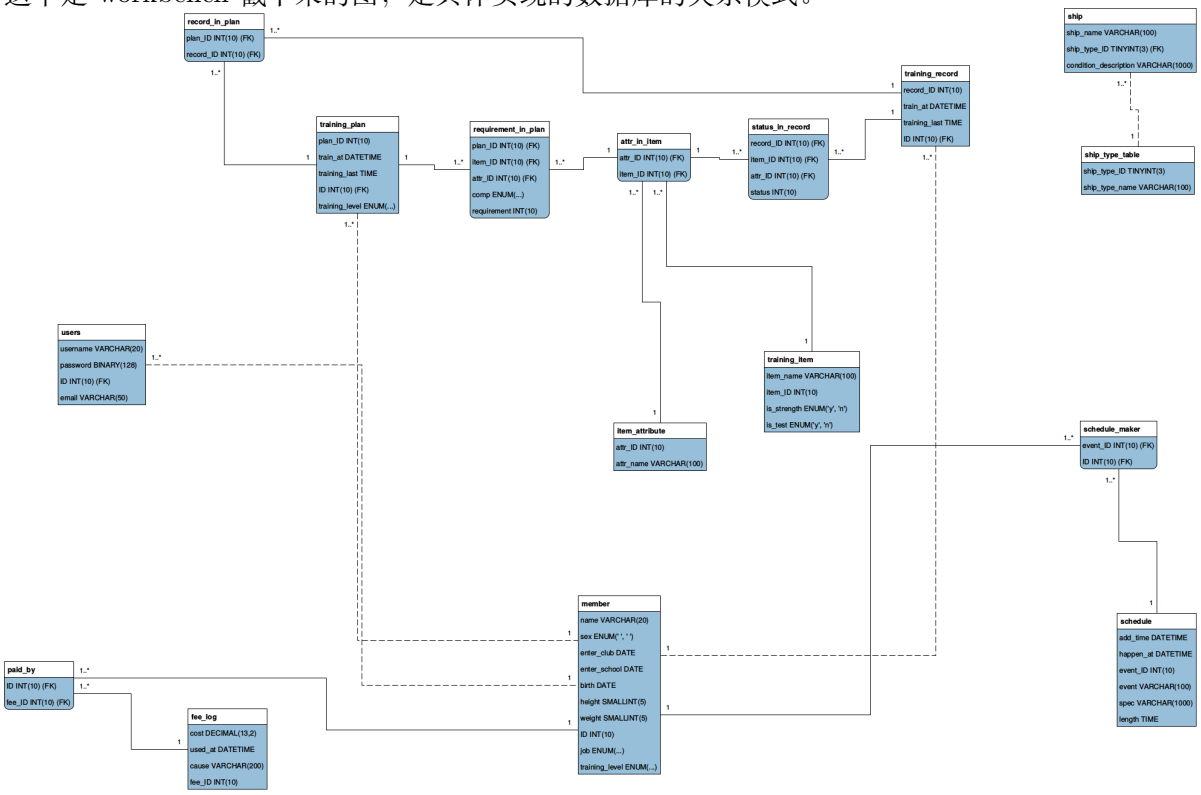
§1.4.3 数据流程图

基本上参照课本的设计图，其中系统层就是我们设计的网页应用。他简化了用户和数据库的交互，并提供一定的等级分层，以及维护系统的数据完整性。

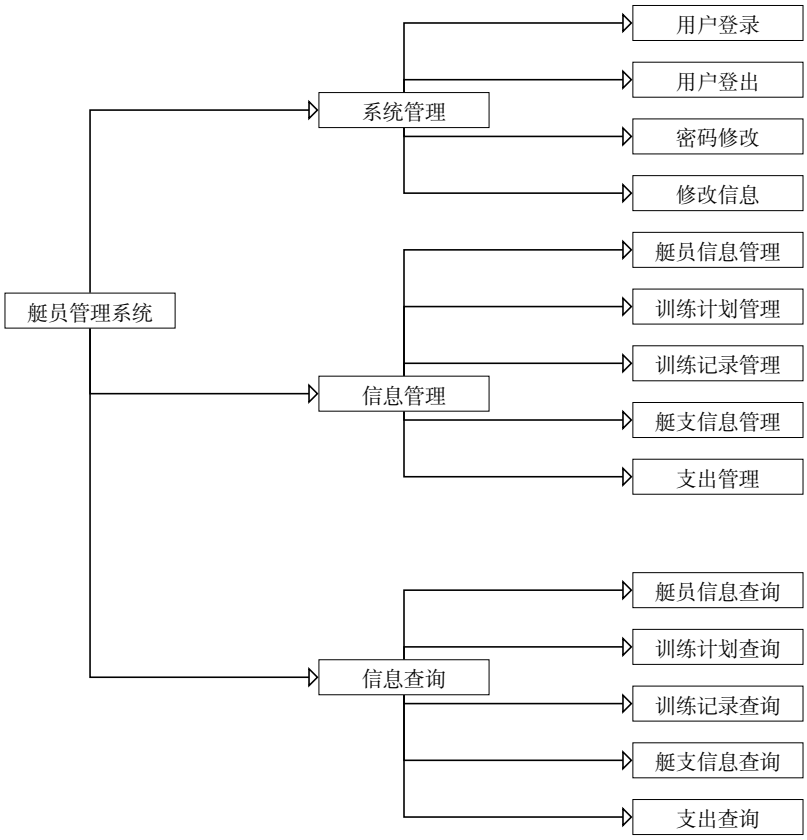


§1.4.4 数据库关系模式设计

这个是 workbench 截下来的图，是具体实现的数据库的关系模式。



§1.4.5 系统功能模块图



§1.4.6 物理结构设计

```
-- with debug if and if not
drop database if exists crewmen;
create database crewmen character set utf8 collate utf8_general_ci;
use crewmen;

drop table if exists users;

drop table if exists schedule_maker;
drop table if exists paid_by;
drop table if exists schedule;
drop table if exists fee_log;

drop table if exists record_in_plan;
drop table if exists requirement_in_plan;
drop table if exists status_in_record;

drop table if exists attr_in_item; -- referenced by requirement and status

drop table if exists training_record; -- by status by record_in_plan
drop table if exists training_plan; -- by requirment by record_in_plan
drop table if exists item_attribute; -- referenced by attr_in_item
drop table if exists training_item; -- referenced by attr_in_item
```

```
drop table if exists member; -- referenced by item plan record paid_by schedule_maker users
drop table if exists ship;
drop table if exists ship_type_table;


create table if not exists member
       (name varchar(20) not null,
       -- big chunk for information
       sex enum(' 男', ' 女') default ' 男',
       enter_club date, -- the crew
       enter_school date, -- school
       birth date,
       height smallint unsigned,
       weight smallint unsigned,
       ID int unsigned,
       -- big chunk for information
       job enum('couch', 'crew leader', 'crew member') default 'crew member',
       training_level enum('newbie', 'medium', 'old bird') default 'newbie',
       primary key(ID)
       );-- character set utf8 collate utf8_general_ci;

create table if not exists users
       (username varchar(20) not null,
       password binary(128) not null, -- half byte each, so 128 * 0.5 * 8 is 512 using sha2('pass', 512)
       -- priority tinyint unsigned not null defalut 2, -- the lower the better
       ID int unsigned, -- don't have to be unique, since it's PK for member
       email varchar(50) not null unique,
       primary key(username),
       foreign key(ID) references member(ID) on delete cascade
       );

-- how to enforce constraint on full participation, every schedule should have at least a maker...
create table if not exists schedule
       (add_time datetime default current_timestamp,
       happen_at datetime, -- time could be unsure of
       event_ID int unsigned auto_increment,
       event varchar(100) not null,
       spec varchar(1000),
       length time,
       primary key(event_ID)
       );

create table if not exists schedule_maker
       (event_ID int unsigned,
       ID int unsigned,
       primary key(event_ID, ID),
       foreign key(event_ID) references schedule(event_ID),
       foreign key(ID) references member(ID) on delete cascade
       );
```

```
-- same question as schedule, every fee should have a payer
create table if not exists fee_log
       (cost decimal(13,2), -- cost could range from cent to 10^13 which is 10^4 billion yuan, should be enough
        used_at datetime,
        cause varchar(200),
        fee_ID int unsigned,
        primary key(fee_ID)
       );

create table if not exists paid_by
       (ID int unsigned,
       fee_ID int unsigned,
       primary key(ID, fee_ID),
       foreign key(ID) references member(ID) on delete cascade,
       foreign key(fee_ID) references fee_log(fee_ID)
       );

-- plan and record session!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
-- plan and record session!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
-- plan and record session!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
-- plan and record session!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

-- It's divided into three main parts:
-- The plan and record are symmetrical
-- ignore! no longer supported (although plan can have multiple makes?? weird functionality though)


