

目录

1 项目名称	2
2 摘要	2
3 软件分类	2
4 应用领域	2
5 开源协议	2
6 作品概述	2
6.1 背景及应用领域	2
6.2 作品特点	3
6.3 设计思路	3
6.4 功能描述	4
7 体系结构	5
8 关键技术点	5
8.1 获取时间	5
8.2 API 的接收与请求	7
8.3 根据时间跳转	8
8.4 数据库	12
9 功能模块设计	21
9.1 首页	22
9.2 课程详细页面	22
9.3 饮食	23
9.4 天气	24
9.5 训练计划	25
9.6 计划详细内容	25
9.7 动作详情	26
9.8 运动历史	26
10 过程中遇到的问题	27
10.1 天气获取	27
10.2 数据库存储	27
10.3 真机与模拟器	28
11 工作分配	28
12 比赛收获	29

1 项目名称

Run-Gymnastic

2 摘要

近年来，随着公众对健康生活方式的向往和对运动科技的高度关注，智能手表作为市场上备受追捧的产品，已经成为连接健康与科技的重要桥梁。本项目正是基于此背景，针对跑步爱好者的交互体验，推出了创新的 Run-Gymnastic 应用。通过智能手表这一便捷平台，Run-Gymnastic 不仅提供实时天气信息，还为用户量身定制训练方案，并辅以智能化的饮食与装备建议，旨在全方位满足跑者多方面的需求，实现“活动有方，五脏自和”的理想生活状态。

3 软件分类

生活类

4 应用领域

健身与健康

5 开源协议

"license": "Apache-2.0"

6 作品概述

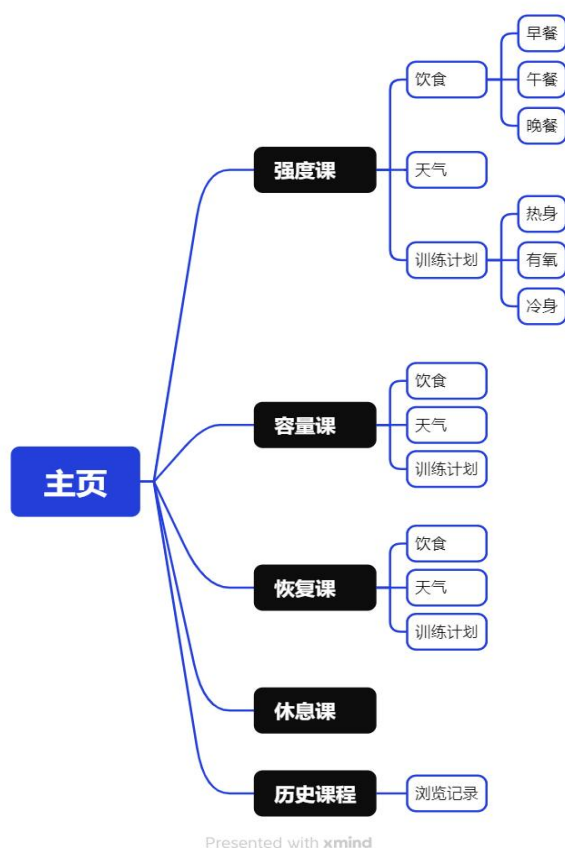
6.1 背景及应用领域

一款专为跑步爱好者精心打造的创新型跑步训练计划手表应用，集成了丰富的功能以满足跑者的多元化需求。这款应用不仅提供定制化的训练课程和计划，还实时更新天气信息，智能推荐适合的训练饮食和装备，通过这一系列的交互体验，为跑者带来前所未有的便利和个性化服务，助力他们实现更高效的跑步训练和更健康的生活方式。

6.2 作品特点

本项目开发的智能手表应用，以其创新的交互设计为核心，为用户提供了一种直观且愉悦的操作体验。应用界面友好，简化了用户操作流程，使得即使是初次接触智能手表的用户也能轻松上手。应用具备强大的实时数据处理能力，能够为用户提供即时反馈，例如实时的天气更新和动态调整的训练计划，确保用户始终获得最新的信息和个性化服务。这些特点共同构成了一个易于使用、互动性强、响应迅速的应用，极大地提升了用户的满意度。

6.3 设计思路



该项目专注于为用户提供一个全面而便捷的运动课程导航平台。首先，用户会看到一个简洁明了的主页，这里列出了可选的课程类别，如“强度课”、“容量课”等。一旦用户选择了某一课程，系统会迅速引导他们进入该课程的详细页面。在课程的详细页面上，用户可以查看到与课程紧密相关的天气信息，这对于户外运动来说至关重要。

此外，系统还会为用户提供专业的饮食建议，详细说明了在训练前后应该摄入的食物种类和营养配比，以确保用户能够获取足够的能量并促进身体的恢复。最核心的信息是详细的训练计划。这些计划结合了有氧和无氧运动，旨在帮助用户全面提升体能。用户可以根据这些计划进行训练，达到自己设定的健身目标。

