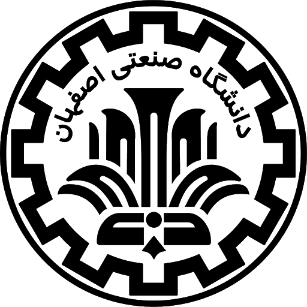
﷽



**دانشگاه صنعتی اصفهان**

دانشکده برق و کامپیوتر

سیستم نرم‌افزاری استفاده از داده های

لحظه ای هواشناسی در خانه هوشمند

**استاد راهنما**

دکتر مسعود رضا هاشمی

**دانشجو**

امین حلوائی اردکانی

**تاریخ ارائه**

بهمن 1402

**تشکر و قدردانی**

از استاد عزیز، جناب آقای دکتر هاشمی، بابت همراهی گرانقدر و صمیمانه ایشان در این مسیر کمال تشکر و قدردانی را به عمل می‌آورم.

با آرزوی پیشرفت همیشگی و توفیق در پروژه‌های آینده.

**چکیده**

در این پروژه به طراحی و توسعه سیستم نرم‌افزاری ای جهت بهره بردن از داده های آب و هوا در یک خانه هوشمند پرداختیم. در ابتدا یک طرح اولیه از نیازمندی های پروژه ارائه شد و در ادامه با توجه به این نیازمندی ها، طراحی روند ها، شبکه و پایگاه داده سیستم انجام شد. در مرحله پیاده سازی از ابزار های مختلف توسعه نرم‌افزار تحت وب از جمله Django-rest-framework و FastAPI استفاده شد. همچنین روندی ارائه خواهد که با استفاده از آن، آب و هوا به دمای بهینه قابل تنظیم در خانه تبدیل می‌شود.

**کلمات کلیدی:** Smart Home، Weather Data، Software Design،Web services ، Database

فهرست مطالب

[فصل اول مقدمه 1](#_Toc158482042)

[1-1- خانه هوشمند 2](#_Toc158482043)

[1-2- داده های هواشناسی 3](#_Toc158482044)

[1-2-1- ذخیره سازی داده‌ها 3](#_Toc158482045)

[1-2-2- خدمات API داده‌ها 3](#_Toc158482046)

[1-3- معماری سیستم 4](#_Toc158482047)

[فصل دوم مفاهیم و پیش‌‍‌زمینه‌ها‌ 5](#_Toc158482048)

[2-1- وب سرویس 5](#_Toc158482049)

[2-2- مقایسه REST و SOAP 6](#_Toc158482050)

[3-2- مفاهیم RESTful API 7](#_Toc158482051)

[2-3-1- منابع 8](#_Toc158482052)

[2-3-2- شناسه های یکنواخت منابع (URIs) 8](#_Toc158482053)

[2-3-3- عملیات های CRUD 9](#_Toc158482054)

[2-4- ابزار های مورد استفاده 10](#_Toc158482055)

[2-4-1- کنترل‌کننده خانه 10](#_Toc158482056)

[2-4-2- سرور 10](#_Toc158482057)

[فصل سوم روش انجام پروژه 12](#_Toc158482058)

[3-1- نیازمندی های کاربر 13](#_Toc158482059)

[3-2- طراحی اجزا و فرایند‌ها 14](#_Toc158482060)

[3-2-1- طراحی رابطه موجودیت های پایگاه داده 14](#_Toc158482061)

[3-2-2- طراحی روند فعالیت های کنترلر 17](#_Toc158482062)

[3-3- ساختار پروژه 25](#_Toc158482063)

[3-3-1- سرور جنگو (backend) 25](#_Toc158482064)

[3-3-2- کنترلر خانگی (controller) 27](#_Toc158482065)

[3-4- پیاده‌سازی 28](#_Toc158482066)

[3-4-1- اپلیکیشن accounts 28](#_Toc158482067)

[3-4-2- اپلیکیشن home 29](#_Toc158482068)

[3-4-3- اپلیکیشن weather 34](#_Toc158482069)

[3-4-4- اپلیکیشن cservice 36](#_Toc158482070)

[3-5- سرویس‌های ثانویه 37](#_Toc158482071)

[3-5-1- احراز هویت 37](#_Toc158482072)

[3-5-2- مستندات 37](#_Toc158482073)

[3-6- امنیت 37](#_Toc158482074)

[فصل چهارم نتیجه‌گیری 38](#_Toc158482075)

[پیوست‌ها 39](#_Toc158482076)

[مراجع 40](#_Toc158482077)

# فصل اول مقدمه

در دورانی که تکنولوژی به سرعت پیشرفت می‌کند و سیستم‌های اطلاعاتی یکپارچه نقش اساسی در زندگی امروزی ایفا می‌کنند، این پروژه در راستای ساخت یک سیستم خانه هوشمند تلاش می‌کند. در توسعه این سیستم تمرکز اصلی بر بهبود اتوماسیون خانگی، کنترل وسایل و بهینه‌سازی مصرف انرژی از طریق یک رابط کاربری مناسب است.  
در این فصل، ضمن معرفی خانه هوشمند و داده های هواشناسی، به معرفی یک معماری رایج در حوزه خانه هوشمند می‌پردازیم که از آن در این پروژه نیز بهره‌گیری شده است. سپس المان های تشکیل دهنده این شبکه و نقش آن ها را یک به یک بررسی خواهیم کرد.

## 1-1- خانه هوشمند

یک خانه هوشمند به یک مسکن گفته می‌شود که از فناوری های مدرن مبتنی بر شبکه و اتوماسیون برای افزایش کارایی، امنیت و کیفیت زندگی ساکنانش استفاده می‌کند. یکپارچگی دستگاه‌ها و سیستم‌های هوشمند به ساکنان این امکان را می‌دهد تا جنبه‌های مختلف خانه را از راه دور کنترل و نظارت کنند، یک محیط زندگی بهتر و متصل‌تر ایجاد می‌شود.



شکل 1. توپولوژی شبکه یک خانه هوشمند [1]

یک سیستم خانه هوشمند شامل اجزا مختلفی است که چند مورد از کلیدی ترین آن‌ها عبارتند از:

* دیوایس ها
* کنترلر
* سرور
* کلاینت

از بین المان های معرفی شده بعضی از آن ها مانند دیوایس ها و کنترلر در موقعیت خانه کاربر مستقر هستند. سرور این سیستم در دیتاسنتر به صورت ابری راه اندازی می‌شود. اما مورد آخر محدود به موقعیت مکانی خاصی نیست و در هر مکانی از جهان که به اینترنت دسترسی داشته باشد توانایی اتصال به سیستم را دارد.

## 1-2- داده های هواشناسی

داده‌های هواشناسی نقش حیاتی در صنایع مختلف ایفا می‌کنند، از کشاورزی و حمل و نقل تا مدیریت بلایای طبیعی و برنامه‌ریزی روزانه زندگی. این اهمیت و احساس نیاز باعث شده تا این داده ها از طرف سرویس دهنده های مختلفی در سطح اینترنت ارائه شود. این داده ها از طریق ایستگاه های هواشناسی و ماهواره ها در نقاط مختلف جهان جمع‌آوری می‌شوند.

### 1-2-1- ذخیره سازی داده‌ها

حجم زیادی از داده‌های تولید شده توسط ابزارهای مانیتورینگ هوا نیاز به سیستم‌های پیچیده ذخیره‌سازی دارد. سازمان‌های هواشناسی از راه‌حل‌های ذخیره‌سازی مبتنی بر ابر و سرورهای اختصاصی برای ذخیره و مدیریت حجم عظیم داده‌های هواشناسی استفاده می‌کنند. داده‌های تاریخی اغلب برای اهداف تحقیقاتی و بهبود مدل‌های پیش‌بینی درازمدت آرشیو می‌شوند

### 1-2-2- خدمات API داده‌ها

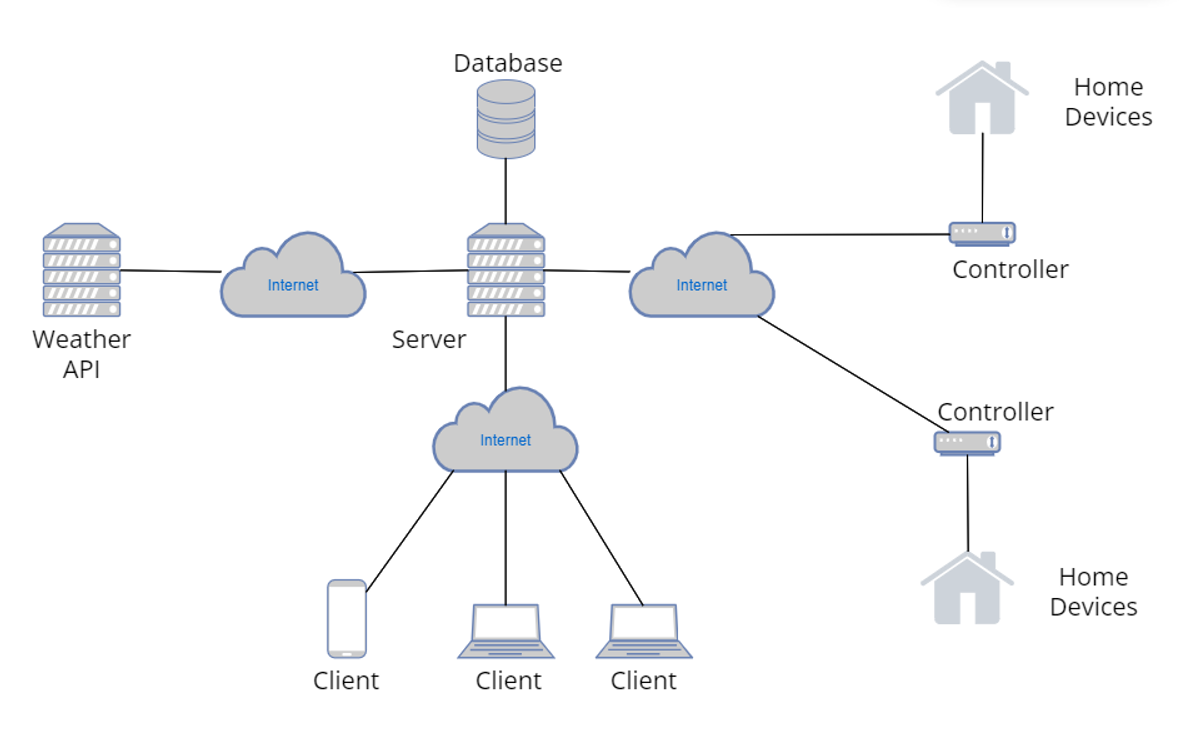
برخی سازمان ها، داده‌های هواشناسی را به صورت دسترسی برای عموم، کسب‌وکارها و توسعه‌دهندگان، به صورت خدمات رابط برنامه‌نویسی (API) ارائه می‌دهند. این API‌ها امکان ادغام سریع اطلاعات هواشناسی را به مختلف برنامه‌ها، وب‌سایت‌ها و خدمات فراهم می‌کنند. توسعه‌دهندگان می‌توانند از این API‌ها برای دریافت شرایط هوای فعلی، پیش‌بینی‌ها و داده‌های تاریخی به صورت برنامه‌نویسی استفاده کنند. API ها به طور معمول داده‌ها را به فرمت‌های استاندارد مانند JSON یا XML ارائه می‌دهند تا توسعه‌دهندگان بتوانند از این اطلاعات به راحتی در برنامه‌های خود استفاده کنند.

یکی از ارائه کنندگان این داده ها [openweathermap](https://openweathermap.org/) است که میتوان با درخواست های http از سرویس REST ارائه شده استفاده کرد. در این پروژه از سرویس رایگان این ارائه دهنده برای دریافت اطلاعات آب و هوا استفاده شده است.



## 1-3- معماری سیستم

در این بخش به معرفی معماری استفاده شده در طراحی این سیستم می‌پردازیم. در این طراحی از بین المان های مذکور در بخش 1-1 استفاده کرده‌ایم تا ساختار ارتباطی بین آن ها را نمایش دهیم. علاوه بر آن المان ها از اجزاء رایجی مانند پایگاه داده و سرور ارائه دهنده API نیز استفاده کردیم.