-- The main similarity lies in the relation of (record, item, attribute, status/requirement)
-- when it indicates that a record can have multiple items
-- when an item can have multiple attributes
-- Therefore this relation can be able to indicate a collection of status of a collection of training items in

-- notice how attr_in_item is referenced by (requirement_in_plan and status_in_record)
-- may need to define training item
create table if not exists training_item
       (item_name varchar(100),
       item_ID int unsigned auto_increment,
       is_strength enum('y', 'n'), -- strength or aerobic
       is_test enum('y', 'n'),     -- test or regular
       primary key(item_ID)
       );

create table if not exists training_record
       (record_ID int unsigned auto_increment,
       train_at datetime,
       training_last time,
       ID int unsigned,
```

```
       primary key(record_ID),
       foreign key(ID) references member(ID)
       );

create table if not exists training_plan
       (
       plan_ID int unsigned auto_increment,
       train_at datetime,
       training_last time,
       ID int unsigned,
       training_level enum('newbie', 'medium', 'old bird', 'all') default 'all',
       primary key(plan_ID),
       foreign key(ID) references member(ID)
       );

create table if not exists item_attribute
       (attr_ID int unsigned,
       attr_name varchar(100) unique,
       primary key(attr_ID)
       );

create table if not exists attr_in_item
       (attr_ID int unsigned,
       item_ID int unsigned,
       primary key(attr_ID, item_ID),
       foreign key(attr_ID) references item_attribute(attr_ID),
       foreign key(item_ID) references training_item(item_ID)
       );

create table if not exists requirement_in_plan
       (
       plan_ID int unsigned,
       item_ID int unsigned,
       attr_ID int unsigned,
       comp enum('larger', 'smaller', 'no requirement') not null default 'no requirement',
       requirement int unsigned,
       primary key(plan_ID, item_ID, attr_ID),
       foreign key(item_ID, attr_ID) references attr_in_item(item_ID, attr_ID),
       foreign key(plan_ID) references training_plan(plan_ID)
       );

create table if not exists status_in_record
       (record_ID int unsigned,
       item_ID int unsigned,
       attr_ID int unsigned,
       status int unsigned,
       primary key(record_ID, item_ID, attr_ID),
       foreign key (item_ID, attr_ID) references attr_in_item(item_ID, attr_ID),
       foreign key (record_ID) references training_record(record_ID)
```

```
       );

-- below is record and plan relation
create table if not exists record_in_plan
       (plan_ID int unsigned,
       record_ID int unsigned,
       primary key(plan_ID, record_ID),
       foreign key(plan_ID) references training_plan(plan_ID),
       foreign key(record_ID) references training_record(record_ID)
       );

-- Obsoleted
-- create table if not exists plan_maker
--        (ID int unsigned,
--        plan_ID int unsigned,
--        primary key(ID, plan_ID),
--        foreign key(ID) references member(ID) on delete cascade,
--        foreign key(plan_ID) references training_plan(plan_ID)
--        );

-- plan and record session!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
-- plan and record session!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
-- plan and record session!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
-- plan and record session!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

create table if not exists ship_type_table
       (ship_type_ID tinyint unsigned,
       ship_type_name varchar(100),
       primary key(ship_type_ID)
       );

create table if not exists ship
       (ship_name varchar(100),
       ship_type_ID tinyint unsigned,
       condition_description varchar(1000),
       primary key(ship_name),
       foreign key(ship_type_ID) references ship_type_table(ship_type_ID)
       );
```

