**目前前端架構**

目前所有的html都放在後端，以html顯示，整體架構已app做區別，到時候要修改就要去每個app裡面的view/temples裡面找html

**music:**

* discover:首頁，會顯示推薦歌曲等等，需要自行設計各種card
* serach\_reault：顯示查詢的結果
* music\_list顯示歌手所有歌的頁面
* my\_music\_list：顯示使用者不同專輯的歌曲，可以對自己的專輯做管理

**user:**

html在iRiver/user/templates/registration

* login:登戶畫面 ，主要方式透過 google line ，要如何做驗證跟套第三方登錄要跟後端溝通
* 目前user app架構比較鬆散，主要要設定嘅人使用者資訊
* 設定page: 調整像是要顯示亮色還是暗色，音質品質等等

**iRiver:**

還未登錄後顯示的首頁

* 首頁基本上就是介紹整個網站，像是網站地圖，開發人員等等

**CLASS：**

把所有常用到的功能寫成到lib資料夾裡面，最常見的事session跟 fetch，其他的像是音樂控制器可以重新寫一個

* Fetch：把5個http動詞拆分開來
* target 是url,params是參數
* 整體情況還要依據前端需求改變

const params = {

session\_key: this.session,

action: "check\_session\_key"

};

const response = await this.fetch.POST({ target: this.target, params });

* sessionController:管理session，通常session都只能字串才能儲存，sessionController 可以直接給他一個key值做儲存

const sessionController = new SessionController();

const isLogin = sessionController.get({ item: 'isLogin' })

**目標：**

* 前後端分離
* 需要加入白天模式跟夜間模式
* 那些檔案名稱可以再重新跟後端組討論變數名稱
* 因為前後端分離，會有token問題，需要在post之前跟後端獲取驗證，然後加入到header上，代表fetch class要改寫

**後端:**

django-cors-headers插件

，可以使用Django REST Framework（DRF）

**Search**

* 搜尋結果

| **方式** | **變數** |
| --- | --- |
| GET | query |

**discover**

* 給使用者呈現最新歌曲以及最熱門歌曲
* 輸出 html

**my\_music\_list**

* 顯示個人專輯
* 從資料庫查詢
* input: 需給改方把跟哪個專輯

**獲取資料**

* get\_navbar\_data : 顯示必要資料
* my\_music\_list：獲取個人資料
* music\_list：獲取那歌手的相關歌曲資料
* query\_db\_song：網路查詢
* query\_web\_song：資料庫查詢
* is\_song\_exist：去認歌曲存在

**dow**

* download\_song：把使用者想聽的那首歌下載
* download\_songs：把歌手所有歌下載

**Lib**

* dow : 下載歌曲跟照片
* sql：所有
* web\_scutter ：爬蟲
* log ：紀錄下載狀況，之後再自動化可以在篩選下載狀況

**需要改進：**

* 前後端分離
* 使用api風格與前端溝通

**程式規則**

**Class:**

1. 名稱第一個都要以英文開頭

**變數:**

1. 使用駝峰命名法：像是 **calculateTotalAmount()**。
2. 如果是常數就要用全部大寫，像是TIMEOUT

**function:**

**react**

目前規劃，可以在討論

**React + ts + TailwindCSS ＋next?**

* **Prettier +Eslint是插件 一個是排版跟檢查靜態錯誤**

**django:**

1. 使用 Django 的命名慣例：Django 採用了一些命名慣例，例如模型類名應使用單數形式，並使用大寫字母開頭，如 **class UserProfile(models.Model):**。
2. 按功能區分應用程式：將 Django 專案分為多個應用程式，每個應用程式應該專注於特定功能或業務邏輯。這有助於保持代碼組織結構和可擴展性。
3. 使用 Django 的視圖和模板：Django 提供了強大的視圖（views）和模板（templates）系統。建議使用這些功能來處理應用程式的邏輯和呈現頁面。
   1. 將邏輯運算移到模型（models）層：Django 的模型是用於處理資料邏輯的理想地點。將複雜的業務邏輯和資料操作放在模型層中，使視圖只負責處理與 HTTP 請求和回應相關的邏輯。
   2. views 裡面只會有回應http的東西
4. ORM：Django 的 ORM（Object-Relational Mapping，用於與數據庫進行交互。優先使用 ORM 提供的方法和技術，而不是直接編寫原始 SQL 語句，主要還是要看成員能力跟時間。