

دانشگاه فنی و حرفه‌ای

دانشکده فنی مشهد (شهید محمد منتظری)

پروژه کاردانی نرم افزار

**درآمدی بر رایانش ابری**

**و چگونگی پیاده سازی آن**

استاد راهنما:

سید مصطفی چادوردوزان

اساتید مشاور:

محسن کیومرثی و علی امینی

تهیه کننده:

سید مجتبی احمدی

بهار 1393

# چکیده

اهمیت یافتن و ارزش یافتن اطلاعات از یک سو و رشد بسیار بالای حجم و نیاز به بروز رسانی مداوم و در عین حال حفظ یکپارچگی داده از سوی دیگر باعث شده تا ابزار ها و راه حل های نوینی برای پردازش آن ها ایجاد شود. رایانش ابری محصول ادغام تکنولوژی های سنتی رایانش[[1]](#footnote-1) و شبکه ای مانند رایانش مشبک[[2]](#footnote-2)، رایانش توزیع شده[[3]](#footnote-3)، رایانش موازی[[4]](#footnote-4) و تکنولوژی های از این دست می باشد.

در این مقاله سعی بر آن داشته ایم تا چگونگی پیاده سازی یک ابر رایانه ای را توصیف کنیم. در این راستا از میان سه مدل اصلی ابرها، ما فقط به بررسی مدل بستر به عنوان سرویس[[5]](#footnote-5) (PaaS) می پردازیم. هدف این ابر ها فراهم کردن بستری آنلاین برای ایجاد و راه اندازی برنامه های تحت وب[[6]](#footnote-6) می باشد. ایجاد یک برنامه با استفاده از خدمات PaaS نسبت به روش های سنتی برنامه با کار کمتر و در زمان کوتاه تری انجام می شود، دیگر نیازی به نصب و پیکربندی بستر ها و ابزارها و برنامه‌ های گسترش نرم افزار نیست.

بهترین روش درک عملکرد یک ماشین پیاده سازی یک نمونه از آن می باشد. ما در اینجا با ساخت یک نمونه ابر چگونگی عملکرد ابرهای رایانه ای را نشان داده ایم.

فهرست

[مقدمه 1](#_Toc389255282)

[فصل 1: کلیات 3](#_Toc389255283)

[اهداف پروژه 3](#_Toc389255284)

[نمودار مورد استفاده (Use Case) 4](#_Toc389255285)

[تکنولوژی به کار رفته 5](#_Toc389255286)

[فصل 2: پایگاه داده 10](#_Toc389255287)

[انتخاب مدل مناسب 10](#_Toc389255288)

[تقسیم پایگاه داده 11](#_Toc389255289)

[افزونگی تکنیکی 12](#_Toc389255290)

[ساختار نهایی پایگاه داده 13](#_Toc389255291)

[پایگاهی برای هر کاربر 13](#_Toc389255292)

[پایگاهی برای هر برنامه 14](#_Toc389255293)

[جدول های هسته (Core) 14](#_Toc389255294)

[جدول های کاربران (Users) 15](#_Toc389255295)

[جدول های برنامه ها (Tools) 16](#_Toc389255296)

[فصل 3: کنترل‌گر ها 17](#_Toc389255297)

[Gate: try به عنوان validator 17](#_Toc389255298)

[Home: ارتباط 2 جدول از 2 پایگاه 19](#_Toc389255299)

[Create: یک تابع بازگشتی 20](#_Toc389255300)

[Tool: مفسر کد برنامه ها 21](#_Toc389255301)

[Explorer و Editor: یک محیط توسعه 23](#_Toc389255302)

[Tools و Select: دسترسی و مجوز 23](#_Toc389255303)

[فصل 4: نما ها 24](#_Toc389255304)

[gate: دروازه ورودی سایت 24](#_Toc389255305)

[home: داشبورد کنترل برنامه ها 25](#_Toc389255306)

[tools: لیستی از تمام برنامه ها 25](#_Toc389255307)

[create: ساخت یک برنامه جدید 26](#_Toc389255308)

[explorer: یک مرورگر ساده 27](#_Toc389255309)

[editor: یک ویرایشگر ساده 27](#_Toc389255310)