§1.5 系统功能的实现

源码 github： https://github.com/326623/crew-system

这个项目最主要的模块我认为是训练计划和训练记录，还有用户权限的实现，但是其中还有挺多实现的细节。下面是项目的一个结构图。其中根目录下面的 crewmen.py 是 app 的主体，这个地方声明了 app 读入了配置，并从项目中 (project) 读入各个项目的信息 (blueprint)。project 目录下面是每个模块的具体代码，用一个独立的包来表示具体实现是参考了 (flask-blueprint) 的。另一个比较主要的实现技巧就是用 ORM(object relational mapping)，主要思想就是把数据库的操作封

装成一个类。在这里我用的是第三个的包叫做 **SqlAlchemy**。

config.py 就是配置文件，用一个类来包装，再由 crewmen.py 读入。

crew-management.sql 是数据库的 sql 实现。

crew-user.sql 是用户权限的设置。

test-data.sql 是初始的数据库的测试数据。

models.py 是 orm 的封装的类的包。每个子项目如果想要和数据库交互都要调用它。

每个子项目 (project) 都有一个 templates 文件，这个是存放 html 的。根目录下还有 core.py 和 form.py 分别是项目模块对应的 url 的处理函数和处理表单的代码。

```
    crewmen git:(master)    tree -d
.
├── project
│   ├── fee
│   │   └── templates
│   ├── member
│   │   └── templates
│   ├── training
│   │   └── templates
│   └── user
│       └── templates
└── templates
```

```
  .
├── config.py
├── crew-management.sql
├── crewmen.py
├── crew-user.sql
├── models.py
├── project
│   ├── fee
│   │   ├── core.py
│   │   ├── form.py
│   │   ├── __init__.py
│   │   └── templates
│   │       ├── add_fee_log.html
│   │       └── show_fee_log.html
│   ├── __init__.py
│   ├── member
│   │   ├── core.py
│   │   ├── form.py
│   │   ├── __init__.py
│   │   └── templates
│   │       ├── add_member.html
│   │       ├── allmember_profile.html
│   │       ├── delete_member.html
│   │       └── member_profile.html
│   ├── training
│   │   ├── core.py
│   │   ├── form.py
│   │   ├── __init__.py
```

```
|    |    └─── templates
|    |        ├─── add_item.html
|    |        ├─── add_plan.html
|    |        ├─── add_plan_item.html
|    |        ├─── show_item.html
|    |        └─── training_plan.html
|    └─── user
|        ├─── core.py
|        ├─── form.py
|        ├─── __init__.py
|        └─── templates
|            ├─── login.html
|            ├─── password_update.html
|            └─── register.html
├─── requirements.txt
├─── templates
|    ├─── base.html
|    ├─── home.html
|    └─── welcome.html
└─── test-data.sql
```

§1.5.1 用户系统

用户登录 提供用户名和密码的表单。数据库储存密码这个设计利用的是 sha-512 的单向 hash 方式储存。以后估计会改用 bcrypt。因此每次收到密码，在后端要把字符串用 hash 得到对应的编码与数据库相应的用户名的密码进行比对。如果正确，从数据库中的成员 (member) 信息读出权限，并存放到 session 里面，并把用户的登录状态还有用户名存放到 session 里面。

以下是主要的代码：

```
@user_blueprint.route('/login', methods=['GET', 'POST'])
def login():
"""
If the user doesn't exist, return error accordingly, if the user exists, with wrong password, return error
if successfully logged in, query db for ID and priority, store in session
"""
error = None
try:
if request.method == 'POST':
username = request.form['username']
password = request.form['password']

login_user = User.query.filter_by(username=username).first()

db_hash = login_user.password.decode("utf8")

hashcode = hashing.hash_value(password)
```

```
if hashing.check_value(db_hash, password):
session['login_ID'] = login_user.ID
session['logged_in'] = True
session['login_job'] = login_user.member.job
session['login_username'] = login_user.username
return redirect(url_for('home'))
else:
error = 'Invalid credentials. Please try again'
else:
return render_template('login.html')

except AttributeError:
error = 'This user is not registered!'

return render_template('login.html', error=error)
```