6.4 功能描述

6.4.1 首页

实时时间显示
课程选项展示与管理
课程详情快速跳转
运动历史查看

6.4.2 课程详细页面

日期、饮食、天气快速查看
激励名言展示
比赛倒计时提醒
训练计划查看

6.4.3 饮食界面

日期显示与饮食时间建议
详细饮食建议与计划查看
快速返回课程页面

6.4.4 天气页面

实时天气信息与装备建议

6.4.5 训练计划页面

多样化训练项目与具体运动时间

6.4.6 计划详细内容页面

返回按钮与具体训练内容展示
查看后续训练内容

6.4.7 动作详细页面

提供动作要领
点击完成课程计入数据库

7 体系结构



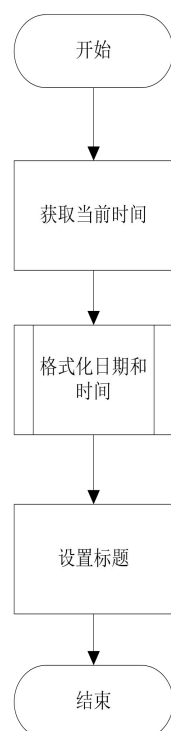
8 关键技术点

8.1 获取时间

该功能用于获取当前系统的时间，确保时间信息的准确性和即时性。通过库函数进行操作显示实时时间，如首页的表盘或任何需要显示当前时间的页面。同时，用于计算时间差等与时间相关的操作。

(1) 获取系统时间

1.流程图如下：



2.代码如下：

```
// 获取当前系统时间
const currentDate = new Date();
// 格式化日期和时间，例如：'23Feb 14:54'
const formattedDate = this.formatDate(currentDate);
// 设置 title 为格式化后的当前时间
this.date = formattedDate;

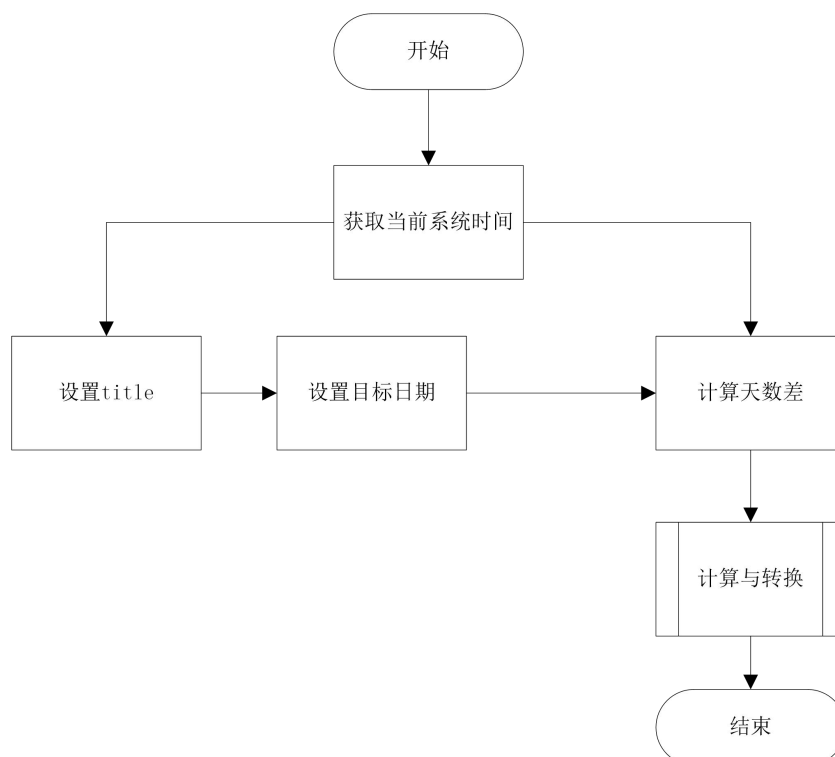
formatDate(date) {
  // 格式化日期和时间的函数，根据您的需要调整格式
  const monthNames = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'];
  const day = date.getDate();
  const monthIndex = date.getMonth();
  //const year = date.getFullYear();
  const hours = date.getHours();
  const minutes = date.getMinutes();

  // 确保分钟是两位数
  const formattedMinutes = minutes < 10 ? `0${minutes}` : minutes;

  // 返回格式化后的日期和时间字符串
  return `${day} ${monthNames[monthIndex]} ${hours}:${formattedMinutes}`;
},
```

(2) 计算当前日期与现在日期的时间差

1.流程图如下：



2.代码如下：

```
// 设置目标日期为2024年10月19日
const targetDate = new Date(2024, 10, 19); // 注意月份是从0开始的，所以10月是9

// 计算两个日期之间的天数差
this.days = this.daysBetweenDates(currentDate, targetDate);
```

```
daysBetweenDates(date1, date2) {
  // 获取时间差的毫秒数
  const diff = Math.abs(date2 - date1);

  // 转换成天数
  const days = Math.ceil(diff / (1000 * 60 * 60 * 24));

  return days;
},
```