در این طراحی یک کنترلر نظیر هر خانه و تمام دیوایس هایش وجود دارد که وظیفه خودکار سازی فرایند های خانه و دریافت اطلاعات از سرور از طریق اینترنت را به عهده دارد. بخش مرکزی این سیستم، سرور است که وظیفه سرویس دادن به هر دو المان کنترلر های خانگی و کلاینت ها به عهده اوست. همچنین از سرور یک ارتباط داخلی به پایگاه داده داریم که برای عملیات های نوشتن و خواندن داده های سیستم از آن استفاده می‌شود. سرور برای دریافت اطلاعات هواشناسی نیاز به برقراری ارتباط با یک weather API دارد که این ارتباط نیز از طریق اینترنت برقرار می‌شود.

در فصل سوم به پیاده سازی بیشتر این المان ها از جمله کنترلر خانه، سرور و پایگاه داده خواهیم پرداخت. همچنین اطلاعات دقیق‌تر و با جزئیات بیشتری درباره نحوه برقراری این ارتباط ها و ابزار های مورد استفاده ارائه خواهد شد.

# فصل دوم مفاهیم و پیش‌‍‌زمینه‌ها‌

## 2-1- وب سرویس[[1]](#footnote-1)

وب سرویس مجموعه روش هایی برای تبادل پیام بین اجزاء موجود در اینترنت است. این ارتباطات می‌تواند بین برنامه های سرور و کاربر برقرار شود. از آنجایی که ارتباط بین برنامه های مختلف انجام می‌شود می‌توان گفت که وب سرویس ها در قلمرو لایه اپلیکیشن قرار دارند.

پروتکل های متنوعی برای ایجاد وب سرویس ها وجود دارد که در اینجا دو مورد از آن ها را ذکر و بررسی خواهیم کرد:

* REST (Representational State Transfer) :

REST یک سبک معماری است که از مدل کلاینت-سرور بدون ‌حالت[[2]](#footnote-2) برای ارتباط استفاده می‌کند. این ارتباط از طریق پروتکل HTTP صورت می‌گیرد و به طور گسترده برای ساختن وب سرویس های با مقیاس بزرگ، مورد استفاده قرار می‌گیرد.

این روش برای انتقال بدنه پیام از فرمت json استفاده می‌کند.

* SOAP (Simple Object Access Protocol):

SOAP یک پروتکل برای تبادل اطلاعات ساختارمند در خدمات وب است. از فرمت XML برای بدنه پیام استفاده می‌کند و معمولاً بر روی HTTP یا پروتکل‌های دیگر برای ردوبدل پیام اتکا می‌کند.

پروتکل و فریمورک های دیگری نیز وجود دارند که از آن ها می‌توان برای طراحی وب سرویس ها استفاده کرد از جمله GraphQL و gRPC که شناخته شده نیز هستند. این فریمورک ها معمولا در سیستم های با طراحی متفاوت مانند میکروسرویس[[3]](#footnote-3) ها استفاده می‌شوند.

طبق طراحی ارائه شده در فصل مقدمه برای چنین سیستمی که از معماری کلاینت – سرور استفاده می‌کند، استفاده از پروتکل REST می‌تواند مناسب باشد. به API ای با اتکا به قوانین پروتکل REST طراحی شده باشد یک RESTful API گویند. در ادامه به مقایسه اجمالی REST و SOAP می‌پردازیم و دلایل انتخاب REST را بیشتر بررسی می‌کنیم.

## 2-2- مقایسه REST و SOAP

انتخاب قالب مناسب برای استفاده در پیاده‌سازی API بستگی به نیازمندی های پروژه دارد. در جدول ارائه شده به مقایسه دو مورد مذکور از جنبه های مختلفی می‌پردازیم. در ادامه ویژگی های سیستم را با جنبه های مختلف هردو قالب تطابق می‌دهیم و مورد برتر را برای استفاده در پروژه انتخاب می‌کنیم.

|  |  |  |
| --- | --- | --- |
| شاخص | REST | SOAP |
| پروتکل | HTTP | HTTP |
| فرمت پیام | JSON/XML | XML |
| وضعیت حالت | Stateless | Stateless/Stateful |
| انعطاف | پشتیبانی از فرمت های پیام مختلف | تنها از XML پشتیبانی می‌کند |
| عملکرد | سریع به دلیل استفاده از JSON | کندتر به علت پردازش XML |
| خوانایی | بالا به دلیل استفاده از JSON | کم به دلیل استفاده از XML |
| امنیت | متکی به HTTP | پشتیبانی از WS-Security |
| مقیاس پذیری | بالا به علت بدون حالت بود | احتمالا کمتر به علت حالت دار بودن |
| راحتی پیاده‌سازی | ساده | پیچیده |

یک سیستم خانه هوشمند ممکن است از لحاظ میزان درخواست ها بار زیادی از شبکه را دریافت کند. یکی از راه های کاهش حجم ترافیک شبکه استفاده از فرمت داده json است. این فرمت نسبت به xml که در soap استفاده می‌شود، از برای انتقال داده یکسان دارای حجم کمتری است. در نتیجه از جنبه کنترل ترافیک شبکه استفاده از rest دارای ارجحیت است. همچنین استفاده از json بار پردازشی کمتری را به سرور وارد می‌کند. برای کاهش ترافیک شبکه سیستم از روش های دیگری نیز استفاده شده از جمله اینکه به جای ارسال داده برای پردازش در سمت کنترلر آن را در سمت سرور پردازش کنیم. این مورد در فصل سوم بیشتر توضیح داده می‌شود.

استفاده از rest در اکثر جنبه ها نسبت به soap برتری دارد، اما ممکن در شاخص امنیت ممکن است شبهه هایی پیش بیاید که آیا rest از لحاظ امنیت مشکل دارد؟ جواب این سوال بستگی به نحوه استفاده از rest دارد. تلاش rest سبک تر کردن قالب است و از overhead ناشی از امنیت صرف نظر کرده است به این دلیل که می‌توان بحث امنیت را در سطح اپلیکیشن پیاده‌سازی کرد و دچار مشکلی نبود. اما اگر برای امنیت در اپلیکیشن فکری نشود soap در این حالت نسبت به rest ارجحیت دارد. در این پروژه امنیت در سطح اپلیکیشن لحاظ شده و از استفاده از rest مشکلی ایجاد نخواهد کرد.

در نتیجه وب سرویس را در این پروژه به صورت یک RESTful API پیاده‌سازی خواهیم کرد و کنترلر های خانگی و کلاینت ها از این استاندارد برای بهره‌گیری از منابع تعریف شده در سرور استفاده خواهند کرد.

## 3-2- مفاهیم RESTful API

در این بخش قصد داریم چندین مفهوم مهم در RESTful API را بررسی کنیم. در فصل بعد نحوه استفاده از این مفاهیم در پروژه فعلی همراه با مثال های واقعی بررسی خواهد شد. در اینجا تنها به بررسی مفاهیم به صورت کلی می‌پردازیم.

### 2-3-1- منابع[[4]](#footnote-4)

انجام عملیات های پیشبینی شده در سیستم و همچنین عملیات هایی که کاربر مشتاق به انجام دادن آن ها است نیاز به موجودیت هایی دارد که بتوان این عملیات ها را روی آن ها انجام داد و تاثیرات ذخیره کرد. این منابع می‌توانند موارد انتزاعی یا عینی باشند. برای مثال اشیاء موجود در پایگاه داده مانند یک کاربر یا یک پروفایل را می‌توان از منابع سیستم به شمار آورد.

این منابع با یکدیگر ارتباطات مختلفی دارند. برای مثال در سیستم یک انتشارات کتاب، نویسنده و کتاب دو منبع به شمار می‌روند. هر کتاب دارای فقط یک نویسنده است و هر نویسنده ممکن است چندین کتاب داشته باشد. این یک مثال ساده از رابطه یک به چند در میان منابع است. انواع دیگری از ارتباطات از جمله یک به یک و چند به چند نیز ممکن از میان منابع حاکم باشد.

### 2-3-2- شناسه های یکنواخت منابع (URIs)[[5]](#footnote-5)

شناسه‌های یکنواخت (URIs) یک اصطلاح عمومی است که برای شناسایی و یافتن منابع در اینترنت استفاده می‌شود. این‌ شناسه ها از دو بخش اصلی تشکیل شده‌اند :URL (آدرس‌های یکنواخت منبع) و :URN (نام‌های یکنواخت منبع). یک URI وسیله‌ای برای شناسایی و دسترسی یکنواخت به یک منبع فراهم می‌کند.

عبارت دیگری که در ادامه از آن استفاده خواهیم کرد عبارت endpoint است. endpoint ها نیز همانند URI ها به منابع مختلف سیستم اشاره می‌کنند اما نگاه آن‌ها به منابع به عنوان یک سرویس است که در اختیار برنامه نویس قرار می‌گیرد در نتیجه رایج تر است که فضای توسعه یک API بیشتر از لفظ endpoint استفاده شود. endpoint ها و سرویس ها ممکن به یک سرویس اشاره کنند و نه یک منبع خاص. برای مثال سرویس بازیابی گذرواژه در یک endpoint خاص مانند https://api.example/api/password-recovery قرار داشته باشد. واضح است که این endpoint به منبع خاصی اشاره نمی‌کند و در حالت تعمیم یافته به یک سرویس اشاره داد.

در این پروژه به هردو نوع endpoint ها خواهیم پرداخت و از آن‌ها استفاده خواهیم کرد.

### 2-3-3- عملیات های CRUD

سیستم برای اینکه بتواند فعالیت های از پیش تعیین شده خودش را انجام دهد نیاز دارد که بتواند روی منابع ارائه شده توسط endpoint ها عملیات های زیر را انجام دهد:

* **C**reate
* **R**ead
* **U**pdate
* **D**elete

انجام این عملیات ها روی منابع از طریق فعل های HTTP انجام می‌شود که در ادامه فعل نظیر هرکدام از عملیات های بالا ذکر شده است:

* POST
* GET
* PUT/PATCH
* DELETE

جدول زیر مثالی از یک endpoint برای منبع کاربر است. هدف ساخت سرویسی است که تمام عملیات های CRUD را بتواند برای برنامه نویس فراهم کند. همانطور که در جدول زیر برای endpoint کاربران این امکان فراهم شده است. بنابراین برنامه نویس آزادی عمل مناسب را برای کار و پیاده سازی فرایند هایی که نیاز به دسترسی به منبع کاربر دارد را خواهد داشت.

|  |  |  |
| --- | --- | --- |
| عملیات | endpoint | |
|  | URN | HTTP verb |
| خواندن تمام کاربران | /users | GET |
| خواندن یک کاربر خاص | /users/{id} | GET |
| ساختن یک کاربر | /users | POST |
| بروزرسانی یک کاربر خاص | /users/{id} | PUT |
| حذف یک کاربر خاص | /users/{id} | DELETE |

## 2-4- ابزار های مورد استفاده

در روند توسعه المان های مختلف سیستم سعی شده است که برای هر بخش از ابزار های مناسب و مرتبط با نیازمندی ها استفاده شود. در ادامه ابزار های مورد استفاده در هر جزء و دلیل استفاده از آن در سیستم ذکر خواهد شد.

### 2-4-1- کنترل‌کننده خانه[[6]](#footnote-6)

کنترلر های درون خانه، کامپیوتر های کوچکی با توان پردازش نسبتا پایین خواهند بود که وظیفه دریافت اطلاعات از سرور اصلی را به عهده داشته و این اطلاعات را برای استفاده در وسایل خانه از جمله وسایل سرمایش و گرمایش استفاده می‌کنند. در نتیجه ابزار های مورد استفاده در این کنترلر ها باید از لحاظ پردازشی سبک باشند تا بتوان از سخت افزار محدود آن ها نهایت استفاده را برد.

در راستای همین هدف از فریمورک FastAPI برای پیاده‌سازی عملکرد کنترلر استفاده کردیم. این فریمورک مبتنی بر زبان پایتون است و تمرکزش بر سادگی پیاده‌سازی و سرعت بالای برنامه است. طبق [ادعای وبسایت رسمی](https://fastapi.tiangolo.com/benchmarks/) و بنچمارک های TechEmpower FastAPI از سریع ترین فریمورک های مبتنی بر پایتون و از این لحاظ تنها از دو فریمورک Uvicorn و Starlette عملکرد ضعیف تری دارد و این هم به این دلیل است که FastAPI به صورت داخلی از این دو فریمورک برای پیاده‌سازی خودش استفاده کرده است و منطقی است که از این دو عملکرد ضعیف تری داشته باشد. برای ارسال درخواست های HTTP از HTTPX به عنوان HTTP Client استفاده شده است.