导航栏的动态显示，以及试图通过 url 访问普通权限无法访问的页面

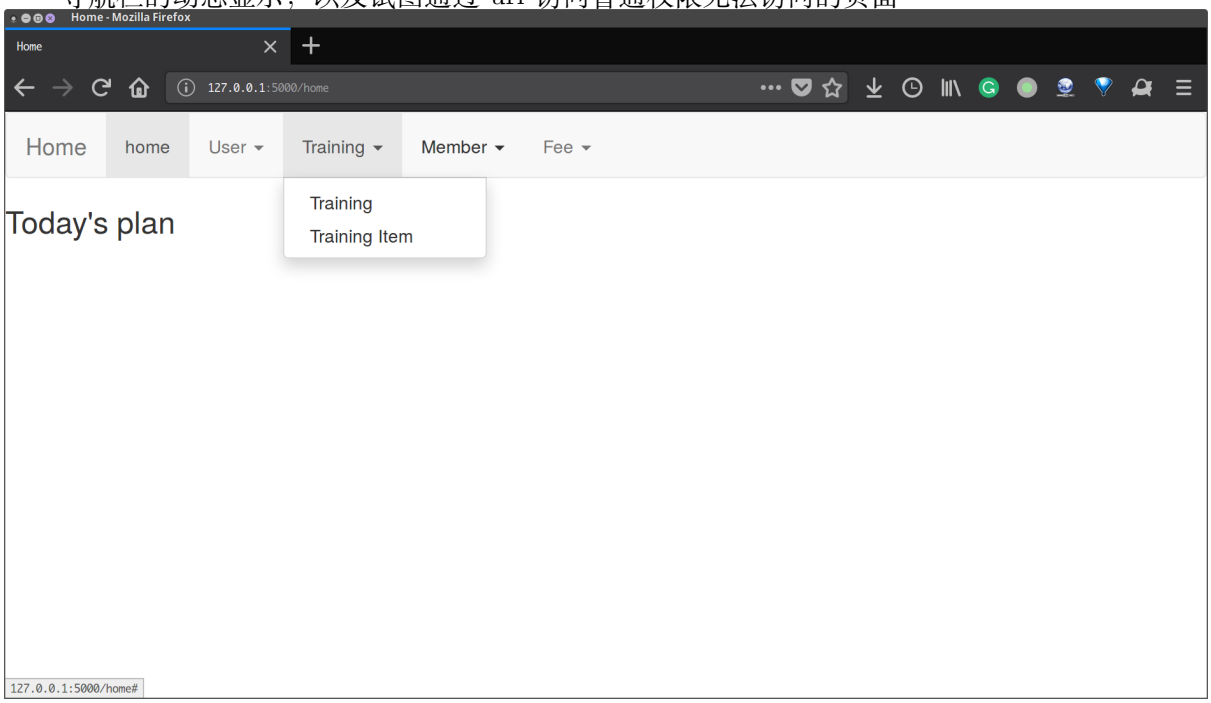


图 1.5-1 普通的权限看到的导航栏

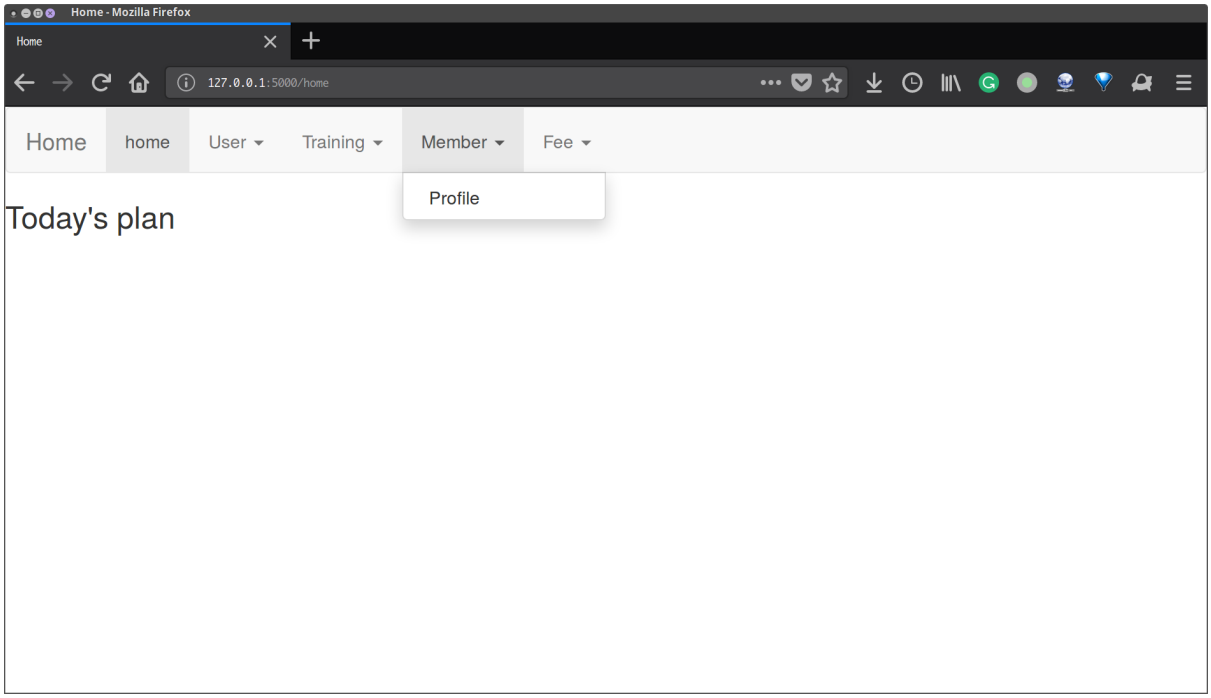


图 1.5-2 普通的权限看到的导航栏

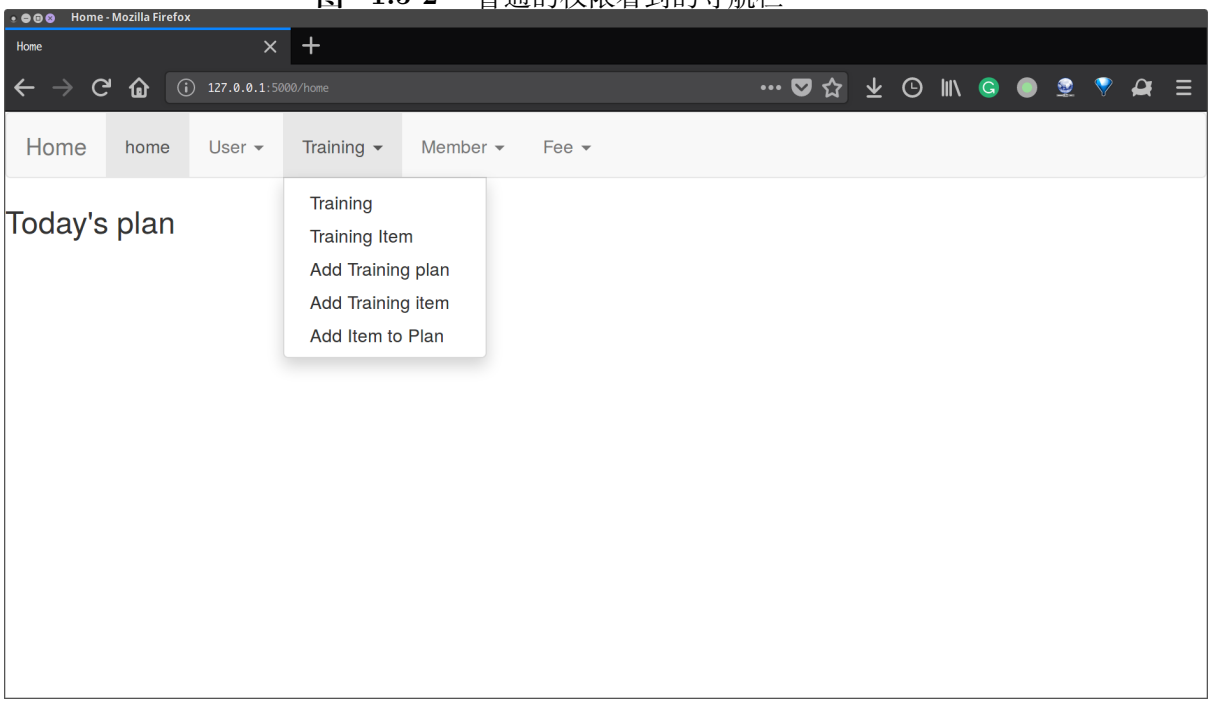


图 1.5-3 特权的导航栏

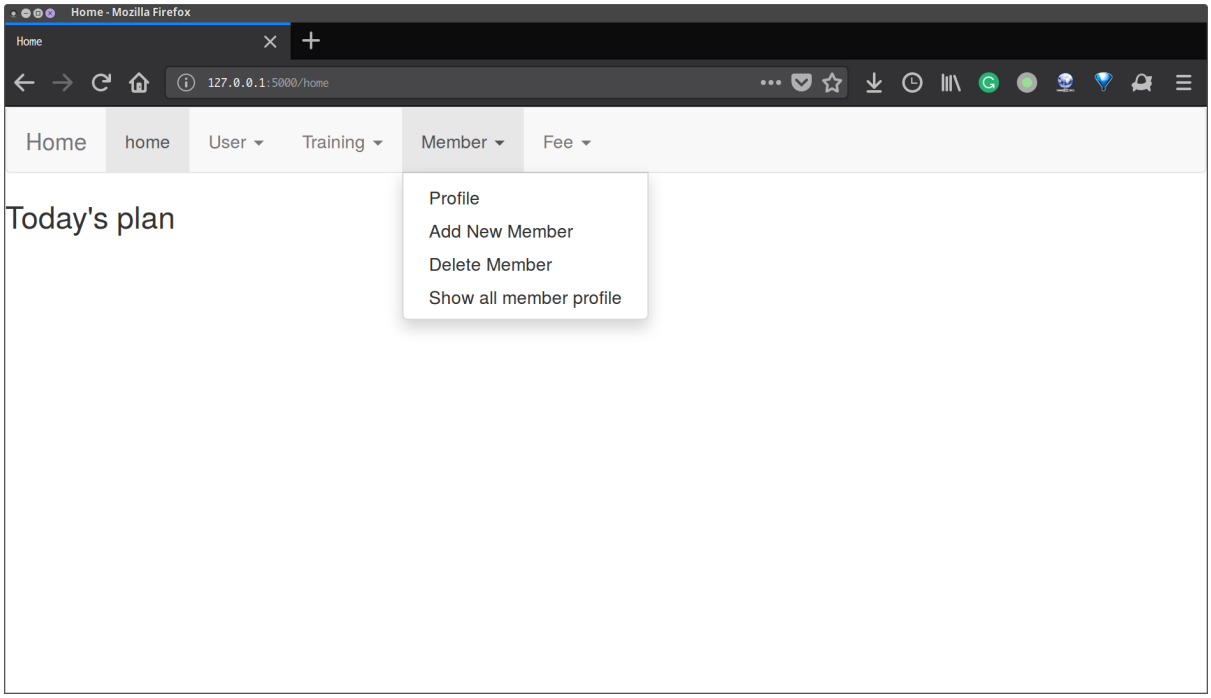


图 1.5-4 特权的导航栏

特权机制 对于某些页面进行限制，比如添加新成员，添加修改训练计划等。如果用户访问受限制的 url，先从登录后存放的 session 中找到对应的权限。如果权限不足，返回重定向，如果权限足够，返回对应的页面。这里用到了 python 的修饰器，其主要的思想就是把一个函数传到修饰器对应的函数 power_required 里面去，在这里修饰 power_required 如果发现用户的权限不足就会重定向到 home 的 url 上，如果权限足够就会把控制权交回给需要权限才能访问的函数。

以下是主要的代码：

```
def power_required(f):
    @wraps(f)
    def wrap(*arg, **kwargs):
        job = session['login_job']
        if job == 'crew leader' or job == 'couch':
            return f(*arg, **kwargs)
        else:
            flash('You have no power.')
            return redirect(url_for('home'))
    return wrap
```