8.2 API 的接收与请求

集成专业的天气 API 接口，以确保能够即时获取最新的天气信息。

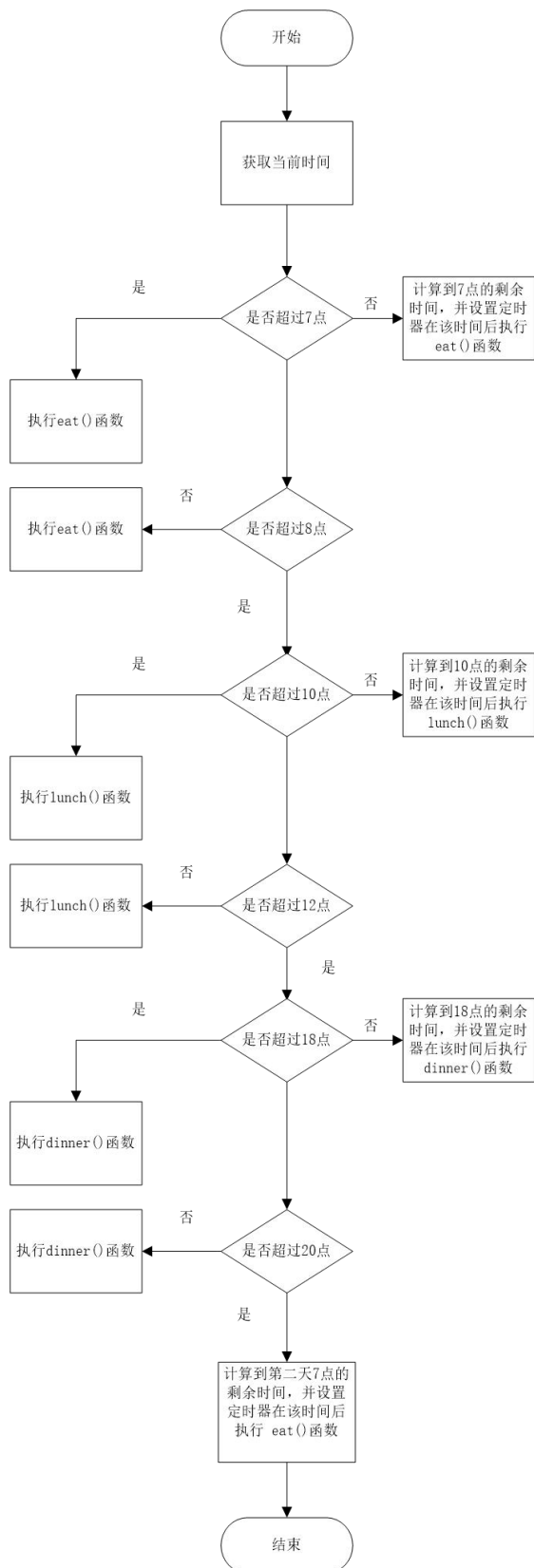
```
import fetch from '@blueos.network.fetch';
export default {
  data() {
    return {
      tem: '',
      wind: '',
      wetness: '',
      pro: '',
      cit: ''
    }
  }
}
```

```
const winddirection = responseData.data.lives[0].winddirection;
const humidity = responseData.data.lives[0].humidity;
const province = responseData.data.lives[0].province;
const city = responseData.data.lives[0].city;
const weather = responseData.data.lives[0].weather;
this.tem = `${temperature}°C`;
this.wind = `${winddirection}风`;
this.wetness = `${humidity}%`;
this.pro = `${province}`;
this.cit = `${city}`;
this.we = `${weather}`;
console.log(responseData);
},
fail: (error) => {
  console.error("Fetch error: ", error);
}
})
},
```

8.3 根据时间跳转

根据实际时间与设定的饮食建议时间做对比，实现直接跳转到饮食页面。

1.流程图如下：



2.代码如下:

```

    this.jump_eat();
},|
jump_eat(){
    const currentDate = new Date();//获取当前时间
    const hou = currentDate.getHours();//取此时的小时
    const miu = currentDate.getMinutes();//取此时的分钟
    /*以下两个变量的作用为，例如计算17:45距离18:00还有多少分钟
    记录距离下一个整点差多少分钟：tm=60-45=15 分钟
    记录时间间隔：tim= (18-1-17) + tm=15 分钟
    */
    // 测试：
    //const hou = 7;
    //const miu = 0;
    let tm = 0;
    let tim = 0;

    if(hou < 7)
    {
        if(miu > 0)
        {
            tm = 60 - miu;

```

```

            tim = ((7 - 1 - hou) * 60 + tm) * 60 * 1000;
        }
    }
    else{
        tim = ((7 - hou) * 60) * 60 * 1000;
    }
    setTimeout(this.eat , tim);
}
else if(hou >= 7 && hou <= 8)
{
    if(hou == 7||(hou == 8&&miu == 0))
        this.eat();
    else{
        tm = 60 - miu;
        tim = ((10 - 1 - hou) * 60 + tm + 30) * 60 * 1000;
        setTimeout(this.lunch , tim);
    }
}
else if(hou > 8 && hou < 10)
{
    if(miu > 0)