[select: انتخاب یک برنامه جدید 28](#_Toc389255311)

[فصل 5: امنیت 29](#_Toc389255312)

[جعبه امن (زندان chroot) 29](#_Toc389255313)

[محدودیت منابع 30](#_Toc389255314)

[فصل 6: پیشنهاد ها و برنامه های آینده 32](#_Toc389255315)

[پشتیبانی از چند زبان برنامه نویسی 32](#_Toc389255316)

[توسعه برنامه ها به کمک git 33](#_Toc389255317)

[استفاده از ssh برای ارتباط با سورس کد 33](#_Toc389255318)

[ساخت یک زیردامنه برای هر برنامه 34](#_Toc389255319)

[سازگاری با مجوز های مختلف 34](#_Toc389255320)

[اعمال محدودیت پهنای باند برای هر برنامه 34](#_Toc389255321)

[پیوست: کد برنامه 35](#_Toc389255322)

[پیوست: آشنایی با رایانش ابری 68](#_Toc389255323)

[رایانش ابری چیست 68](#_Toc389255324)

[تعریف: 68](#_Toc389255325)

[مدل سرویس: 69](#_Toc389255326)

[مدل آرایش و گسترش: 70](#_Toc389255327)

[امور مربوط به رایانش ابری 71](#_Toc389255328)

[امنیت 71](#_Toc389255329)

[حریم خصوصی 72](#_Toc389255330)

[قابلیت اطمینان 72](#_Toc389255331)

[مسائل حقوقی 72](#_Toc389255332)

[استاندارد باز 73](#_Toc389255333)

[سازگاری 73](#_Toc389255334)

[آزادی 73](#_Toc389255335)

[امکان دوام بلند مدت 74](#_Toc389255336)

[راه حل 74](#_Toc389255337)

[پیوست: آشنایی با معماری MVC 75](#_Toc389255338)

[لایه Model 75](#_Toc389255339)

[لایه View 76](#_Toc389255340)

[لایه Controller 77](#_Toc389255341)

[پیوست: آشنایی با تکنیک ORM 78](#_Toc389255342)

[كاهش زمان تحويل پروژه 79](#_Toc389255343)

[كدي با طراحي بهتر 80](#_Toc389255344)

[نياز به تخصص کم تر 80](#_Toc389255345)

[كاهش زمان آزمايش 81](#_Toc389255346)

[پیوست: آشنایی معماری REST 82](#_Toc389255347)

# فهرست تصاویر

[تصویر 1: نمودار حالت استفاده 4](#_Toc389117080)

[تصویر 2: انتخاب مدل پایگاه داده 10](#_Toc389117081)

[تصویر 3: نمایه کلی پایگاه داده 11](#_Toc389117082)

[تصویر 4: انتخاب مدل جدول بندی 12](#_Toc389117083)

[تصویر 5: ساختار نهایی پایگاه داده 13](#_Toc389117084)

[تصویر 6 صفحه gate 24](#_Toc389117085)

[تصویر 7 صفحه home 25](#_Toc389117086)

[تصویر 8 صفحه tools 25](#_Toc389117087)

[تصویر 9 صفحه create 26](#_Toc389117088)

[تصویر 10 صفحه explorer 27](#_Toc389117089)

[تصویر 11 صفحه editor 27](#_Toc389117090)

[تصویر 12 صفحه select 28](#_Toc389117091)

# مقدمه

با پیشرفت فناوری اطلاعات نیاز به انجام کارهای محاسباتی در همه جا و همه زمان به وجود آمده است. همچنین نیاز به این هست که افراد بتوانند کارهای محاسباتی سنگین خود را بدون داشتن سخت‌افزارها و نرم‌افزارهای گران، از طریق خدماتی انجام دهند. رایانش ابری آخرین پاسخ فناوری به این نیازها بوده است. عموماً مصرف کننده‌های رایانش ابری مالک زیر ساخت فیزیکی ابر نیستند، بلکه برای اجتناب از هزینه سرمایه‌ای، تنها آن را از عرضه کنندگان اجاره می‌کنند. آنها منابع را در قالب سرویس مصرف می‌کنند و تنها بهای منابعی که به کار می‌برند را می‌پردازند.