لیست کامل نیازمندی های پروژه که شامل ابزار های استفاده شده در توسعه است از طریق فایل [requirements.txt](https://github.com/aminhalvaei/smarthome/blob/master/controller/requirements.txt) موجود در [مخزن گیب‌هاب](https://github.com/aminhalvaei/smarthome/tree/master) پروژه قابل دسترسی است.

### 2-4-2- سرور

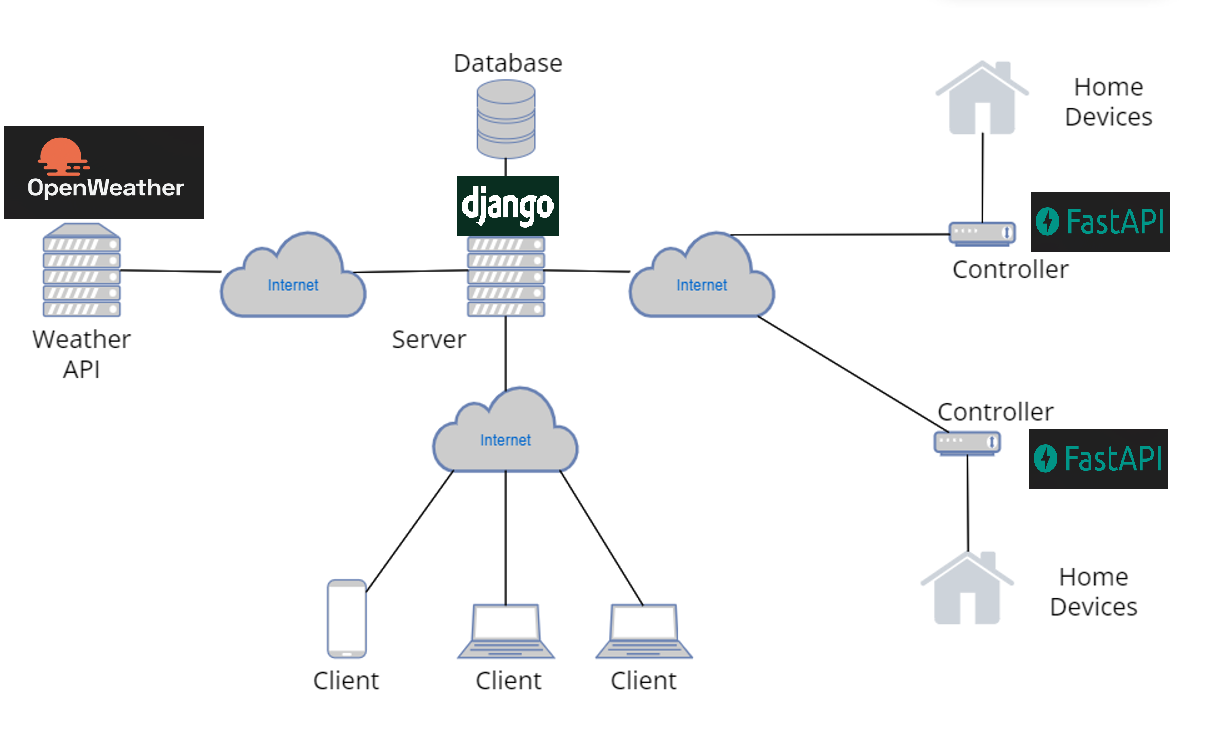
اصلی ترین فریمورک مورد استفاده در توسعه سرور اصلی سیستم جنگو[[7]](#footnote-7) است. ویژگی اصلی ای که باعث انتخاب جنگو برای توسعه این سرور انتخاب شد مقیاس‌پذیری آن است. در یک سیستم خانه هوشمند به مرور زمان حجم ردوبدل شدن اطلاعات می‌تواند بسیار زیاد شود. جنگو با مکانیزم های داخلی بهینه‌سازی و استراتژی های کش کردن اطلاعات تا حد خوبی از عهده این کار برمی‌آید.

علاوه بر این موضوع جنگو دارای ویژگی های گسترده ای است که کار توسعه را بسیار آسان تر می‌کند و در ادامه نیز نگهداری سیستم نیز به خوبی قابل انجام خواهد بود. دو مورد از این ویژگی های Django ORM و Admin Panel هستند.

Django ORM به عنوان یک واسط بین سرور و دیتابیس عمل می‌کند و با ارائه ویژگی هایی پیچیدگی کار با یک محیط پایگاه داده مجزا را کاهش می‌دهد. پنل ادمین نیز امکانی است که بررسی صحت و دیباگ کردن پروژه را کاملا برای توسعه دهنده راحت تر و ملموس تر می‌‌کند. مزیت دیگری که پنل ادمین برای توسعه دارد این است که طبق متد های توسعه چابک[[8]](#footnote-8) امکان نمایش و توضیح کار انجام شده در پروژه را برای کاربر نهایی و ذینفعان فراهم می‌کند.

علاوه بر جنگو برای پیاده‌سازی یک RESTful API نیاز به فریمورکی داریم که پشتیبانی از REST را در جنگو، فراهم آورد. برای این نیاز به نصب فریمورک django-rest-framework داریم که به اختصار به عنوان drf شناخته می‌شود.

لیست کامل ابزار مورد استفاده برای توسعه و اجرای سرور در محیط پروداکشن در فایل [requirements.txt](https://github.com/aminhalvaei/smarthome/blob/master/backend/requirements.txt) موجود در [مخزن گیت‌هاب](https://github.com/aminhalvaei/smarthome/) پروژه ارائه شده است.



# فصل سوم روش انجام پروژه

در این فصل به توضیح روند انجام پروژه و بخش های پیاده‌سازی شده می‌پردازیم. ابتدا به بررسی نیازمندی های کاربر و ویژگی های سیستم به صورت دقیق می‌پردازیم و پس از آن چندین مورد از طراحی های صورت گرفته در سیستم از جمله فرایند ها و روابط موجودیت های پایگاه داده را بررسی می‌کنیم. در ادامه ساختار دایرکتوری پروژه و کاربرد هرکدام از فایل های موجود در آن را شرح می‌دهیم. در یک بخش مجزا سرویس های ارائه شده برای کنترلر و کلاینت را لیست می‌کنیم. در بخش آخر نیز برخی از نکات بهینه‌سازی و امنیت سیستم را بررسی خواهیم کرد.

## 3-1- نیازمندی های کاربر

نیازمندی های کاربر سیستم به انواع مختلفی شکسته شده است و در ادامه نیازمندی های هر بخش ذکر شده است.

* احراز هویت:

1. ثبت نام
2. ورود به سیستم
3. خروج از سیستم
4. تایید ایمیل
5. تغییر گذرواژه
6. بازیابی گذرواژه

* خانه:

1. اضافه کردن خانه
2. حذف خانه
3. ویرایش خانه
4. تنظیم درجه عایق‌بندی خانه
5. اضافه کردن موقعیت مکانی
6. تغییر موقعیت مکانی
7. تعریف شهر، استان و کشور
8. ثبت کنترلر
9. تنظیم کنترلر به حالت دستی یا اتوماتیک
10. فعال یا غیرفعال کردن کنترلر

* وضعیت هوا:

1. مشاهده وضعیت فعلی سیستم سرمایش گرمایش خانه
2. مشاهده وضعیت آب و هوا بیرون از خانه
3. تنظیم ترجیحات در وضع هوای خانه
4. مشاهده ترجیحات فعال
5. تغییر ترجیحات وضع هوای خانه
6. مشاهده وضعیت به تفکیک پارامتر های هواشناسی

## 3-2- طراحی اجزا و فرایند‌ها

با توجه به نیازمندی های ارائه شده در بخش قبل نیاز به طراحی پایگاه داده ای داریم که توانایی پشتیبانی از این ویژگی‌ها را داشته باشد. پایگاه داده از نوع رابطه ای برای این سیستم انتخاب شده است و طراحی مدل رابطه ای آن در ادامه ارائه شده است.

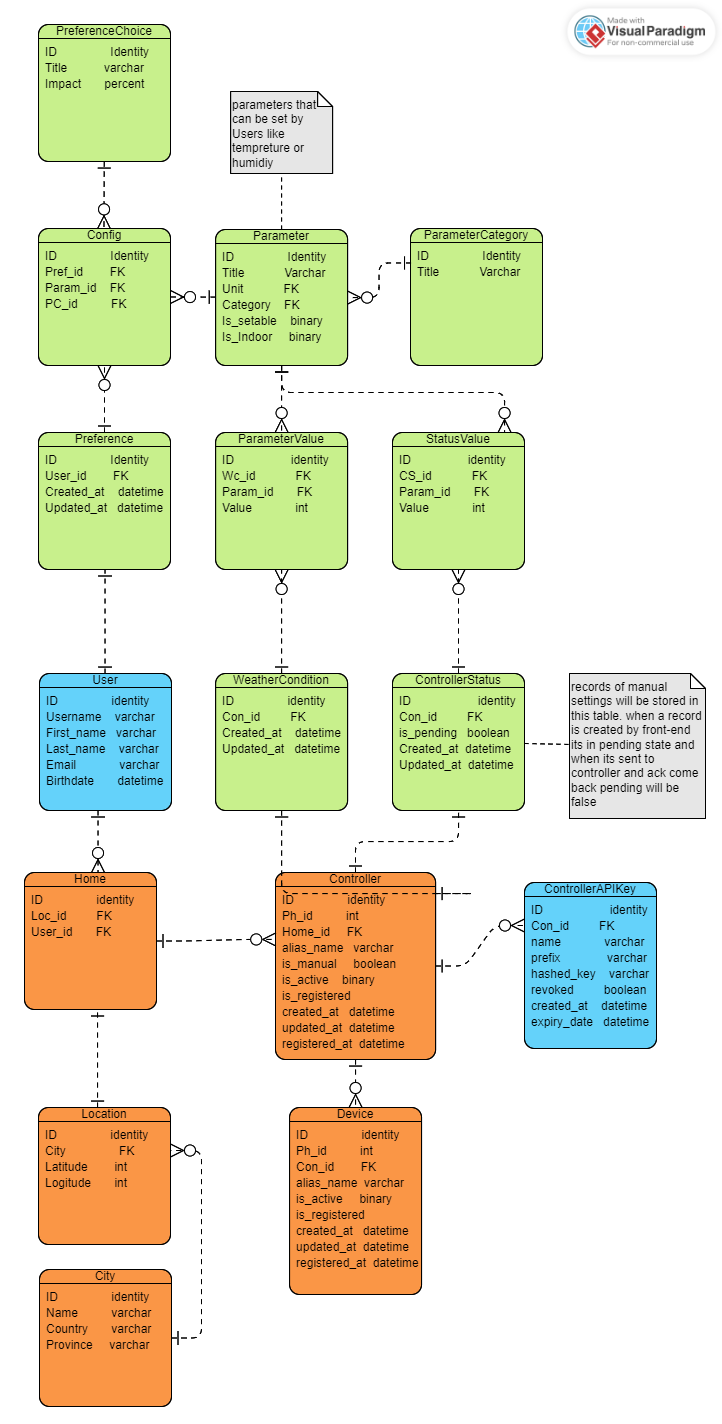
### 3-2-1- طراحی رابطه موجودیت های پایگاه داده[[9]](#footnote-9)

|  |  |
| --- | --- |
| **اکانت ها (Accounts)** | |
| **نام جدول** | **محتوا** |
| User | نگه‌داری اطلاعات کاربر |
| ControllerAPIKey | نگه‌داری اطلاعات احراز هویت کنترلر |

در تصویر ارائه شده برای طراحی پایگاه داده، رنگ های جداول نشان دهنده هر قسمت از پایگاه داده هستند. برای مثال رنگ سبز جداول مربوط به وضعیت آب و هوا را نمایش می‌دهد. در ادامه به توضیح ماهیت این جداول و ارتباط آن‌ها با یکدیگر می‌پردازیم.

|  |  |
| --- | --- |
| **هواشناسی (weather)** | |
| **نام جدول** | **محتوا** |
| Parameter | پارامتر های هواشناسی مانند دما و فشار |
| ParameterCategory | دسته‌بندی پارامتر ها |
| Preference | ترجیحات کاربر |
| PreferenceChoice | گزینه های از پیش تعریف شده برای ترجیحات |
| Config | مقدار ثبت شده برای ترجیحات کاربر |
| WeatherCondition | آخرین وضعیت هوا |
| ControllerStatus | آخرین وضعیت کنترلر خانه |
| ParameterValue | مقدار یک پارامتر برای وضعیت هوا |
| StatusValue | مقدار یک پارامتر برای وضعیت کنترلر |

|  |  |
| --- | --- |
| **خانه (Home)** | |
| **نام جدول** | **محتوا** |
| Home | اطلاعات مربوط به خانه |
| Controller | اطلاعات مربوط به کنترلر |
| Device | اطلاعات مربوط به دستگاه های خانه |
| Location | موقعیت مکانی خانه |
| City | اطلاعات محل سکونت |



### 3-2-2- طراحی روند فعالیت های کنترلر

به دلیل این که فعالیت های سمت کنترلر خانگی فاقد رابط کاربری هستند در نتیجه عملکرد آن ها باید از طریق روند های از پیش تعیین شده دقیق و استفاده از سرویس های مناسب سمت سرور به صورت خودکار صورت پذیرد.