```

```

    {
        tm = 60 - miu;
        tim = ((10 - 1 - hou) * 60 + tm + 30) * 60 * 1000;
    }
    else{
        tim = ((10 - hou) * 60 + 30) * 60 * 1000;
    }
    setTimeout(this.lunch , tim);
}
else if(hou >= 10 && hou <= 12)
{
    if(hou == 10 && miu < 30)
    {
        tim = (30 - miu) * 60 * 1000;
        setTimeout(this.lunch , tim);
    }
    else if(hou == 12 && miu != 0)

```

```

    tm = 60 - miu;
    tim = ((18 - 1 - hou) * 60 + tm + 30) * 60 * 1000;
    setTimeout(this.dinner, tim);
}
else {
    this.lunch();
}
}
else if (hou > 12 && hou < 18) {
    if (miu > 0) {
        tm = 60 - miu;
        tim = ((18 - 1 - hou) * 60 + tm + 30) * 60 * 1000;
    }
    else {
        tim = ((18 - hou) * 60 + 30) * 60 * 1000;
    }
    setTimeout(this.dinner, tim);
}
}

```

```

}
else if(hou >= 18 && hou <= 20)
{
    if(hou == 18 && miu < 30)
    {
        tim = (30 - miu) * 60 * 1000;
        setTimeout(this.dinner , tim);
    }
    else if(hou == 20 && miu != 0)
    {
        tm = 60 - miu;
        tim = ((24 - 1 - hou) * 60 + tm + 7 * 60) * 60 * 1000;
        setTimeout(this.eat , tim);
    }
    else{
        this.dinner();
    }
}
else{

```

```

    if(miu > 0)
    {
        tm = 60 - miu;
        tim = ((24 - 1 - hou) * 60 + tm + 7 * 60) * 60 * 1000;
    }
    else{
        tim = ((24 - hou) * 60 + 7 * 60) * 60 * 1000;
    }
    setTimeout(this.eat , tim);
}

```

8.4 数据库

一、数据库的后端代码主要由四个部分构成

1. TrainController

● 概述：

TrainController 类是一个 Spring Boot RESTful 控制器，提供了对 Nexuscore 对象的管理功能。通过 HTTP 请求，客户端可以获取所有记录或添加新记录。

```

8  import com.fasterxml.jackson.databind.ObjectMapper;
9  import org.springframework.beans.factory.annotation.Autowired;
10 import org.springframework.web.bind.annotation.*;
11
12 import java.util.Date;
13 import java.util.Map;
14
15 @RestController
16 @RequestMapping("/train")
17 @Slf4j
18 public class TrainController {
19
20     @Autowired
21     private TrainMapper trainMapper;
22
23     @GetMapping("/getAll")
24     public Result getAll() {
25
26         return Result.success(trainMapper.getAll());
27     }
28
29     @PostMapping("/add")
30     public Result add(@RequestBody Nexuscore nexuscore) {
31
32         log.info("{} ", nexuscore);
33
34         trainMapper.insert(nexuscore);
35
36         return Result.success();
37     }
38 }
39

```

● 类定义和注解

```

@RestController
@RequestMapping("/train")
@Slf4j

```

- ① `@RestController`: 声明这是一个 RESTful 控制器。
- ② `@RequestMapping("/train")`: 基础 URL 路径。
- ③ `@Slf4j`: 生成日志记录器。

● 获取所有记录的 API

```

@GetMapping("/getAll")
public Result getAll() {

    return Result.success(trainMapper.getAll());
}

```

- ① `@GetMapping("/getAll")`: 映射 HTTP GET 请求，路径为 `/train/getAll`。
- ② 调用 `trainMapper.getAll()` 方法，获取所有 Nexuscore 记录。

- 添加记录的 API

```
@PostMapping("/add")
public Result add(@RequestBody Nexuscore nexuscore) {

    log.info("{} ", nexuscore);

    trainMapper.insert(nexuscore);

    return Result.success();
}
```

- ① @PostMapping("/add"): 映射 HTTP POST 请求，路径为/train/add。
- ② 使用@RequestBody 将请求体中的 JSON 数据转换为 Nexuscore 对象。
- ③ 记录传入的 Nexuscore 对象信息。
- ④ 调用 trainMapper.insert(nexuscore)方法，将 Nexuscore 对象插入数据库。
- ⑤ 返回成功结果。

2. TrainMapper

- 概述:

TrainMapper 接口是一个 MyBatis 的 Mapper 接口，定义了用于操作 Nexuscore 表的 SQL 语句。通过这个接口，可以获取所有 Nexuscore 记录或插入新的记录。

```
1 package com.ncut.mapper;
2
3 > import ...
4
5 @Mapper 2 usages
6 public interface TrainMapper {
7
8     @Select("select * from nexuscore") 1 usage
9     List<Nexuscore> getAll();
10
11     @Insert("insert into nexuscore (date, train_item, duration) values (#{date}, #{trainItem}, #{duration}) ") 1 usage
12     void insert(Nexuscore nexuscore);
13 }
14
15
```

- 获取所有记录的方法

```
@Select("select * from nexuscore") 1 usage
List<Nexuscore> getAll();
```