§1.5.2 训练计划

计划查询 每个人都能访问今日的训练计划。

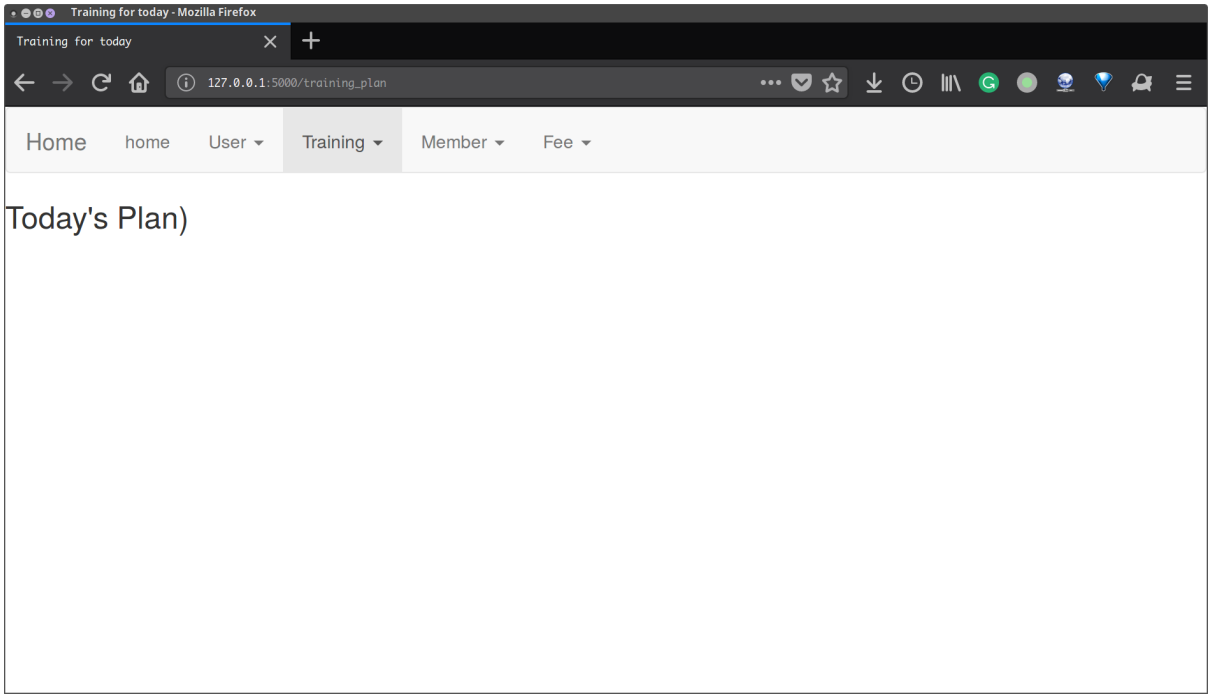


图 1.5-5 本日的计划

图 1.5-6 添加计划

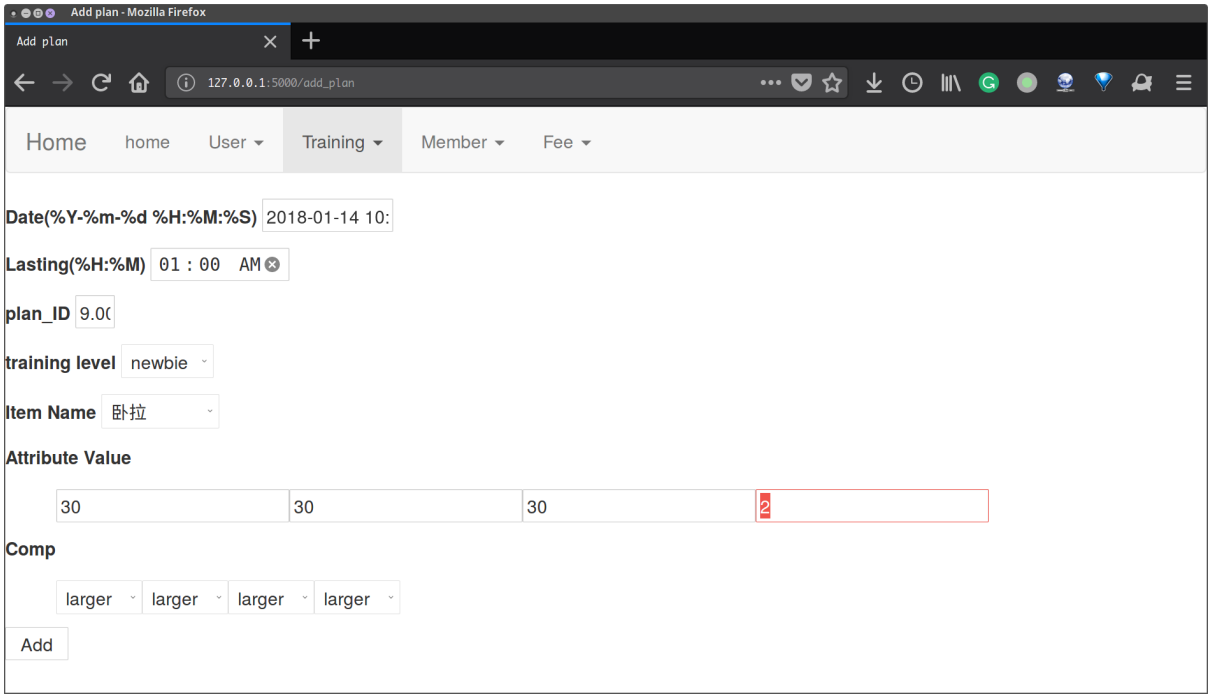


图 1.5-7 计划具体

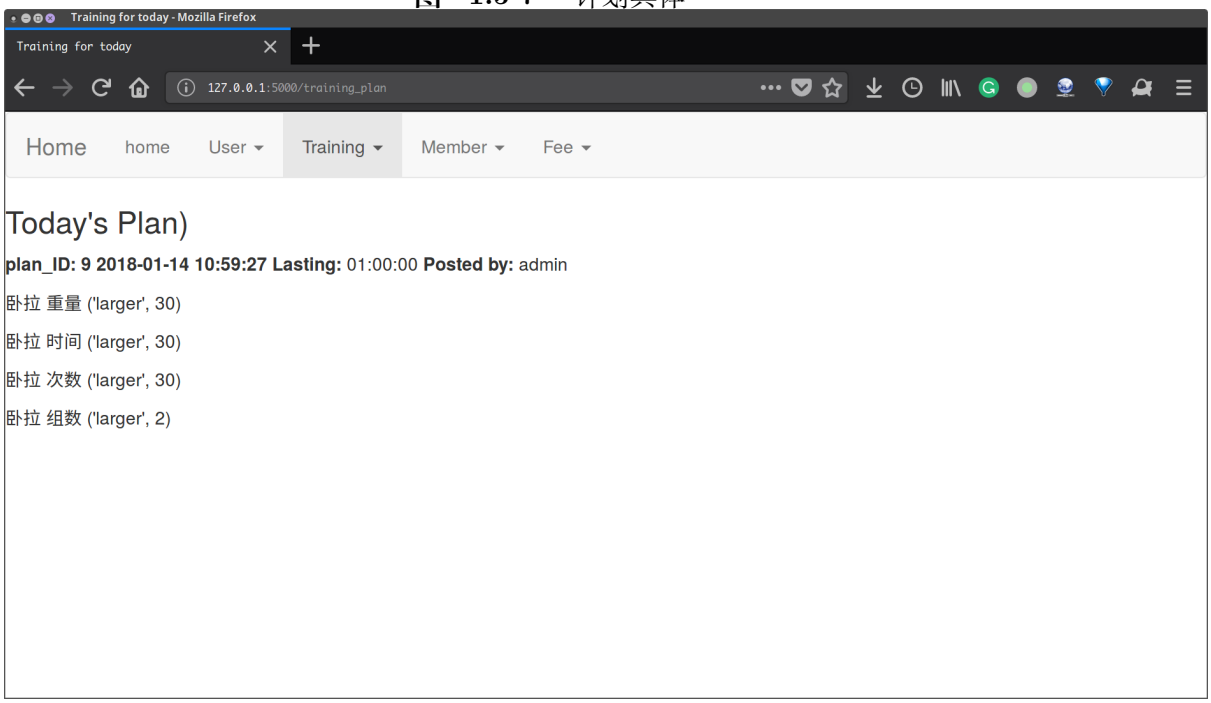


图 1.5-8 添加之后

在数据库中查找今天的训练计划。以下是实现部分的源代码：

```
@training_blueprint.route('/training_plan')
@login_required
def show_training_plan():
    today = datetime.date.today()
    nextday = today + datetime.timedelta(days=1)
    today_plan = TrainingPlan.query.filter(TrainingPlan.train_at >= today,
                                           TrainingPlan.train_at < nextday).all()
    return render_template('training_plan.html', plans=today_plan)
```

更改计划 只有队长或者教练能够修改，添加或者删除训练计划。对于返回的表单进行检查。

下面源码中在 add_plan_item 函数上面添加修饰函数@power_required 就能在真正调用网页之前先检查权限。下面的实现还有不完美的地方，其中之一就是对于异常的处理不过关。没有检查传进来的数据的可靠性。

在插入项目及其相应的指标的时候，可以发现挑选一个项目会有其对应的指标的表格出现，这里就运用了 AJAX，Asynchronous JavaScript And XML，异步的 javascript 和数据库交互返回结果。