همچنین قابل ذکر است که فعالیت های کلاینت اکثرا به ارتباط به یک نوع خاص از مدل های سرور و انجام عملیات های CRUD خلاصه می‌شود برای همین در این قسمت تمرکز روی بحث پیچیده تر یعنی تعریف ترتیب فعالیت ها در کنترلر خانگی است. در ادامه به توصیف این روند ها برای چند مورد از مهم ترین نیازمندی های سیستم در سمت کنترلر پرداخته ایم.

**ثبت کنترلر در سیستم:**

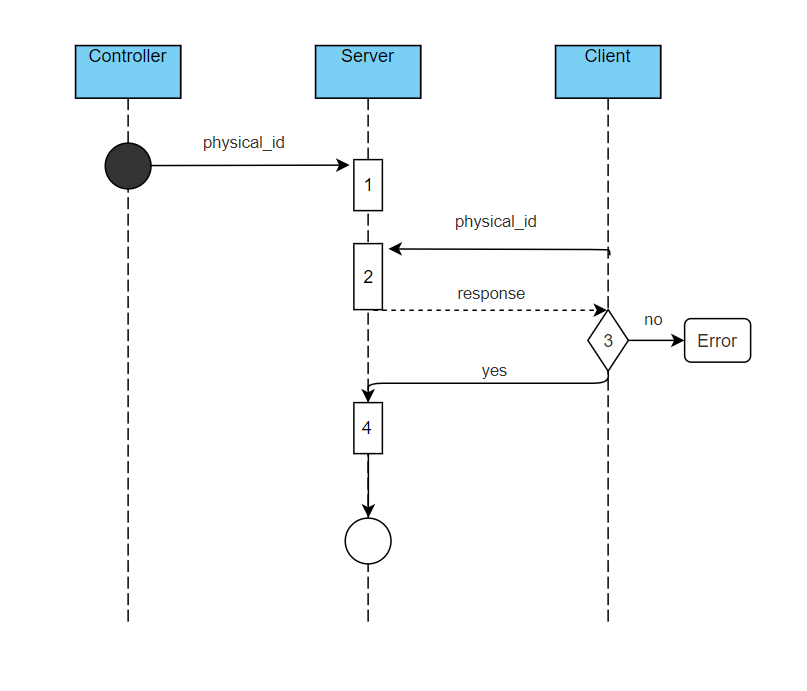
در این فرایند فرض شده است که کنترلر مورد استفاده در خانه که درخواست ثبت در سیستم را ارسال می‌کند قبل از عرضه به مشتری به صورت اولیه در سیستم ذخیره شده است و در حالت ثبت نشده قرار دارد. همچنین در فایل مربوط به تنظیمات کنترلر که configs.py نام دارد که شامل پارامتر های مختلفی است از جمله آن پارامتر هایی که برای اتصال به سرور و ثبت مورد نیاز هستند.

دو مورد از این پارامتر ها در ادامه ذکر شده اند:

* CONTROLLER\_PHYSICAL\_ID
* CONTROLLER\_API\_KEY

مورد اول یعنی شناسه فیزیکی یک کد یکتا و غیرقابل تغییر است که کنترلر از طریق آن از دیگر کنترلر ها متمایز می‌شود. و مورد دوم کلید مورد نیاز برای احراز هویت در سمت سرور است که کنترلر برای دسترسی به تمام خدمات سرور به آن نیاز دارد. در بخش امنیت بیشتر به توصیف نقش این کلید خواهیم پرداخت.

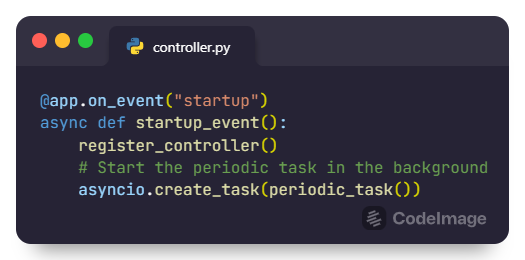
در شکل زیر شمای کلی این روند ارائه شده است:



این فرایند ثبت کنترلر در سرور شامل دو بخش است:

1. ارسال درخواست اولیه توسط کنترلر
2. ادامه روند و ثبت نهایی توسط کاربر

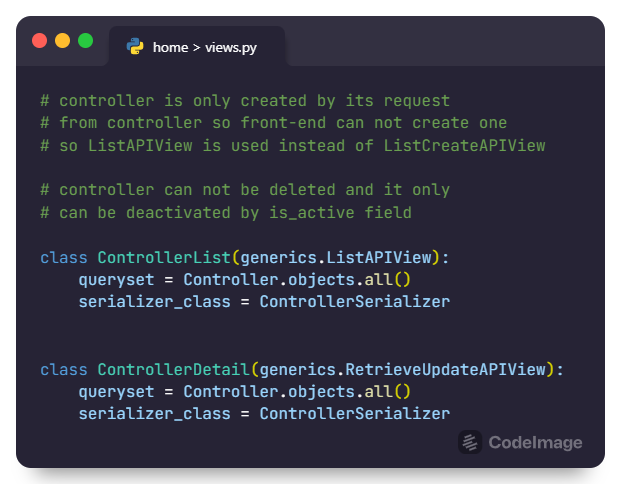
شروع کننده فرایند کنترلر است که پس از روشن شدن اولیه درخواست ثبت خودش را برای سرور ارسال می‌کند. این رفتار در قطعه کد زیر قابل مشاهده است.



پس از آن یک پیام HTTP شامل physical\_id و کلید احراز هویت به سمت سرور ارسال می‌شود و در صورت معتبر بودن این پیام کنترلر به حالت ثبت اولیه در می‌آید. ثبت اولیه از طریق True شدن مقدار خصیصه ای به نام is\_register\_pending از مدل Controller صورت می‌گیرد. قطعه کد زیر این عملکرد را نشان می‌دهد.



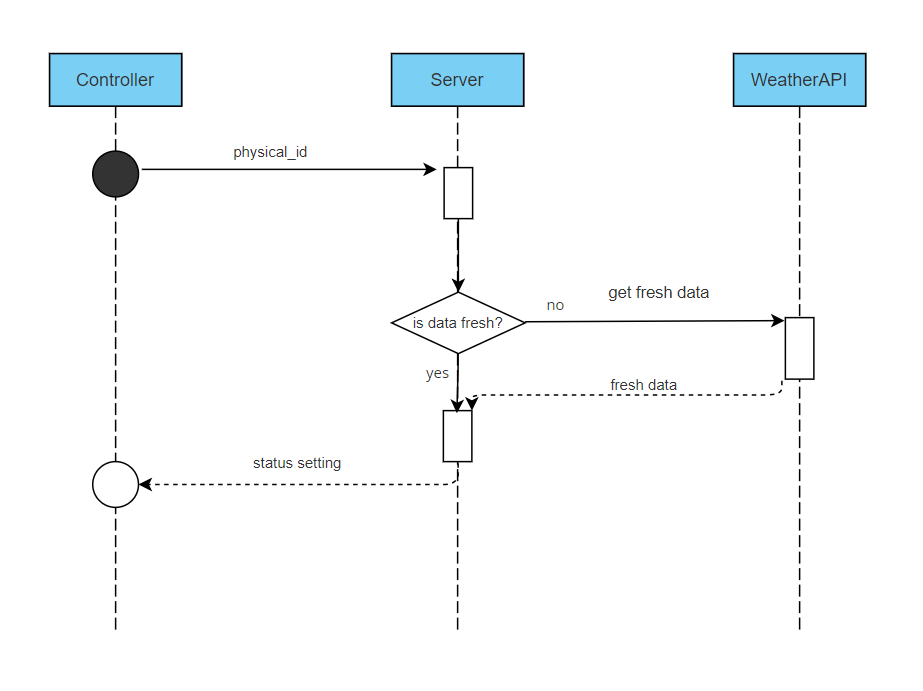
در بخش دوم فرض شده است که کاربر شناسه physical\_id مربوط به کنترلر خودش را می‌داند مثلا از روی جعبه کنترلر خریداری شده این شناسه را بدست می‌آورد. کلاینت با استفاده از عملیات Read از مجموعه عملیات های CRUD ارائه شده توسط سرور استفاده کند و با یک پرس و جو به دنبال کنترلر مورد نظرش بگرد. نتیجه پرس و جو به کلاینت برگردانده می‌شود و در صورت پیدا شدن کنترلری با مشخصات ارائه شده توسط کاربر که در حالت ثبت اولیه است کلاینت با یک عملیات Update روی شی مرتبط با کنترلر مورد نظر، با True کردن خصیصه is\_registered آن را به حالت ثبت نهایی درمی‌آورد. قطعه کد زیر عملیات های CRUD قابل انجام بر روی مدل Controller را نمایش می‌دهد.



حال پس از اینکه کنترلر ثبت نهایی شد این امکان را دارد که از سرویس دریافت وضعیت آب و هوا استفاده کند.

**دریافت تنظیمات هواشناسی از سرور:**

پس از اینکه کنترلر در سیستم ثبت نهایی شد کنترلر شروع به گرفتن تنظیمات پارامتر های هواشناسی از سمت سرور می‌کند. در این فرایند نیز ارائه کلید احراز هویت توسط کنترلر الزامی است و در صورت عدم تطابق کلید با کلید ذخیره شده در سمت سرور یا گذشتن از تاریخ انتقضای کلید، کنترلر به این سرویس دسترسی نخواهد داشت. شکل زیر شمای کلی فرایند دریافت نتظیمات هواشناسی از سرور را نشان می‌دهد.



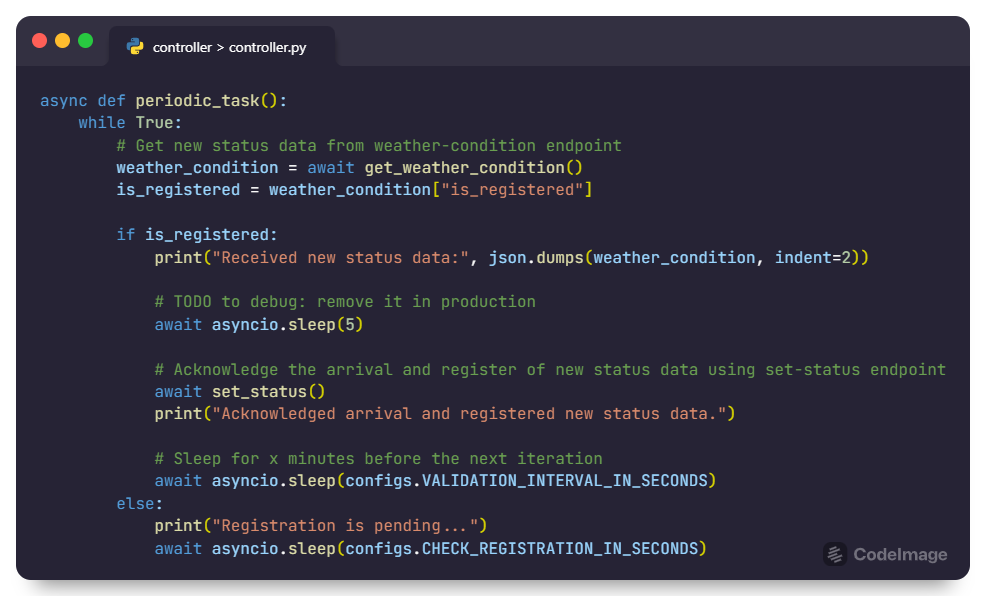
در فرایند بالا المان شروع کننده کنترلر است که یک پیام HTTP شامل physical\_id که در body پیام در فرمت json قرار دارد را ارسال می‌کند. همچنین در header مربوط به احرازهویت یعنی Authorization نیز کلید احراز هویت API قرار دارد. قطعه کد زیر پیاده سازی این درخواست را نمایش می‌دهد.