- ① @Select("select * from nexuscore"): 指定 SQL 查询语句，用于获取 nexuscore 表中的所有记录。
- ② 方法 getAll(): 返回包含所有 Nexuscore 对象的列表。

- 插入新记录的方法

```
@Insert("insert into nexuscore (date, train_item, duration) values (#{date}, #{trainItem}, #{duration}) ") 1 usage
void insert(Nexuscore nexuscore);
```

- ① @Insert("insert into nexuscore (date, train_item, duration) values (#{date},

`#{trainItem}, #{duration})"`): 指定 SQL 插入语句, 将 Nexuscore 对象的属性值插入到 nexuscore 表中。

② 方法 `insert(Nexuscore nexuscore)`: 接受一个 Nexuscore 对象作为参数, 并将其插入到数据库中。

3. Nexuscore

● 概述

Nexuscore 类是一个数据对象类, 用于表示训练记录。它包含训练的日期、项目和持续时间等信息, 并提供了序列化和反序列化的支持。

```
package com.ncut.pojo;

import ...

@Data 5 usages
@NoArgsConstructor
@AllArgsConstructor
public class Nexuscore {

    private static final long serialVersionUID = 1L;

    private Long id;

    @JsonFormat(pattern="yyyy-MM-dd",timezone="GMT+8")
    private Date date;

    private String trainItem;

    private Integer duration;
}
```

● 类定义和注解

```
@Data 5 usages
@NoArgsConstructor
@AllArgsConstructor
public class Nexuscore {
```

① `@Data`: 自动生成 `getter`、`setter`、`toString`、`equals`、`hashCode` 等方法。

② `@NoArgsConstructor`: 自动生成无参构造函数。

③ `@AllArgsConstructor`: 自动生成包含所有字段的构造函数。

● 类字段定义

```
private static final long serialVersionUID = 1L;
```

定义一个静态常量 `serialVersionUID`, 用于序列化。

```
private Long id;
```

定义 `id` 字段, 类型为 `Long`, 用于唯一标识 Nexuscore 对象。

```
@JsonFormat(pattern="yyyy-MM-dd", timezone="GMT+8")

private Date date;
```

- ① `@JsonFormat(pattern="yyyy-MM-dd", timezone="GMT+8")`: 指定日期格式化模式和时区。
- ② `date` 字段，类型为 `Date`，表示训练日期。

```
private String trainItem;
```

定义 `trainItem` 字段，类型为 `String`，表示训练项目的名称。

```
private Integer duration;
```

定义 `duration` 字段，类型为 `Integer`，表示训练持续时间，以分钟为单位。

4. RunPulseApplication

- 概述

`RunPulseApplication` 类是 Spring Boot 应用程序的主类，用于启动应用程序。该类包含应用程序的入口点，并配置了必要的 Spring Boot 注解。

```
package com.ncut;

import lombok.extern.slf4j.Slf4j;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
@Slf4j
public class RunPulseApplication {

    public static void main(String[] args) {
        SpringApplication.run(RunPulseApplication.class, args);
    }
}
```

- 类定义和注解

```
@SpringBootApplication
@Slf4j
public class RunPulseApplication {
```

- ① `@SpringBootApplication`: 标记这是一个 Spring Boot 应用程序的主类，包含配置、自动配置和组件扫描的功能。
- ② `@Slf4j`: 使用 Lombok 注解生成一个名为 `log` 的日志记录器。

- 主函数


```
public static void main(String[] args) {  
    SpringApplication.run(RunPulseApplication.class, args);  
}
```

- ① `public static void main(String[] args)`: Java 应用程序的入口方法。
- ② `SpringApplication.run(RunPulseApplication.class, args)`: 启动 Spring Boot 应用程序，初始化 Spring 上下文和内嵌服务器。

二、在前端中，通过编写的 JS 函数将训练数据写入数据库，以及将数据库中的历史训练数据的读出

1. 训练数据的写入：



● 概述：

以弓步压腿的动作详情页面为例，上图为手表页面，将写数据的 JS 函数与已完成按钮绑定，点击此按钮，调用函数，实现将该动作的完成时间、名称和训练时间写入数据库中。

```
<input class="btn1" type="button" value="已完成" @click="transform"/>
```

```

transform() {
  // 获取当前系统时间
  const currentDate = new Date();
  // 设置title为格式化后的当前时间
  this.year = currentDate.getFullYear();
  this.month = currentDate.getMonth()+1;
  this.day = currentDate.getDate();

  const data1 = {
    date: `${this.year}-${this.month.toString().padStart(2, '0')}-${this.day.toString().padStart(2, '0')}`,
    trainItem: `${this.title}`,
    duration: `${this.time.replace('s', '')}`
  }

  fetch.fetch({
    url: 'http://localhost:8080/train/add',
    method: 'POST',
    header: {
      'Content-Type': 'application/json;charset=UTF-8',
      // 'Data-Type': 'json'
    },
    data: JSON.stringify(data1),
    success: function(data) {
      console.log('Data sent successfully:', data);
    },
    fail: function(data, code) {
      console.error('Failed to send data:', data, 'Error code:', code);
    }
  })
}