رایانش ابری در نمودار دوره محبوبت گروه گارتنر در راس دوره محبوبت قرار دارد، در این مقطع رایانش ابری در مرکز توجهات است اما هنوز کاملاً پتانسیل‌های خود را بلفعل نکرده است. طبق پیش بینی‌های گارتنر طی سه تا چهار سال آینده رایانش ابری پتانسیل واقعی خود را نمایان می‌کند. رایانش ابری بطور چشمگیری موانع ورود به تجارت نرم‌افزاری را کاهش می‌دهد و برای شرکت‌ها روش‌های جدید کسب سود را می نمایاند. ارائه دهندگان خدمات ابر از طریق تسهیم، بهبود دادن و سرمایه‌گذاری بیشتر در نرم‌افزار و سخت‌افزار به سود دست می یابند. یکبار نصب نرم‌افزار می‌تواند نیازهای کاربران متعددی را پوشش دهد. در قرن ۲۱ شاهد افزایش تمایل استفاده از وسایل قابل حمل سبک برای دسترسی به خدمات اینترنت بجای کامپیوترهای شخصی هستیم. از آنجایی که چنین وسایلی، امکانات پردازشی قوی ندارند (بعبارتی علاقه‌ای به داشتن چنین امکاناتی ندارند)، پس چه کسی قدرت پردازشی را تامین خواهد کرد؟ پاسخ به این سوال در رایانش ابر نهفته است.

در این پایان نامه به بررسی چگونگی ساخت یک ابر از نوع بستر به عنوان سرویس می پردازیم. برای مطالعه این پایان نامه، خواننده لازم است با مفاهیم رایانش ابری و انواع آن، معماری های MVC و REST و تکنیک ORM آشنا باشد. توضیح مختصری در مورد هر یک از این مفاهیم در بخش پیوست ها آورده شده. از خواننده تقاضا می کنیم، در صورت آشنا نبود با این مفاهیم، ابتدا به بخش پیوست ها مراجعه کند. تا درک مفاهیم و اصطلاحات به کار رفته در این پایان نامه دچار سر درگمی نشوند. علاوه بر این موارد آشنایی با زبان python و چارچوب flask و sql و همچنین مبانی طراحی سایت با این زبان لازم است.

این پایان نامه در 6 فصل تنظیم شده. که در فصل 1 به بیان کلیات پروژه مانند اهداف و تکنولژی ها پرداخته ایم. فصل 2 به تحلیل پایگاه داده اختصاص دارد. از آن جا که این پروژه از معماری MVC پیروی می کند، هر لایه این معماری را در یک فصل توضیح داده ایم. مدل ها را در همان فصل پایگاه داده تشریح می کنیم. و در دو فصل 3 و 4 به بررسی کنترل گرها و نما ها می پردازیم. در فصل 5 به نکات امنیتی به کار رفته در ابر می پردازیم و در فصل 6 پیشنهاداتی را ارئه کرده ایم که می توان در آینده به این پروژه اضافه کرد. کد های کامل برنامه نیر به صورت پیوست در انتهای این پایان نامه ضمیمه شده اند.

# فصل 1: کلیات

در این فصل نگاهی کلی به پروژه می اندازیم و اهداف کلی خود را از پروژه مشخص می کنیم و در مورد چگونگی پیاده سازی اهداف صحبت خواهیم کرد.

## اهداف پروژه

برای آشنایی کامل با طرز عملکرد ابر PaaS بهترین روش تولید یک نمونه از آن می باشد. تا با مراحل ساخت آن خود را درگیر کنیم، مسأله را خود تحلیل کنیم، الگریتم های آن را طراحی کنیم و در صورت برخورد با مشکلات احتمالی، آن ها را حل کنیم.