رفتار کنترلر خانگی به این صورت است که درخواست های دریافت تنظیمات پارامتر های هواشناسی را به صورت متناوب به سمت سرور جنگو ارسال می‌کند. دیاگرام ارائه شده نشان دهنده یک دوره تناوب از این رفتار کنترلر های خانگی است. مقدار این دوره تناوب در فایل configs.py به صورت زیر تعریف شده است.



رفتار متناوب توصیف شده ناشی از پیاده‌سازی زیر است.



پس از ارسال این پیام در سرور این درخواست دریافت می‌شود و پس از احراز هویت و بررسی معتبر بودن درخواست در صورت تازه بودن اطلاعات موجود در سرور، بدون درخواست از OpenWeather API سرور جنگو اطلاعات مورد نیاز را به کنترلر ارسال می‌کند. اما در صورت اینکه داده ها منقضی شده باشند یک درخواست به OpenWeather API ارسال می‌شود تا داده های تازه دریافت شوند و پس از پردازش و ذخیره سازی در سرور، اطلاعات را به کنترلر ارسال می‌کند.

زمان انقضای داده ها در فایل configs.py متعلق به سرور است در متغیر WEATHER\_VALID\_DURATION ذخیره شده است و تابع زیر وظیفه بررسی این تاریخ انقضا را بر عهده دارد.



روند توصیف شده در قالب قطعه کد زیر پیاده‌سازی شده است.



لازم به ذکر است که در دیاگرام های ارائه شده تاخیر شبکه ناچیز در نظر گرفته شده است و به همین دلیل است که فلش های ارسال پیام ها و جواب ها مورب نیستند.

## 3-3- ساختار پروژه

پوشه اصلی پروژه که حاوی تمام فایل های پروژه است smarthome نام دارد. با توجه به اینکه توسعه پروژه به دو بخش اصلی سرور جنگو و کنترلر خانگی تقسیم می‌شود، پروژه به دو پوشه اصلی به نام های backend و controller تقسیم شده است. این ساختار از [مخزن[[10]](#footnote-10) پروژه](https://github.com/aminhalvaei/smarthome/) که در گیت‌هاب قرار گرفته است قابل مشاهده است. علاوه بر این دو پوشه اصلی یک پوشه به نام documents نیز در مخزن قرار گرفته که تنها شامل گزارش پروژه است. دو فایل دیگر به نام های .gitignore و README.md نیز در مسیر اصلی پروژه وجود دارند که به ترتیب وظیفه نادیده گرفتن فایل بعضی از فایل های پروژه و نمایش اطلاعات درباره پروژه را به عهده دارند.



در ادامه به توصیف ساختار هرکدام از پوشه های اشاره شده می‌پردازیم.

### 3-3-1- سرور جنگو (backend)

این پوشه دارای ساختار استاندارد یک پروژه جنگو است که با استفاده از دستور زیر بدست می‌آید:

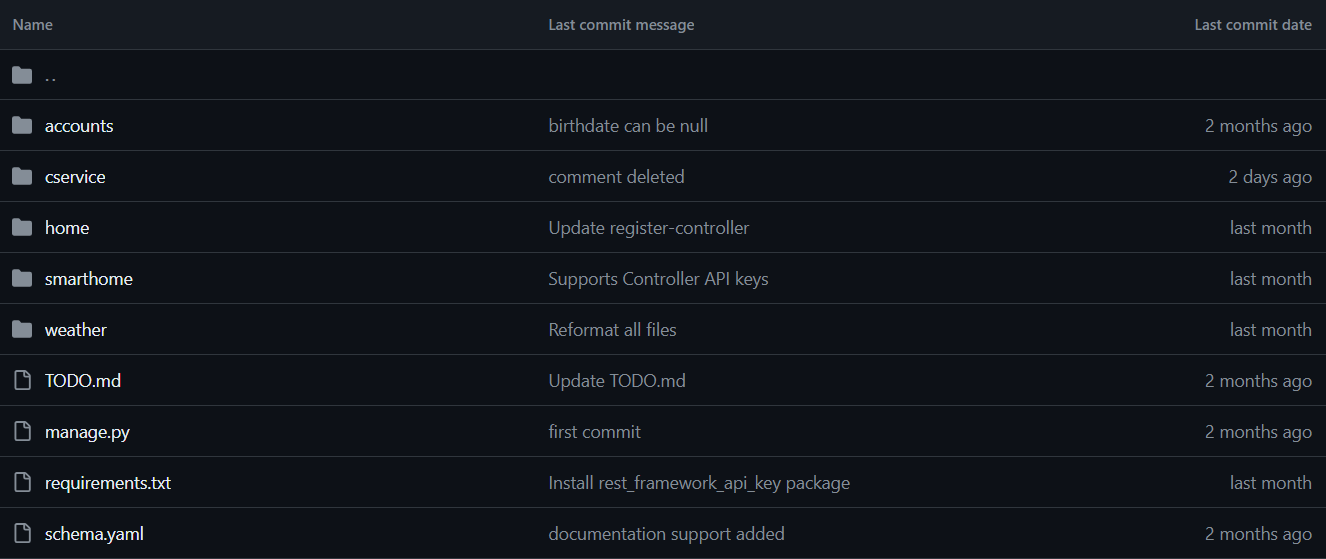
django-admin startproject smarthome

یک پروژه جنگو قابل تقسیم به بخش های مجزایی به نام اپلیکیشن ها است. هر اپلیکیشن شامل عملکرد خاصی است که در ادامه توضیح داده خواهد شد. لیست اپلیکیشن ها به صورت زیر است:

* accounts
* cservice
* home
* weather

اپلیکیشن accounts شامل مدل user ها است و به صورت مستقیم سرویسی را ارائه نمی‌دهد بلکه از مدل ها موجود در آن در اکثر اپلیکیشن های دیگر استفاده می‌شود. در اپلیکیشن های home و weather به ترتیب سرویس های مربوط به مدل های متلعق به خانه و آب و هوا مانند وجود دارند. همه سرویس هایی که کمی از حالت CRUD فاصله گرفته اند و به کنترلر خدمات می‌دهند در این اپلیکیشن قرار دارند. مانند سرویس ها ثبت کنترلر ها در سیستم و دریافت تنظیمات پارامتر های هواشناسی از سرور که پیش‌تر در بخش قبلی به تفصیل توضیح داده شدند.

علاوه بر این پوشه ها که مربوط به اپلیکیشن ها هستند یک پوشه دیگر وجود دارد که همنام با پوشه اصلی پروژه است و smarthome نام دارد. این پوشه با اجرای دستور startproject توسط خود جنگو ساخته شده است. شکل زیر ساختار توصیف شده از پوشه backend را توصیف می‌کند.



در تصویر بالا علاوه بر پوشه های مذکور 4 فایل دیگر وجود دارد که وظیفه آن ها به شرح ذیل است:

* TODO.md : این فایل در فرایند توسعه برای سازماندهی فعالیت هایی که در آینده باید در پروژه انجام شوند مورد استفاده قرار گرفته است.
* manage.py : این فایل توسط جنگو تولید شده است و وظیفه اجرای بعدی از دستورات مانند runserver ، migrate و دیگر عملیات ها را به عهده دارد.
* requirements.txt : نیازمندی های توسعه و اجرای سرور جنگو در که شامل کتابخانه ها و ابزار های مختلفی است در این فایل قرار گرفته اند تا بتوان در صورت استقرار پروژه در یک ماشین دیگر بتوان نیازمندی های پروژه را به راحتی در آن ماشین نیز نصب کرد.
* schema.yaml : این فایل وظیفه تولید مستندات مربوط به endpoint های ارائه شده توسط پروژه را به عهده دارد.

ساختار درونی هر کدام از اپلیکیشن ها و فایل های موجود در آن ها بخش اصلی توسعه این سیستم هستند که در بخش پیاده سازی در آینده درباره آن ها توضیح داده خواهد شد.

### 3-3-2- کنترلر خانگی (controller)

توسعه کنترلر با استفاده از ابزاری متفاوت صورت گرفته و نیازمندی های خاص خودش را دارد. بنابراین تصمیم بر این شد که یک پوشه مجزا برای توسعه کنترلر در نظر گرفته شود. ساختار این پوشه در تصویر بعدی قابل مشاهده است.



پوشه controller از یک دو فایل controller.py و requirements.txt و یک پوشه configs تشکیل شده است. فایل controller.py شامل منظق اصلی کنترلر یعنی شروع فرایند ثبت در سیستم و ارسال درخواست های دریافت تنظیم پارامتر های هواشناسی می‌شود و فایل requirements.txt نیز کاربرد مشابهی با آنچه قبلا برای سرور جنگو توضیح داده شد دارد.

فایل controller.py برای عملکرد خودش نیاز به بعضی از پارامتر های ثابتی مانند آدرس سرور جنگو و شناسه فیزیکی دارد که این اطلاعات در فایل configs.py ذخیره شده اند. برای اینکه امکان استفاده از configs در controller.py به صورت یک package پایتون وجود داشته باشد این فایل همراه با یک فایل خالی با نام \_init\_.py در یک پوشه به نام configs.py قرار داده شده تا این عملکرد بدست آید.

## 3-4- پیاده‌سازی

در این بخش به توصیف عملکرد و کد های پیاده‌سازی شده در هرکدام از اپلیکیشن های accounts، home، weather و cservice خواهیم پرداخت. در هرکدام از این اپلیکیشن ها فایل های مختلفی وجود دارد که منظق برنامه را مشخص می‌کند.

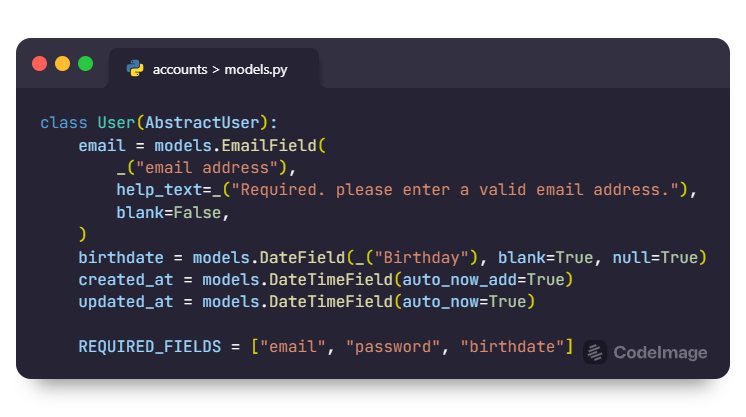
مهم ترین این فایل ها عبارتند از:

* urls.py : این فایل شامل endpoint های تعریف شده در سیستم است. همچنین ارتباط endpoint ها با منطق سیستم نیز در این فایل برقرار می‌شود.
* views.py: منطق اصلی هر اپلیکیشن در این فایل ذخیره شده است.
* serializers.py : فرمت داده های ورودی و خروجی به view ها را تعیین می‌کند.
* models.py : شامل کلاس هایی است که متناظر با جدول های پایگاه داده هستند و وظیفه انجام عملیات های روی پایگاه داده را به عهده گرفته اند.

این فایل ها فقط در قسمت پیاده‌سازی سرور جنگو (backend) مورد استفاده قرار گرفته اند و در بخش پیاده سازی کنترلر از آن ها استفاده نشده است بنابراین برای کنترلر تنها به بررسی کد controller.py که در برگیرنده تمام منطق کنترلر است خواهیم پرداخت.

### 3-4-1- اپلیکیشن accounts

برخلاف اپلیکیشن های دیگر در accounts ما به ارائه سرویسی به بیرون از سرور نمی‌پردازیم. در اینجا هدف ارائه کلاس کاربر اصلی سیستم است که به عنوان یک مدل در فایل models.py طبق تصویر زیر تعریف شده است.