```

- 绑定:

```
<input class="btn1" type="button" value="已完成" @click="transform"/>
```

将 JS 中定义的 transform 函数，绑定到按钮上。

- 定义函数名:

```
transform() {
```

- 获取当前时间:

```

// 获取当前系统时间
const currentDate = new Date();
// 设置title为格式化后的当前时间
this.year = currentDate.getFullYear();
this.month = currentDate.getMonth()+1;
this.day = currentDate.getDate();

```

- ① 通过 Date 函数获取当前的系统时间存放在 currentDate 中。
- ② 对 currentDate 调用 Date 类的方法,获取年月日存放在 data 中定义好的 year、month 和 day 中

- 将获取的时间信息转化为 JS 对象

```

const data1 = {
  date: `${this.year}-${this.month.toString().padStart(2, '0')}-${this.day.toString().padStart(2, '0')}`,
  trainItem: `${this.title}`,
  duration: `${this.time.replace('s', '')}`
}

```

- ① 对月份和日，原本是数字，使用 `toString` 方法转化为字符串，再使用 `padStart` 方法规定字符串长度 2，若长度不够，则用 '0' 填充。
- ② 对存放持续时间的变量 `time` 使用 `replace` 方法，将末尾的 `s` 替换为 ' '。
- ③ 定义一个 `data1` 的 JS 对象，`date` 属性存放格式化好的年月日的字符串，`trainItem` 属性存放训练项目名称，`duration` 属性存放格式化好的持续时间。

● 发送 POST 请求

```
fetch.fetch({
  url: 'http://localhost:8080/train/add',
  method: 'POST',
  header: {
    'Content-Type': 'application/json;charset=UTF-8',
    // 'Data-Type': 'json'
  },
  data: JSON.stringify(data1),
  success: function(data) {
    console.log('Data sent successfully:', data);
  },
  fail: function(data, code) {
    console.error('Failed to send data:', data, 'Error code:', code);
  }
})
```

- ① 使用 `fetch.fetch` 方法发送一个 POST 请求到 <http://localhost:8080/train/add>
- ② `url` 指定请求的服务器的地址。
- ③ `method` 指定使用 POST 方法发送数据。
- ④ `header` 指定请求的内容类型为 JSON 格式，并且字符编码为 UTF-8。
- ⑤ 使用 `JSON.stringify` 方法将 JS 对象 `data1` 转换成 JSON 字符串，便于在网络请求中发送。
- ⑥ `data` 指定在网络请求中发送的数据是转换成 JSON 字符串的 `data1`。
- ⑦ `success` 是一个回调函数，请求成功输出打印出传送的数据。
- ⑧ `fail` 是一个回调函数，请求失败打印出错误信息。

2. 历史训练数据的读出



● 概述：

上图为手表页面，将读数据的 JS 函数放到该页面的 OnShow 函数中，打开该页面，执行 OnShow 函数，便能将从数据库中读到的运动的历史数据显示在页面中。

```
<list-item type="item" for="{ { newList } }">
  <text class="text_date">{{ $item.date.substring(0,10) }}</text>
  <text class="text_item">{{ $item.trainItem }}</text>
  <text class="text_seconds">{{ $item.duration + 's' }}</text>
</list-item>
```

```
onShow() {
  fetch.fetch({
    url: 'http://localhost:8080/train/getAll',
    method: 'get',
    responseType: "json",
    success: (response) => {
      let responseData;
      if (typeof response === 'string') {
        responseData = JSON.parse(response);
      } else {
        responseData = response;
      }
      console.log(response)
      this.newList = responseData.data.data;
    },
    fail: (error) => {
      console.error("Fetch error: ", error);
    }
  })
},
```

● 页面显示:

```
<list-item type="item" for="{ { newList } }">
  <text class="text_date">{{ $item.date.substring(0,10) }}</text>
  <text class="text_item">{{ $item.trainItem }}</text>
  <text class="text_seconds">{{ $item.duration + 's' }}</text>
</list-item>
```

① 使用 for 循环，遍历 newList，每一个遍历到的元素都放到 item 中，然后通过 item 的属性来展示数据。

② 从数据库中读出的日期格式与页面期望显示的不同，于是对 item 的 date 使用 substring 方法，对获取到的日期进行切片，使其在页面中显示的是年月日

③ 从数据库中读出的 duration 它表示的是时间，但是读出来只有数字，没有对应的单位，于是显示的时候在后面加上单位。

● 发送 GET 请求


```

onShow() {
  fetch.fetch({
    url: 'http://localhost:8080/train/getAll',
    method: 'get',
    responseType: "json",
    success: (response) => {
      let responseData;
      if (typeof response === 'string') {
        responseData = JSON.parse(response);
      } else {
        responseData = response;
      }
      console.log(response)
      this.newList = responseData.data.data;
    },
    fail: (error) => {
      console.error("Fetch error: ", error);
    }
  })
},

