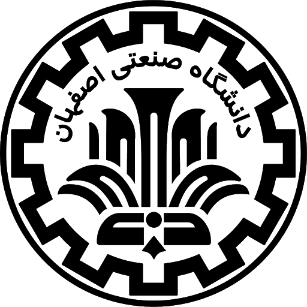
﷽



**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. همچنین، از آقای زارع نیز به خاطر راهنمایی های مستمرشان در طول انجام این پروژه صمیمانه سپاسگزاری می‌کنم.

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](#_Toc156149245)

[1-1- خانه هوشمند 2](#_Toc156149246)

[1-2- داده های هواشناسی 3](#_Toc156149247)

[1-2-1- ذخیره سازی داده‌ها 3](#_Toc156149248)

[1-2-2- خدمات API داده‌ها 3](#_Toc156149249)

[1-3- معماری سیستم 4](#_Toc156149250)

[فصل دوم مفاهیم و پیش‌‍‌زمینه‌ها‌ 5](#_Toc156149251)

[2-1- وب سرویس 5](#_Toc156149252)

[2-2- مقایسه REST و SOAP 6](#_Toc156149253)

[3-2- مفاهیم RESTful API 7](#_Toc156149254)

[2-3-1- منابع 8](#_Toc156149255)

[2-3-2- شناسه های یکنواخت منابع (URIs) 8](#_Toc156149256)

[2-3-3- عملیات های CRUD 9](#_Toc156149257)

[2-4- ابزار های مورد استفاده 10](#_Toc156149258)

[2-4-1- کنترل‌کننده خانه 10](#_Toc156149259)

[2-4-2- سرور 10](#_Toc156149260)

[فصل سوم روش انجام پروژه 12](#_Toc156149261)

[3-1- نیازمندی های کاربر 13](#_Toc156149262)

[3-2- طراحی اجزا و فرایند ها 13](#_Toc156149263)

[3-3- ساختار پروژه 13](#_Toc156149264)

[3-4- پیاده‌سازی 13](#_Toc156149265)

[3-5- سرویس ها 13](#_Toc156149266)

[3-6- بهینه‌سازی 13](#_Toc156149267)

[3-7- امنیت 13](#_Toc156149268)

[فصل چهارم نتیجه‌گیری 14](#_Toc156149269)

[پیوست‌ها 15](#_Toc156149270)

[مراجع 16](#_Toc156149271)

# 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

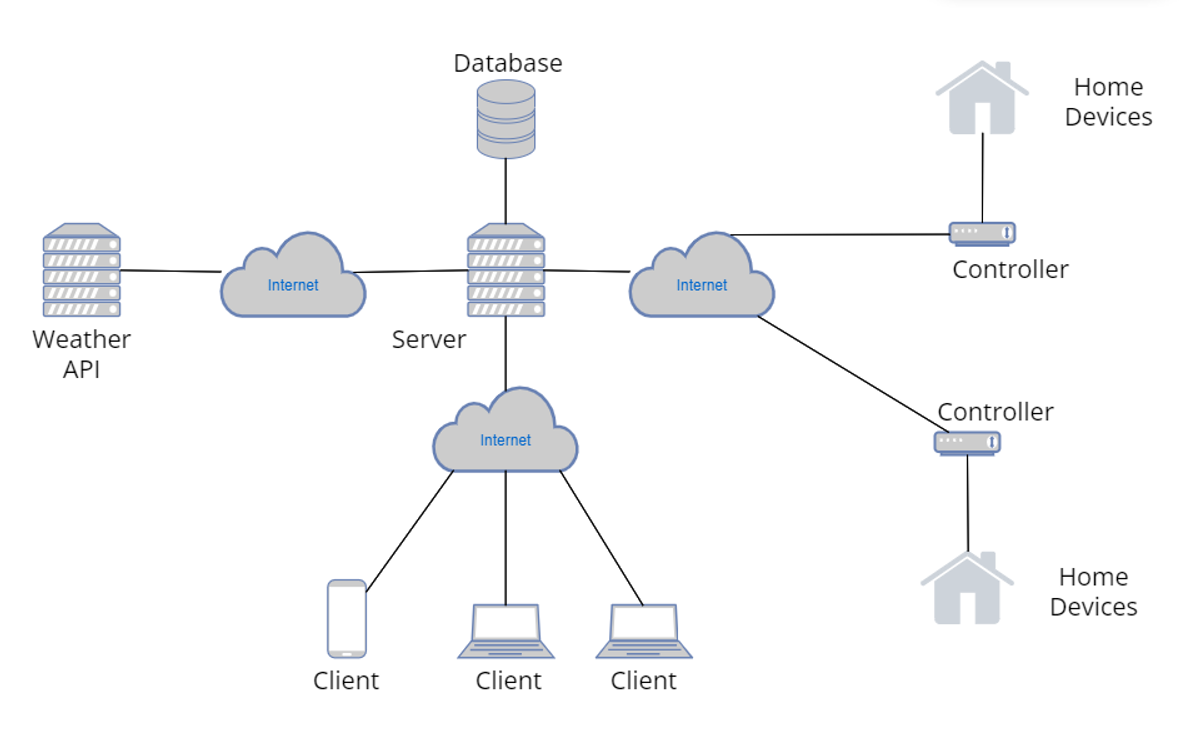
برخی سازمان ها، داده‌های هواشناسی را به صورت دسترسی برای عموم، کسب‌وکارها و توسعه‌دهندگان، به صورت خدمات رابط برنامه‌نویسی (API) ارائه می‌دهند. این API‌ها امکان ادغام سریع اطلاعات هواشناسی را به مختلف برنامه‌ها، وب‌سایت‌ها و خدمات فراهم می‌کنند. توسعه‌دهندگان می‌توانند از این API‌ها برای دریافت شرایط 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