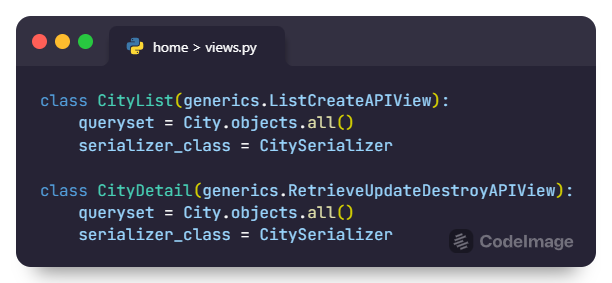
در این کلاس از یک کلاس دیگر به نام AbstractUser ارث بری کرده ایم. این کلاس شامل فیلد هایی مانند username، first\_name، last\_name و چندین فیلد دیگر می‌شود. فیلد email در کلاس پدر نیز وجود دارد اما ما با تعریف دوباره آن ویژگی دلخواهمان که خالی نبودن این فیلد است را به آن اضافه کردیم. همچنین 3 فیلد تاریخ تولد و تاریخ های ساخت اکانت و بروزرسانی آن نیز به مدل اضافه شده اند. از این مدل در تمام اپلیکیشن های بعدی استفاده شده است.

### 3-4-2- اپلیکیشن home

در این برنامه سرویس هایی که مربوط به موجودیت های مربوط به خانه هستند را در معرض کاربران بیرون از سرور (مانند برنامه نویس front-end) قرار داده ایم. لیست endpoint های ارائه شده از قرار زیر است که از فایل [urls.py](https://github.com/aminhalvaei/smarthome/blob/master/backend/home/urls.py) قابل بررسی هستند.

|  |  |  |  |
| --- | --- | --- | --- |
| name | view | operations | Endpoint |
| city-list | CityList | CR | cities/ |
| city-detail | CityDetail | RUD | cities/<int:pk> |
| location-list | LocationList | CR | locations/ |
| location-detail | LocationDetail | RUD | locations/<int:pk> |
| home-list | HomeList | CR | homes/ |
| home-detail | HomeDetail | RUD | homes/<int:pk> |
| controller-list | ControllerList | R | controllers/ |
| controller-detail | ControllerDetail | RU | controllers/<int:pk> |
| device-list | DeviceList | R | devices/ |
| device-detail | DeviceDetail | RU | devices/<int:pk> |

منطق عملکرد این endpoint ها تنها ارائه خود مدل به بیرون است و سرویس خاصی را انجام نمی‌دهند. یک نمونه view که این عملکرد را برای ما پیاده می‌کند به این شکل است.



برای پیاده‌سازی این view ها از نوع generic ها استفاده کردیم که کار را برای پیاده‌سازی endpoint هایی که فقط CRUD را انجام می‌دهند مناسب است. در این کلاس ها تنها مجموعه داده ها و کلاس سریالایزر را مشخص کرده ایم و بقیه کار به صورت اتوماتیک توسط خود جنگو انجام خواهد شد.

شایان ذکر است که تمام همیشه امکان استفاده از generic view ها وجود ندارد. برای مثال در فرایند دریافت تنظیمات پارامتر های هواشناسی باید در یک view تمام عملکرد های مورد نیاز را کدنویسی کنیم.

**شهر و موقعیت مکانی (City & Location) :**



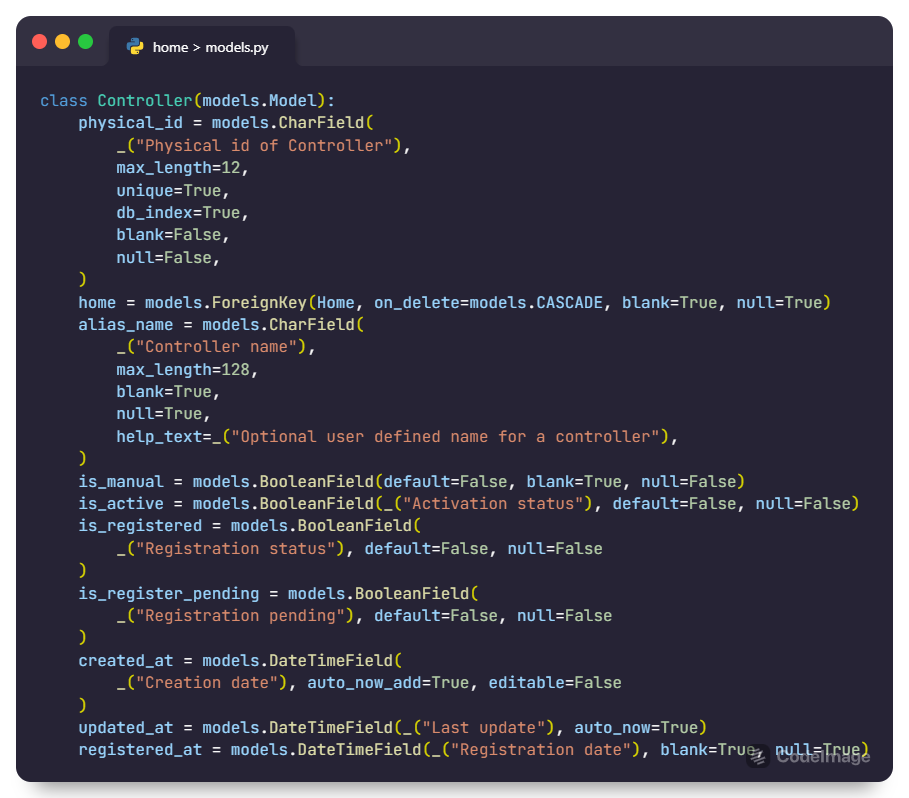
مدل شهر را داریم که شامل خصیصه های نام شهر، کشور و استان است. در مدل موقعیت مکانی نیز دو خصیصه اصلی طول و عرض جغرافیایی هستند که ذخیره شده اند علاوه بر آن ها یک کلید خارجی به جدول شهر داریم که شهر مربوط به آن موقعیت مکانی را مشخص می‌کند.

**خانه (Home) :**



مدل خانه شامل خصیصه های نام، کاربر، و موقعیت مکانی مربوط به خانه می‌شود. در اینجا اولین استفاده از مدل کاربر که قبلا در اپلیکیشن accounts تعریف شد را مشاهده می‌کنیم.

**کنترلر (Controller) :**



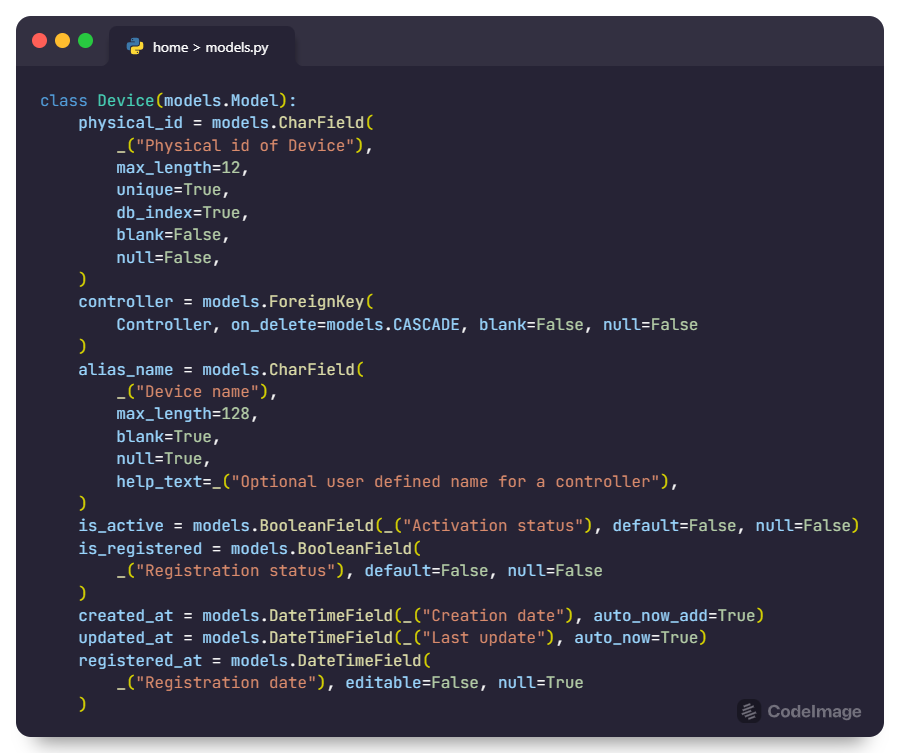
همانطور که قبلا دیدیم به هر کنترلر یک شناسه فیزیکی یکتا تعلق می‌گرفت. در این مدل این شناسه با نام physical\_id که یک رشته به طول 12 کاراکتر است. همچنین به دلیل اینکه بسیاری از پرس‌وجو هایی از پایگاه داده سیستم صورت می‌گیرد بر اساس شناسه فیزیکی هستند، برای افزایش سرعت عملکرد پایگاه داده در برگرداندن داده ها این خصیصه را به عنوان یک db\_index تعریف کرده ایم.

خصیصه های بعدی به ترتیب به کلید خارجی خانه و نام کنترلر اشاره دارند. پس از آنها 3 خصیصه از نوع بولین وجود دارند که وضعیت های زیر را مشخص می‌کنند.

* is\_manual : خودکار یا دستی بودن یک کنترلذ
* is\_active : وضعیت فعال بودن کنترلر
* is\_registered : وضعیت ثبت نهایی کنترلر
* is\_register\_pending : وضعیت ثبت اولیه کنترلر

و در ادامه 3 خصیصه داریم که به ترتیب تاریخ های ساخت کنترلر، آخرین بروزرسانی اطلاعات و ثبت نهایی کنترلر را نشان می‌دهند.

**وسایل (Device) :**



عملکرد این موجودیت تا حد زیادی شبیه به کنترلر است و خصیصه های مشترک زیادی دارند. یک خصیصه به نام controller نیز در این موجودیت وجود دارد که آن را به مدل کنترلر مرتبط می‌کند.

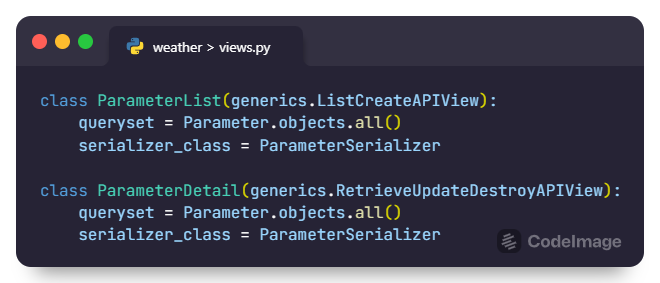
### 3-4-3- اپلیکیشن weather

سرویس هایی که مربوط به آب و هوا هستند در این قسمت معرفی می‌شوند. این سرویس ها در قالب endpoint ها در فایل urls.py در دسترس کاربران بیرونی سیستم قرار گرفته است. لیست endpoint های این اپلیکیشن به صورت زیر است.

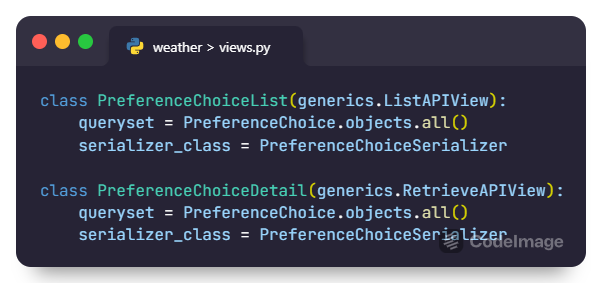
|  |  |  |  |
| --- | --- | --- | --- |
| name | view | operations | Endpoint |
| parameter\_list | ParameterList | CR | parameters/ |
| parameter\_detail | ParameterDetail | RUD | parameters/<int:pk> |
| parameter\_category\_list | ParameterCategoryList | R | parameters-categories/ |
| parameter\_category\_detail | ParameterCategoryDetail | R | parameters-categories/<int:pk> |
| preference\_list | PreferenceList | CR | preference/ |
| preference\_detail | PreferenceDetail | RUD | preference/<int:pk> |
| preference\_choice\_list | PreferenceChoiceList | R | preference-choice/ |
| preference\_choice\_detail | PreferenceChoiceDetail | R | preference-choice/<int:pk> |
| config\_list | ConfigList | CR | config/ |
| config\_detail | ConfigDetail | RUD | config/<int:pk> |
| weather\_condition\_list | WeatherConditionList | CR | weather-condition/ |
| weather\_condition\_detail | WeatherConditionDetail | RUD | weather-condition/<int:pk> |
| parameter\_value\_list | ParameterValueList | CR | parameter-value/ |
| parameter\_value\_detail | ParameterValueDetail | RUD | parameter-value/<int:pk> |
| controller\_status\_list | ControllerStatusList | CR | controller-status/ |
| controller\_status\_detail | ControllerStatusDetail | RUD | controller-status/<int:pk> |
| status\_value\_detail | StatusValueList | CR | status-value/ |
| status\_value\_list | StatusValueDetail | RUD | status-value/<int:pk> |

این endpoint ها نیز مانند endpoint های مربوط به home تنها دسترسی به انجام عملیات های CRUD را در دسترس کاربران (front-end) قرار می‌دهد.