```

- ④ 使用 `fetch.fetch` 方法发送一个 GET 请求到 <http://localhost:8080/train/getAll>
- ⑤ `url` 指定请求的服务器的地址。
- ⑥ `method` 指定使用 GET 方法请求数据。
- ⑦ `responseType` 指定期望的响应数据类型为 JSON。
- ⑧ `success` 是一个回调函数，在请求成功时执行。
- ⑨ 首先检查响应是否为字符串类型（某些情况下，响应可能会以字符串形式返回）。
- ⑩ 如果是字符串类型，则使用 `JSON.parse(response)` 将其解析为 JSON 对象；否则，直接使用 `response`。
- ⑪ 打印 `response` 数据到控制台以便调试。
- ⑫ 解析后的数据中 `response` 的数据结构是 `{... data: { ...data: [...] }... }`，于是将 `response.data.data` 赋值给 `newList`。
- ⑬ `fail` 是一个回调函数，在请求失败时执行。
- ⑭ 打印错误信息到控制台，便于调试和错误处理。

9 功能模块设计

9.1 首页

在首页的表盘上，最显著的位置实时显示系统时间，通过精确的函数调用确保时间的即时性。页面会展示一系列精心设计的课程选项，包括“强度课”、“容量课”、“恢复课”和“休息课”。用户可以通过简单的左滑动作快速删除不再需要的课程，实现个性化的课程管理。一旦选定课程，系统会立即响应，引导用户进入该课程的详细页面，从而为用户提供全面而深入的课程信息。



9.2 课程详细页面

进入某一具体课程页面后，如果此时恰好是饮食时间，则会直接跳转到饮食页面，该功能旨在提醒用户及时补充营养、摄入食物。



若不是，以强度课为例，页面顶部显示实时日期，左上角的饮食标志提供专业饮

食建议，右上角的天气标志展示最新天气信息。中央的名言激励用户坚持运动，下方实时显示距离比赛的天数，提醒用户调整训练计划。底部的训练计划区域提供定制化锻炼方案。



9.3 饮食

以早餐为例，进入饮食界面后，首先是页面顶部的实时日期，为用户提供准确的时间参考。紧接着，系统会给出建议的饮食时间，确保用户能够按照科学的时间表进行用餐。

页面的主体部分则详细列出了针对当前情况的具体饮食建议，旨在为用户提供均衡、营养的饮食方案。

在页面的左侧和右侧，设有便捷的按钮，用户可以轻松点击以查看不同时间段的饮食计划，实现个性化的饮食管理。而在页面的最底部，设有一个返回首页的按钮，用户只需简单点击，即可迅速返回到课程详细页面。



9.4 天气

在天气页面的上方，我们集成专业的天气 API 接口，以确保用户能够即时获取最新的天气信息。通过这一连接，页面将实时展示包括当前所在地区及其天气状况、温度、湿度以及风向等在内的关键数据，提供一个全面而准确的天气概览。

在页面的下方，基于对实时天气数据的深入分析，我们为用户提供了精心定制的装备建议。这些建议充分考虑了当前的天气情况，旨在帮助用户选择最适合的装备，以应对多变的天气状况。



9.5 训练计划

进入训练计划页面后，多样化的训练项目将逐一呈现。这些计划涵盖了热身、有氧训练、无氧运动、力量训练以及冷身等多个关键阶段。每个阶段都配备了具体的运动时间，确保您在合适的时长内完成高效而系统的训练。



9.6 计划详细内容

以热身为例，在该页面的左上角，设有一个返回按钮，用户只需轻触即可迅速返回到计划页面，方便随时调整或查看其他训练计划。页面主体部分则详细列出了当前的具体训练内容，确保用户能够清晰了解每一项训练。而在页面的右侧，特别设置了一个按钮，点击后用户可轻松查看接下来的训练内容，为接下来的锻炼做好充分准备。



9.7 动作详情

以弓箭步动作为例，在动作详情页面的顶端醒目地展示着动作的名称；紧接着，页面中部会向用户展示建议的动作持续时间；文字主体部分则详尽地阐述了动作的关键要领；页面底部设有一个“已完成”按钮，点击即可将本次训练课程成功记录到数据库中，方便日后查阅和追踪训练进度。



9.8 运动历史

在运动历史页面可以看到，训练计划中完成后点击已完成按钮的所有运动历史，其内容包括该动作的完成时间，名称，以及每个动作需要坚持的时间。





10 过程中遇到的问题

10.1 天气获取

构建天气应用的过程中，我们期望从开发文档中获取到现成的天气接口，以便能够轻松地集成实时天气数据。然而，开发文档未能提供所需的天气接口。

解决办法：

为集成天气信息，我们选用了高德地图天气 API，它提供全面且易于集成的数据服务。首先，在高德地图官网注册并获取 API 密钥。获得密钥后，我们通过编程发送 HTTP 请求到 API，附带必要的请求参数，以获取 JSON 或 XML 格式的天气数据。

10.2 数据库存储

在开发过程中，我们最初可能使用沙箱来存储数据，以快速验证应用的功能。然而，随着开发的深入和数据的增多，使用沙箱来存储数据变得不再合适。

解决办法：

为保障数据持久性和可用性，我们将数据存储于稳定数据库。首先，在开发环境中设置数据库服务器，创建数据库及所需表结构，依据应用需求设计表结构。随后，编写代码连接数据库，实现数据增删改查操作，利用数据库连接库简化操作。最终，通过访问权限控制、数据加密、定期备份等措施，确保数据库安全，防止数据泄露或未授权访问。