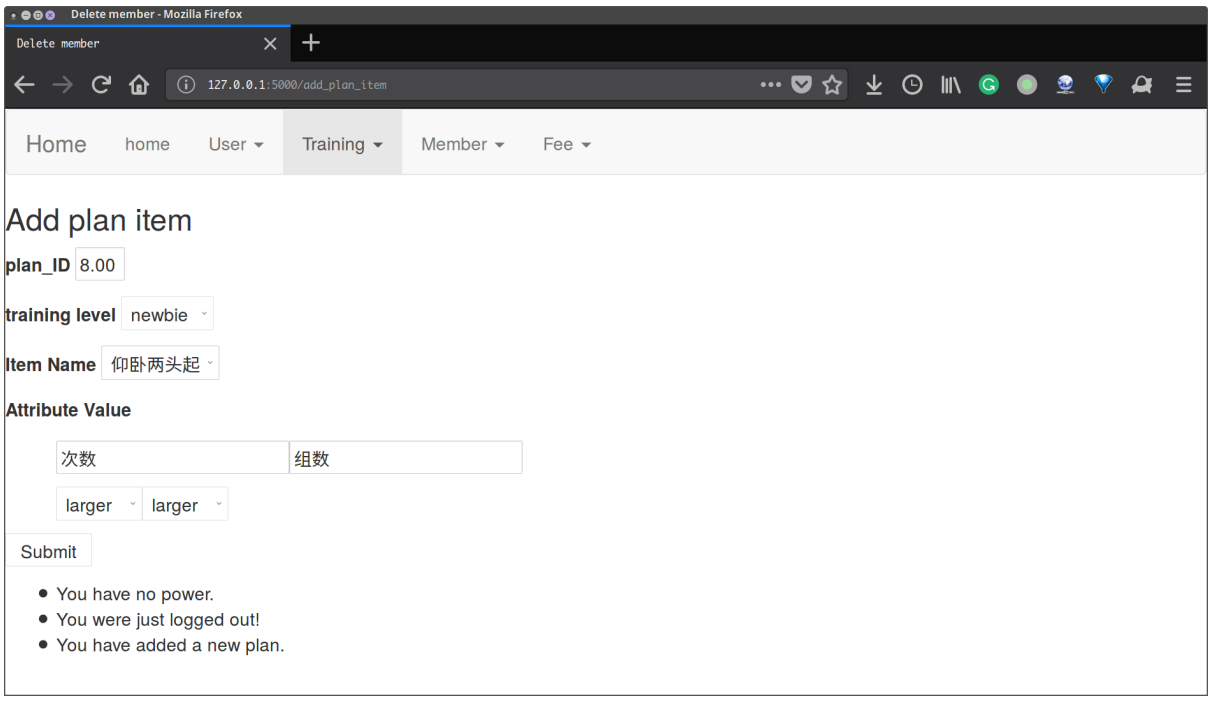


图 1.5-9 计划具体

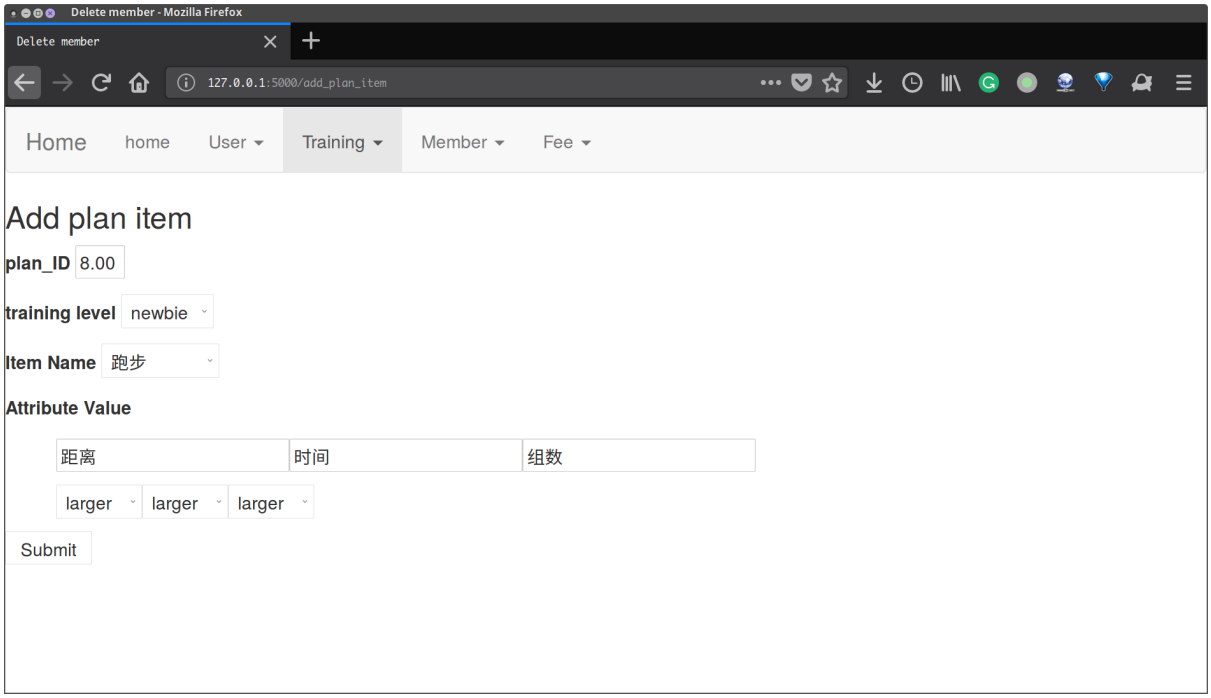


图 1.5-10 添加之后

添加计划主体

```
@training_blueprint.route('/add_plan_item', methods=['GET', 'POST'])
@login_required
@power_required
def add_plan_item():
    form = AddPlanItemForm()

    if form.validate_on_submit():
        plan_id = form.plan_ID.data
        new_req = RequirementInPlan()
        newItem = TrainingItem.query.filter_by(item_name=form.item_name.data).first()
        item_id = newItem.item_ID
        newAttr = [ID.attr_ID for ID in newItem.attr]

        for i in range(len(newAttr)):
            new_req = RequirementInPlan(plan_ID=plan_id,
                                        item_ID=item_id,
                                        attr_ID=newAttr[i],
                                        comp=form.comp.data[i],
                                        requirement=form.attr_name.data[i])
            db.session.add(new_req)

        db.session.commit()

        flash("You have added a new item to the plan")

    return render_template('add_plan_item.html', form=form)
```

和 AJAX 交互的，以及连接数据库的 python 代码，受限于页面大小，只能换行，代码并不能

正常运行

```
@training_blueprint.route('/_get_attr/')
@login_required
@power_required
def _get_attr():
    item_name = request.args.get('item_name', '01', type=str)

    print("from get attr:", item_name)
    #print(item_name)
    #should reorder or rewrite my js
    attr = [(row.attr_ID, row.attr_name) for row in
            TrainingItem.query.filter_by(item_name=item_name).first().attr]
    #print(attr)

    return jsonify(attr)
```

异步的 js 代码片段

```
<script>

 var dropdown = {
     item: $('#item_name'),
     attr: $('#attr_name'),
     comp: $('#comp')
 };

 updateattr();

 function updateattr() {
     var send = {
         item_name: dropdown.item.val()
     };

     console.log(send.item_name);

     //send = {item: "卧拉"}; // default