هدف نهایی پیاده سازی یک نمونه ابر PaaS بسیار ساده می باشد. حداقل موارد لازم برای پیاده سازی ابر به شرح زیر است:

**محیط توسعه**[[7]](#footnote-7): کد های وارد شده توسط کاربران در اینجا کمپیل می شود.

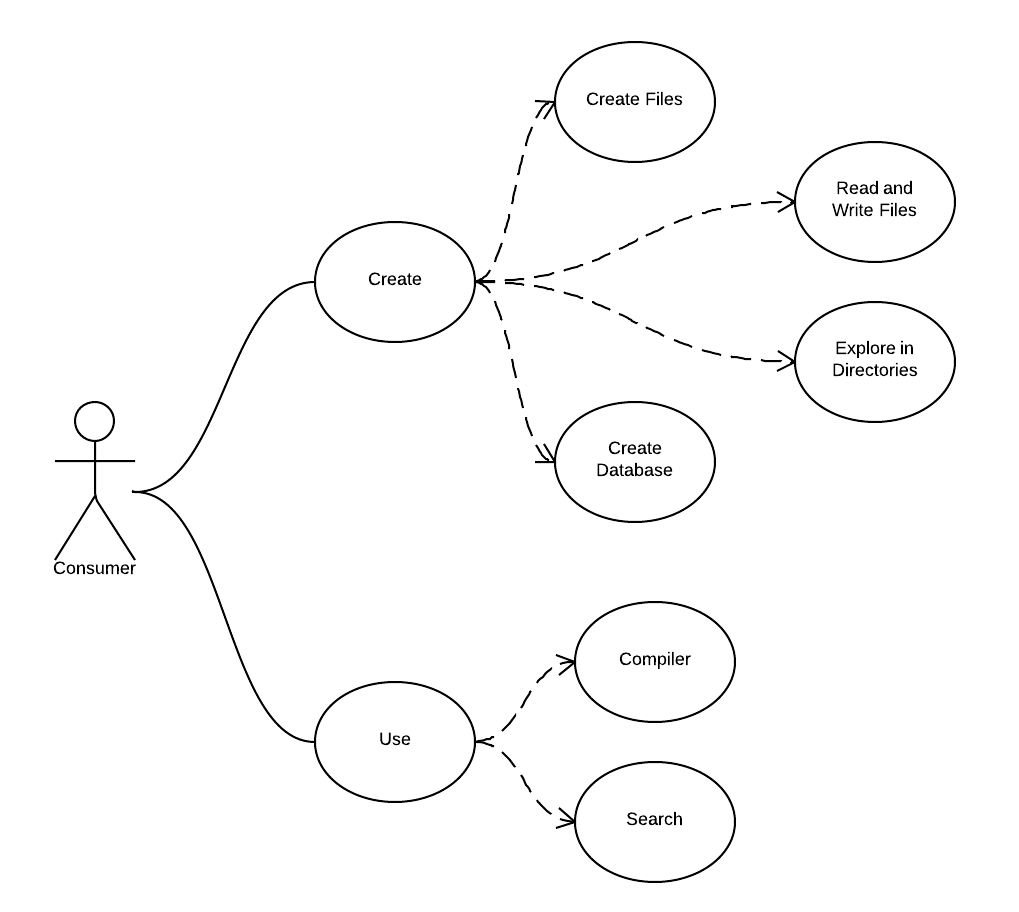
**رابط**[[8]](#footnote-8): وظیفه بر قراری ارتباط میان کاربر و محیط توسعه را به عهده دارد.

**اشکال‌یاب**[[9]](#footnote-9): در صورت وجود مشکل در برنامه آن را به کاربر اطلاع می دهد.

**انتشار**: برای آن که دیگر کاربران به توانند از برنامه استفاده کنند باید مکانی برای انتشار آن ها در نظر بگیریم.

## نمودار مورد استفاده (Use Case)

با برسی اهداف و تجزیه و تحلیل عملکرد ابر PaaS به نمودار زیر می رسیم. که خواسته های کاربر از برنامه را نمایش می دهد.



تصویر 1: نمودار حالت استفاده

دو امکان اصلی در اختیار کاربران قرار می گیرد. یکی ساخت برنامه و دیگری استفاده از برنامه ای که دیگر کاربران ساخته اند. هر کدام از این دو امکان خود از بخش های تشکیل شده اند. برای آن که کاربر به تواند برنامه ایجاد کند، به امکاناتی مانند ساخت فایل، ساخت پایگاه داده و امکان خواندن و نوشتن بروی فایل ها. که انجام این کارها نیازمند امکان کاوش[[10]](#footnote-10) نیز می باشد.