10.3 真机与模拟器

模拟器连接数据库内容在真机上不知道是否能成功，若需要两边实现一致的效果，则需要大赛方支持。

11 工作分配

通过 git 和即使设计协同开发，分工如下：

张诗语：

主要负责开展用户调研的工作，旨在深入了解目标用户群体的需求与偏好，通过收集和分析用户数据，形成对产品设计的依据；基于调研结果，将定义 app 的交互逻辑，确保用户在使用过程中能够顺畅、自然地完成任务；还将负责制定用户操作流程，简化操作步骤，以提升整体的用户体验；界面 UI 设计，通过运用专业的设计理念和工具，使用户在使用过程中获得愉悦的视觉体验。

李俊杰：

- 1.将设计的 UI 页面转化为实际可运行的代码。熟悉前端技术栈以及开发文档，如 HTML、CSS 和 JavaScript，以确保页面正常显示和交互。
- 2.实现页面内容的同时，通过 js 语言的编写来实现获取系统时间的功能。这个功能将帮助用户实时了解当前的时间信息，提升应用的实用性。使用 Git 进行版本控制，确保代码的可靠性和可维护性。

谢邹健：

主要任务是连接天气 API 和编写数据库。

- 1.在网上搜索并筛选出合适的天气 API，了解 API 的接口文档和调用方式，以便在应用中获取实时的天气信息。在连接 API 的过程中，处理可能出现的网络错误和数据异常，确保数据的准确性和稳定性。
- 2.进行数据库的编写和连接工作，为应用提供数据存储和查询的功能。这包括设计数据库结构、编写 SQL 语句以及实现数据的增删改查等操作。使用 Git 进行版本控制，确保代码的稳定性和可维护性。

12 比赛收获

李俊杰：

在参加这个项目之前，我已经对 `ml`、`css` 以及 `js` 等语言有一些基础知识。但是，项目的要求我们进行创新型开发交互，这让我觉得我可以参与进来进一步熟悉这些知识。

在项目开始后的前几周，我们花了很多时间来理解项目的需求和，并构思项目应该从哪些方面去进行创新交互设计。

随着项目的推进，我们遇到了一些挑战，但在团队的共同努力下，我们克服了这些问题，最终成功地完成了项目。

通过这个项目让我学到了很多，我深入了解到蓝河操作系统手表应用开发，进一步熟练使用 `ml`、`css` 以及 `js` 等语言，同时通过及时沟通和查阅文档学习到一些之前没有接触过的内容。我很高兴能够参加这样的项目，我期待未来能继续在应用开发方面不断进步。

谢邹健：

在参加这个项目之前，我已经掌握了一些前端 `api` 调用、后端 `SpringBoot` 框架的一些基础知识以及简单的 `SQL` 语句。完成这个项让我可以更加熟悉这些知识，增强我的实战能力。

在项目开始后的前几周，我们花了很多时间来理解项目的需求并构思项目应该从哪些方面去进行创新交互设计，以及我负责的部分该如何实现

随着项目的推进，我由于不熟悉遇到了一些挑战，但在团队的共同努力下，以及我不断地学习钻研，最终配合其他两位同志，成功地完成了项目。

通过这个项目让我学到了很多，我深入了解到蓝河操作系统手表应用开发，进一步熟练了 `api` 调用、`SpringBoot` 框架和 `SQL` 的基础知识及语句，同时通过及时沟通、网络搜索资料、查阅文档学习到了许多新的知识。我认为参加这样的项目是非常有意义的，我期待未来能继续在应用开发方面不断进步。

张诗语：

在比赛中，我主要负责界面设计相关工作。此次比赛让我更好地实践之前学习的交互设计、页面设计相关的专业知识与理论，收获颇丰。

首先，在交互设计方面，我运用了以用户为中心的设计理念，关注用户在使用过程中的体验。我通过了解用户的需求和痛点，以简单、直观的操作方式来设计交互流程。同时，我运用了交互设计中的可用性原则，如一致性、反馈、简洁性等，确保用户在使用过程中能够快速上手，提高使用效率。

其次，在页面设计方面，我注重信息的布局与呈现。我运用了格式塔原理，通过对比、重复、对齐和亲近性等原则，使界面元素具有良好的视觉层次，提高用户的阅读体验。同时，我还关注色彩的运用，通过色彩搭配来突出重点，引导用户的注意力，增强界面的可读性。

此外，在团队协作方面，我与编程同学保持密切沟通，确保设计能够顺利实现。我学会了如何将设计稿精确还原为实际界面，并在过程中不断调整和优化，以达到最佳的用户体验。

最后，也十分荣幸可以参加本次比赛，本次比赛不仅提升了我的专业技能，也锻炼了我的解决问题和创新思维的能力，让我更加明白了在以后的学习和工作中怎样去更好地关注用户需求，运用专业知识与理论，为用户创造价值！