در اینجا نمونه ای از view های نوشته شده با استفاده از کلاس های generic views آورده شده است.

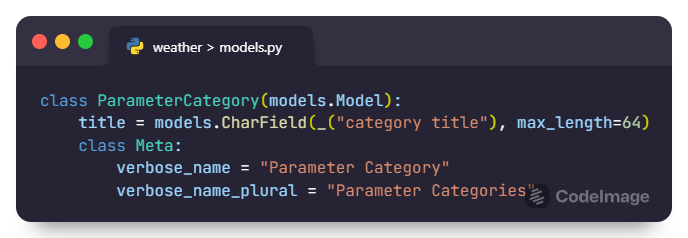


بعضی از مدل های مربوط به این endpoint ها مدل هایی هستند که باید توسط ادمین سیستم تنظیم شوند بنابراین همانطور که از جدول قبل مشخص است، کاربران از جمله برنامه نویس front-end، تنها دسترسی Read را به این مدل ها دارند. برای مثال مدل PreferenceChoice که انتخاب های ممکن برای ترجیحات کاربر و میزان تاثیرشان را ذخیره می‌کند یک جدول با اعضاء ثابت در نظر گرفته شده است که تنها توسط ادمین سیستم و از طریق صفحه ادمین قابل تنظیم است.



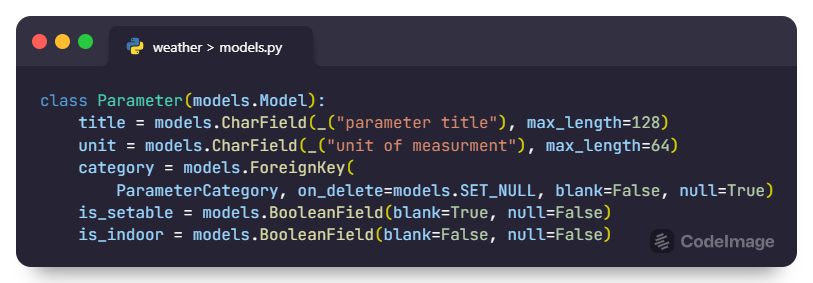
پس از معرفی endpoint ها و view ها در این قسمت به بررسی مدل های مورد استفاده در اپلیکیشن weather خواهیم پرداخت.

**دسته پارامترها (ParameterCategory) :**



این مدل دارای تنها یک خصیصه از نوع رشته به نام title است که به نام دسته پارامتر تعریف شده در سیستم اشاره دارد. سطر های این جدول در طول زمان ثابت اند و تنها ادمین سیستم قابلیت اضافه و کم کردن یا تغییر دسته پارامتر‌ها را دارد. در سیستم فعلی پارامتر های معمول مانند دما، رطوبت با دسته basic و پارامتر های پیچیده تر آب و هوایی با دسته non-basic مشخص شده اند در نتیجه در حالت اولیه این مدل تنها دارای دو سطر متناظر با دو دسته خواهد بود اما ممکن است با گسترش سیستم در آینده نیاز به تغییر دسته بندی فعلی را افزایش و کاهش دسته ها داشته باشیم. در این شرایط است که ادمین سیستم به راحتی توانایی انجام این تغییر را بدون عوض کردن کد سیستم خواهد داشت.

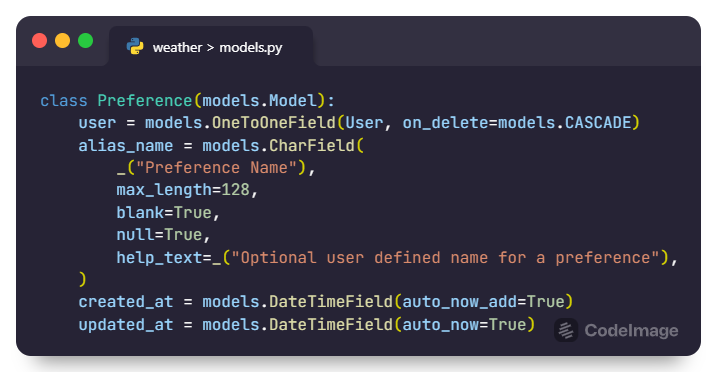
**پارامتر(Parameter) :**



سطرهای این جدول پارامتر های هواشناسی هستند. مواردی مانند دما، رطوبت و فشار از جمله این پارامتر‌ها هستند. هر پارامتر دارای خصیصه های title، unit و category است که به ترتیب به نام پارامتر، واحد و دسته آن اشاره دارند. دو خصیصه دیگر is\_setable و is\_indoor هستند. مورد اول به قابل تنظیم بودن یا نبودن پارامتر اشاره دارد. برای مثال فشار هوا در این سیستم غیرقابل تنظیم در نظر گرفته شده است. مورد دوم اشاره به این دارد که پارامتر مربوط به داخل خانه است یا بیرون. برای مثال برای پارامتر دما ممکن است دو سطر در این پایگاه داده داشته باشیم، یکی برای دمای داخل و دیگری برای دمای بیرون از خانه.

یکی راه های پیاده سازی ای که ممکن بود برای پارامتر های هواشناسی استفاده شود این بود که این پارامتر‌ها را به عنوان خصیصه های ثابتی از یک جدول مانند جدول وضعیت هوا در نظر بگیریم. اما این طراحی مقیاس پذیری بالایی ندارد و در طول زمان ممکن است دچار مشکل شود. مثلا اگر بخواهیم در سیستم تصمیم‌گیری هایی را بر مبنای پارامتر هایی انجام دهیم که در زمان طراحی سیستم به عنوان خصیصه در نظر گرفته نشده اند مجبور بودیم که طراحی جدول پایگاه داده و مدل های سیستم را تغییر دهیم. واضح است که چنین طراحی ای مطلوب ما نیست. بنابراین یک جدول جدا برای پارامتر‌ها طراحی کردیم تا هر زمان که نیاز بود بتوان به آسانی و بدون تغییر کد برنامه تغییرات را روی پارامتر‌ها اعمال کرد.

**ترجیحات کاربر (Preference) :**



مدل ترجیحات این امکان را فراهم کرده که کاربر بتواند شرایط دلخواه خودش را از لحاظ دمای داخل خانه مشخص کند و این ترجیح به صورت اتوماتیک در تنظیم پارامتر دما لحاظ خواهد شد. هر کاربر تنها می‌تواند یک ترجیح فعال داشته باشد بنابراین خصیصه user به صورت OnetoOne تعریف شده است. کاربر از طریق خصیصه alias\_name می‌تواند نام دلخواهی را برای ترجیحش انتخاب کند. دو خصیصه بعدی نیز تاریخ های ساخت و آخرین بروزرسانی را در خودشان ذخیره می‌کنند.

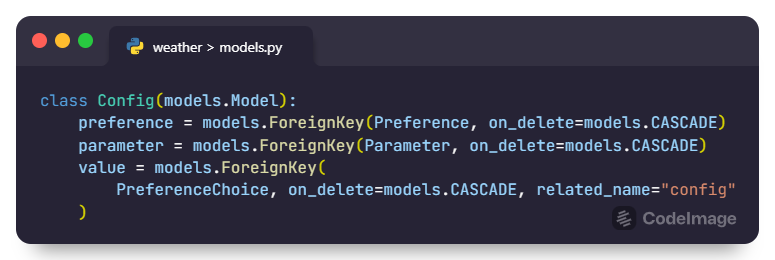
**گزینه های ترجیحات کاربر (PreferenceChoice) :**



سطر های جدول متناظر با این مدل، انتخاب های موجود برای ترجیحات کاربر هستند. دو خصیصه title و impact به ترتیب به نام و تاثیر این انتخاب ها روی دمای نهایی اشاره دارند. تاثیر به صورت درصد تعریف می‌شود. مثلا تاثیر 10 درصد به معنای این است که دمای نهایی بدست آمده برای خانه 10 درصد افزایش می‌یابد. کاربری که ترجیح می‌دهد دمای خانه اش معمولا بالا تر سطح نرمال باشد می‌تواند از این گزینه استفاده کند. تاثیر به صورت پیشفرض صفر در نظر گرفته شده است.

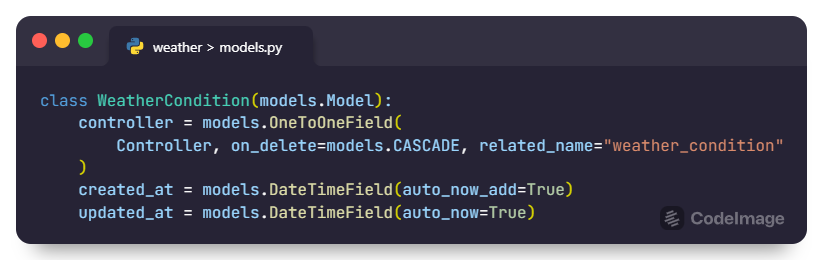
دسترسی به تغییرات روی این تنظیمات به علت حساسیت بالا فقط برای ادمین سیستم وجود دارد. همچنین این طراحی ارائه شده، توانایی تعریف ترجیحات جدید مطابق با اقتضای سیستم، بدون نیاز به تغییر کد و ساختار جداول و مدل ها، در هر زمانی را به ادمین سیستم می‌دهد.

**نتظیم ترجیحات برای کاربر (Config) :**



تنظیم و انتصاب گزینه های ارائه شده در مدل قبل به ترجیحات هرکدام از کاربر ها در این جدول صورت می‌گیرد. این جدول به نوعی ارتباط دهنده تمامی جدول های مربوط به ترجیحات است. هر سطر نشان می‌دهد که برای هرترجیح یک کاربر کدام پارامتر به کدام گزینه مرتبط شده است. تا اینجا فرض شد که ترجیحات تنها به روی دمای داخل عمل می‌کنند اما از این جدول می‌توان برداشت کرد که امکان تاثیر دادن ترجیحات روی پارامتر های دیگر نیز در صورت لزوم وجود دارد. همچنین با توجه به ارتباط چند به یک دو جدول Preference و Config این مورد مشخص می‌شود که می‌توان در یک ترجیح برای پارامتر های مختلفی گزینه های مختلفی انتخاب کرد.

**وضعیت آب و هوا (WeatherCondition) :**



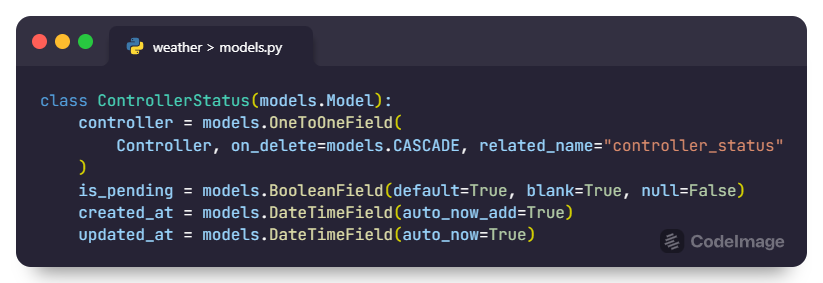
این مدل وظیفه ذخیره سازی اطلاعات هواشناسی دریافتی از سرور OpenWeather به عهده دارد. جزئیات پارامتر های هواشناسی در این جدول ذخیره نمی‌شوند و جدول فعلی پدر آن ها محسوب می‌شود یعنی در آن ها یک کلید خارجی به سمت این جدول وجود دارد.

**جزئیات وضعیت آب و هوا (ParameterValue) :**



جزئیات پارامتر های هواشناسی که در جدول قبلی اشاره شد، در جدول متناظر با این مدل ذخیره می‌شوند. خصیصه weather\_condition که یک کلید خارجی است مشخص می‌کند سطر فعلی مربوط به کدام وضعیت آب و هوا است. خصیصه parameter هم نشان دهنده پارامتری ای است که مقدارش در خصیصه value ذخیره می‌شود.