     dropdown.attr.attr('disabled', 'disabled');
     dropdown.attr.empty();
     dropdown.comp.attr('disabled', 'disabled');
     dropdown.comp.empty();

     var sel_row = "<select></select>"

     $.getJSON("_get_attr", send, function(result) {
         $.each(result, function(i, field) {
           console.log("<select \"id=comp-" + field[0] +
           "\" name=\"comp-" + field[0] +
           "\"><option value=\"larger\">larger<\/option><option value=\"smaller\">smaller<\/option>" + "</selec
             dropdown.comp.append(
             "<select \"id=comp-" + field[0] +
             "\" name=\"comp-" + field[0] +
```

```
            "\"><option value=\"larger\">larger<\/option><option value=\"smaller\">smaller<\/option>" + "</sel
            );
            dropdown.attr.append(
                $('<input>', {
                    name: 'attr_name-' + field[0],
                    value: field[1],
                })
            );

            console.log(i + " " + field);
        });
        dropdown.attr.removeAttr('disabled');
    })
};

dropdown.item.on('change', function() {
    updateattr();
});


</script>
```

§1.6 课程设计心得体会

本项目整体用时大概在 2 周，其中用了一两天设计修改数据库。其他几天大部分时间都在网上查找阅读文档，边读边写出项目的代码。其中遇到了好几个瓶颈区，一个是数据库的设计一开始没有考虑好（抽象出），训练计划，训练项目，训练指标的这三者的关系。一开始纯粹把训练指标当做训练计划的一个属性，但实际上这样做，数据库的表达能力是很弱的。这种情况下数据库对于新增加的指标并不能用 DDL 表达，只能在数据库里面操作更改表，即数据词典，并增加一列没有意义的 null。对于修改数据词典才能支持新的训练指标的关系模式实在是很难实现插入（要增加额外的判断，那个项目应该对应哪些指标）。因此我重新设计了关系模式，把训练指标作为一个实体集来对待。这个设计相比之前的设计就具有了更好的表达能力，在插入的操作上也更好写。只需要在（项目<->指标）的多对多表中查找相应的指标，并返回给用户。在实现中用的是 AJAX 的异步编程，由于 js 不大熟悉，因此也是用了挺久的时间的。

然后其次还有要考虑密码的存储，不能明文存储，所以要用单向的 hash，让步存入数据库。出于安全性考虑，数据库只对 localhost 开放。以后如果要开发小程序可能需要开放端口，我觉得这样对于安全性而言其实是很大的挑战。因此要把这个项目真正用到实际，还需要做的工作还有很多。这次的项目让我了解到了如果想要开发一个动态网页的应用，不仅仅是理论的知识（数据库）要足够，还要有编程的能力，还要懂得如何运维，如何确保用户数据的安全性，我觉得这有点涉及到软件工程的领域。总而言之，就是开发一个看似功能很普通的应用背后，都是一个又一个等待解决的问题，而开发一个项目确实给我一种从无到有，并能从中学习到很多相关领域的知识。

最后，感谢辛苦半年的助教和老师。