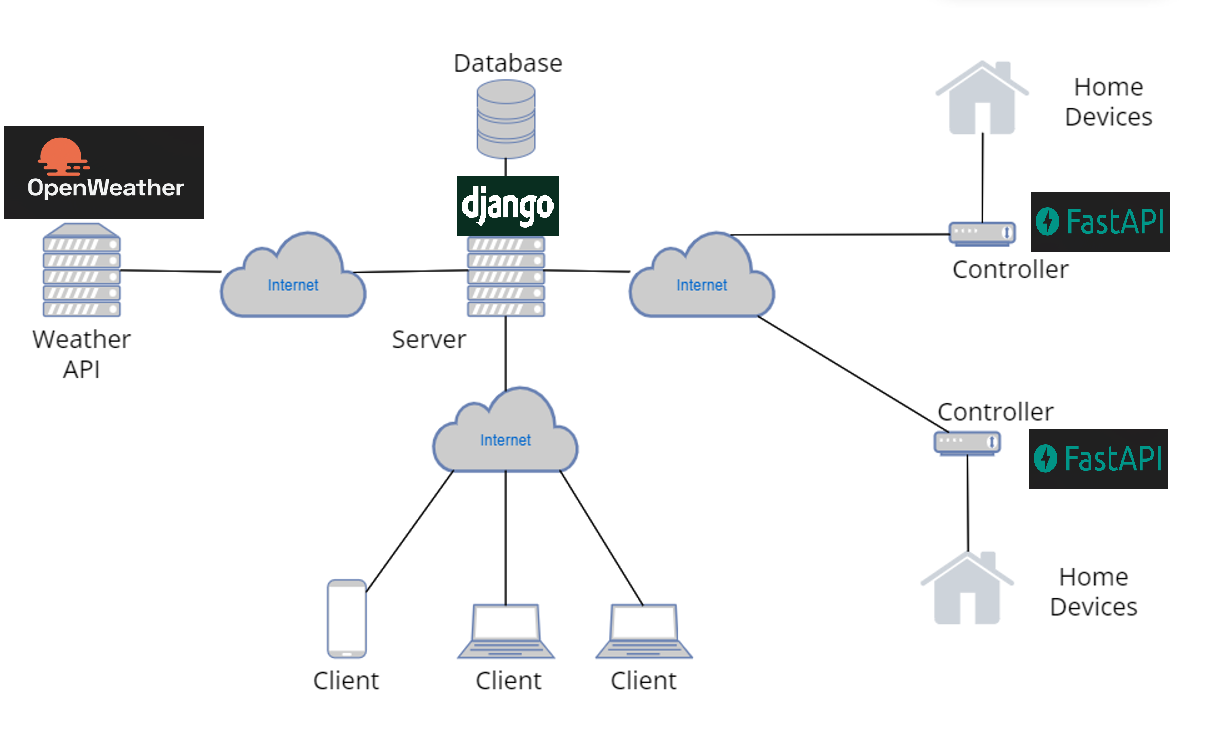
اصلی ترین فریمورک مورد استفاده در توسعه سرور اصلی سیستم جنگو[[7]](#footnote-7) است. ویژگی اصلی ای که باعث انتخاب جنگو برای توسعه این سرور انتخاب شد مقیاس‌پذیری آن است. در یک سیستم خانه هوشمند به مرور زمان حجم ردوبدل شدن اطلاعات می‌تواند بسیار زیاد شود. جنگو با مکانیزم های داخلی بهینه‌سازی و استراتژی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

## 3-2- طراحی اجزا و فرایند‌ها

## 3-3- ساختار پروژه

## 3-4- پیاده‌سازی

## 3-5- سرویس‌ها

## 3-6- بهینه‌سازی

## 3-7- امنیت

# 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)