**وضعیت کنترلر (ControllerStatus) :**



این مدل شباهت زیادی با مدل WeatherCondition دارد با این تفاوت که سطر های جدول متناظر با این مدل از لحاظ منطقی به وضعیت پارامتر های ارسال شده برای کنترلر اشاره دارند. جزئیات پارامتر ها در این جدول ذخیره نمی‌شوند. این مدل یک پدر برای مدل بعدی است که جزئیات وضعیت هر کنترلر که شامل مقدار پارامتر ها است را ذخیره می‌کند.

دیگر تفاوت این مدل با مدل وضعیت آب و هوا خصیصه is\_pending است که به منظور اطمینان از رسیدن اطلاعات به کنترلر پیاده‌سازی شده است. عملکرد آن به این صورت است که پس از ارسال اطلاعات مقدار این خصیصه False می‌شود و پس از رسیدن به مقصد، کنترلر یک پیام Ack برمیگرداند که این خصیصه را True می‌کند.

**جزئیات وضعیت کنترلر (StatusValue) :**



این مدل از لحاظ ساختاری شباهت زیادی به مدل ParameterValue که در قبل توضیح داده شد دارد. اما از لحاظ منطقی در اینجا سطر های جدول جزئیات وضعیت تنظیم شده روی کنترلر های خانگی را نشان می‌دهند.

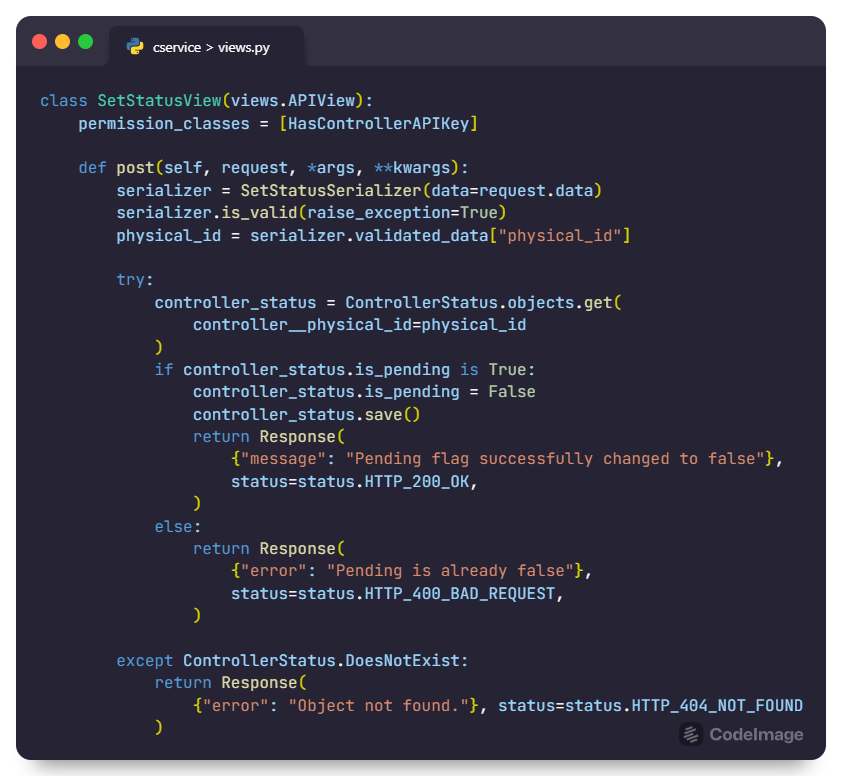
### 3-4-4- اپلیکیشن cservice

در این اپلیکیشن سرویس های مربوط به کنترلر که نیاز به عملیات های بیشتری نسبت به CRUD ساده داشتند آورده شده است. لیست زیر endpoint های مورد نظر را نشان می‌دهد.

|  |  |  |  |
| --- | --- | --- | --- |
| name | view | operation | Endpoint |
| weather\_condition\_view | WeatherConditionView | دریافت وضعیت کنترلر از سرور توسط کنترلر خانگی | weather-condition/ |
| set\_status\_view | SetStatusView | دریافت پیام Ack از طرف کنترلر مبنی بر تنظیم موفقیت آمیز پارامتر های هواشناسی | set-status/ |
| register\_controller\_view | RegisterControllerView | دریافت پیام ثبت اولیه کنترلر | register-controller/ |

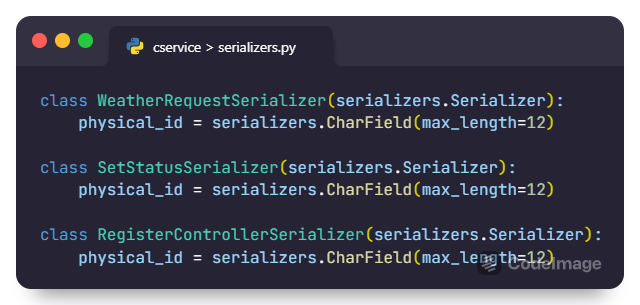
توضیحات درباره نحوه پیاده‌سازی دو endpoint مهم یعنی weather-condition/ و register-controller/ به ترتیب در بخش 3-2-2 آورده شده است.

قطعه کد زیر نحوه ارائه سرویس برای دریافت پیام Ack از طرف کنترلر را توصیف می‌کند. همانند قبل این کد که منطق برنامه را کنترل می‌کند، در قالب یک کلاس، در فایل views.py پیاده سازی شده است.



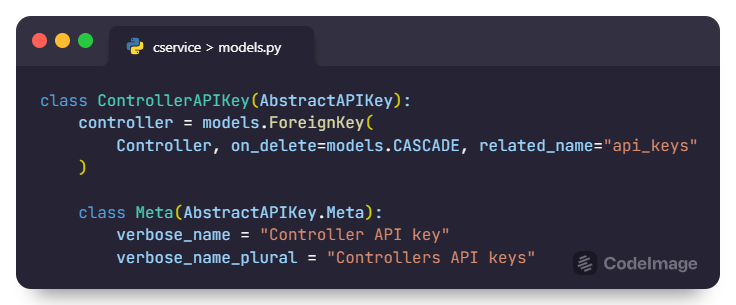
پس از بررسی شدن معتبر بودن پیام از لحاظ احراز هویت و محتوا برنامه سعی می‌کند تا کنترلر مربوط به شناسه فیزیکی موجود در پیام را پیدا کند. در صورتی که شئ مورد نظر وجود نداشته باشد، یک پیام با کد 404 مبنی بر اینکه شئ مورد نظر در سرور وجود ندارد به عنوان پاسخ برگردانده می‌شود. اما در غیر این صورت چک می‌شود که اگر خصیصه is\_pending برابر با False است تغییری در آن ایجاد نمی‌کند و پیام با کد 400 را برمی‌گرداند. اگر همه چیز مطابق میل بود باید is\_pending که قبلا True بوده است، حالا False شود و پیام موفقیت آمیز با کد 200 به عنوان پاسخ برگردانده شود.

ساختار serializer ها نیز برای تمام سرویس های این اپلیکیشن به صورت زیر تعریف شده است.



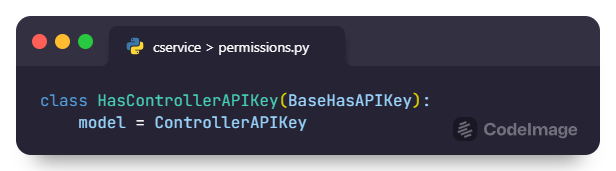
کد بالا نشان می‌دهد که در هر سه سرویس این اپلیکیشن تنها ورودی مورد نیاز به سرویس ها شناسه فیزیکی کنترلر ها است. البته برای اعطای دسترسی به درخواست ها نیاز به کلید احراز هویت نیز است که این کلید در بدنه درخواست وجود ندارد، بلکه در header پیام HTTP در قسمت Authorization قرار گرفته است.

تنها مدل تعریف شده در cservise مدلی است که مربوط به اعطای کلید های احراز هویت به کنترلر است.



این مدل از AbstractAPIKey ارث بری کرده است و با استفاده از تعیین کلید خارجی controller ارتباط مدل کلید ها به مدل controller برقرار شده است.

در cservice یک فایل جدید به نام permissions.py وجود دارد که وظیفه اش تعریف سطح دسترسی کنترلر ها به سرویس ها است.



در این کلاس از BaseHasAPIKey ارث بری کرده ایم و فقط مدل مذکور را به عنوان مقدار متغیر model به این کلاس معرفی می‌کنیم. حال HasControllerAPIKey به عنوان یک سطح دسترسی در سیستم تعریف شده است و امکان بهره‌گیری در view ها وجود دارد، همانطور که قبلا از آن استفاده کرده ایم.

## 3-5- سرویس‌های ثانویه

### 3-5-1- احراز هویت[[11]](#footnote-11)

### 3-5-2- مستندات[[12]](#footnote-12)

## 3-6- امنیت[[13]](#footnote-13)

# فصل چهارم نتیجه‌گیری

در این پروژه، به بررسی رابطه بین سیگنال‌های مغزی و تصویرسازی حرکتی با استفاده از رابط‌های مغز-کامپیوتر (BCI) پرداخته شد. در فصل اول، با مرور کلیات موضوع، انواع سیگنال‌های مغزی و روش‌های تهاجمی و غیر تهاجمی برای ضبط این سیگنال‌ها معرفی شد.

در فصل دوم ابتدا به معرفی دیتاست مورد استفاده، نوع داده‌ها و پیش‌پردازش آنها پرداخته شد. در ادامه الگوریتم‌های یادگیری ماشین مختلفی برای آموزش مدل استفاده شد. الگوریتم‌هایی مانند شبکه‌های عصبی، LGBMClassifier، MIN2Net و CNN های یک بعدی و دو بعدی به کارگرفته شدند. از میان این الگوریتم‌ها، مدل توسعه یافته توسط الگوریتم CNN\_2D از بهترین دقت برخوردار بود. با توجه به دقت بالا حدود 65% ، این مدل برای طبقه‌بندی سیگنال‌‌های EEG استفاده خواهد شد. در ادامه فصل به بررسی محیط‌‌های شبیه‌سازی پرداخته شد که از بین ابزار‌های موجود، پلتفرم OpenViBE به دلیل مواردی همچون متن باز بودن، ماژولار بودن، پردازش لحظه‌ای و غیره برای طراحی روند آزمایش‌های مرتبط با BCI استفاده خواهد شد.

در پروژه فعلی مکانیزم بازخورد به صورت دستی از کاربر دریافت می‌شود. در این زمینه امکان حذف خطای انسانی در نتیجه بهبود عملکرد مدل از طریق تشخیص سیگنال ERP بعد از رخداد خطا، فراهم است و در ادامه اضافه کردن مدل تشخیص ERP به شبیه‌سازی، روند پیشرفت توسعه مدل‌های تصویر‌سازی حرکتی را سرعت می‌بخشد.

# پیوست‌ها

[1] [MI\_Models.ipynb](https://iutbox.iut.ac.ir/index.php/s/g7P7d7YMMRddRNG)

[2] [MIN2Net.py](https://github.com/IoBT-VISTEC/MIN2Net/blob/main/min2net/model/MIN2Net.py)

[3] [utils.py](https://github.com/IoBT-VISTEC/MIN2Net/blob/main/min2net/utils.py)

[4] [loss.py](https://github.com/IoBT-VISTEC/MIN2Net/blob/main/min2net/loss.py)

[5] [BCICIV\_2b\_gdf](https://www.bbci.de/competition/download/competition_iv/BCICIV_2b_gdf.zip) ([description](https://www.bbci.de/competition/iv/desc_2b.pdf))

# مراجع

1. Web Service [↑](#footnote-ref-1)
2. Stateless [↑](#footnote-ref-2)
3. Microservice [↑](#footnote-ref-3)
4. Resources [↑](#footnote-ref-4)
5. Uniform Resource Identifiers [↑](#footnote-ref-5)
6. Home Controller [↑](#footnote-ref-6)
7. Django [↑](#footnote-ref-7)
8. Agile methods [↑](#footnote-ref-8)
9. Entity Relationship Diagram [↑](#footnote-ref-9)
10. Repository [↑](#footnote-ref-10)
11. Authentication [↑](#footnote-ref-11)
12. Documentation [↑](#footnote-ref-12)
13. Security [↑](#footnote-ref-13)