## تکنولوژی به کار رفته

|  |  |  |
| --- | --- | --- |
|  | برنامه | نسخه[[11]](#footnote-11) |
| زبان برنامه نویسی | Python | 2.7.3 |
| چارچوب[[12]](#footnote-12) | Flask | 0.8 |
| پایگاه داده | SQLite | 2.6.0 |
| وب سرور | Werkzeug | 0.8.3 |
| سیستم عامل | Debian | 7.0.0 |
| مجازی ساز | VirtualBox | 4.1.18 |

در ادامه به برسی دلیل انتخاب هر کدام از این برنامه ها می پردازیم.



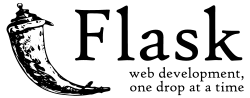
**زبان برنامه نویسی:** در حال حاضر سه زبان برنامه نویسی PHP، Ruby و Python مطرح ترین زبان ها در زمینه طراحی وب می باشند.

کار با PHP به دلیل وجود توابع بسیار زیاد آن راحت و آسان است. اما همین تعداد زیاد توابع باعث سنگین شدن مفسر[[13]](#footnote-13) این زبان شده است. و سرعت پاسخ گویی آن نسبت به دو رقیب خود کاهش قابل توجه ای دارد. هرچند در سایت Facebook با استفاده از تکنیکی خاص (HipHop) کد های برنامه را به زبان ماشین کمپایل کرده اند و مشگل سرعت به گونه ای حل شده است.

از آن طرف زبان Ruby با تمرکز بر روی سرعت خود، نام خود را به عنوان سریع ترین زبان ثبت کرده است. تعداد ماژول ها و کتابخانه های آن، نسبت به دو زبان دیگر کمتر بوده، و این امر کار با این زبان را کمی دشوار می سازد. بهترین مثال برای نمایش قدرت این زبان Twitter است.

زبان Python را می توانیم ما بین دو زبان دیگر دانست. سرعت بالایی دارد اما سریع ترین نیست و کار با آن راحت است اما آسان ترین نیست. سرعت بالا در محاسبات در کنار آسانی طراحی برنامه باعث محبوبیت این زبان شده است. نمونه ای عالی از این زبان Youtube است که کاملا با این زبان پیاده سازی شده است و در بخش های از Google نیز از این زبان استفاده کرده اند.

با بررسی انجام شده زبان Python برای انجام پروژه انتخاب شده است. که علاوه بر داشتن سرعت و قدرت کافی برای انجام این پروژه، از سینتکس[[14]](#footnote-14) روان تری بر خوردار است. که امیدواریم موجب انتقال بهتر مفاهیم شود.



**چارچوب[[15]](#footnote-15):** تا به حال چارچوب های زیادی برای Python نوشته شده محبوب ترین آن ها Django و Flask می باشند.

Django از معماری MVC پیروی می کند و هدف اصلی آن ساخت آسان سایت‌های پیچیده و وابسته به دیتابیس است و بر پایه قابلیت استفاده مجدد[[16]](#footnote-16) و قابل اتصال بودن[[17]](#footnote-17) اجزای مختلف، توسعه سریع و اصل خودت تکرار نکن[[18]](#footnote-18) (DRY) طراحی شده است. برترین مزیت این چارچوب نسبت به سایر چارچوب ها بخش ORM آن است که کار با دیتابیس[[19]](#footnote-19) را بسیار ساده می کند. و نقطه ضعف آن در این است که توسعه برنامه های نوشته شده با Django چندان راحت نیست.

Flask چارچوبی بسیار سبک است که از معماری REST پیوری می کند. اغلب Flask را Microframework صدا می زنند. زیرا هسته برنامه را ساده نگه می دارد. امکاناتی مانند کار با دیتابیس، اعتبار سنجی فرم و از این قبیل امکانات که معمولا در اکثر چارچوب های وب قرار دارد، در Flask وجود ندارد. تمامی این امکانات به صورت کامپونت های جداگانه موجود است. حتی پیاده سازی مدل MVC نیز امکان پذیر است.



**پایگاه داده:** در پروژه هایی مانند این پروژه معمولا از MySQL استفاده می شود. ما ابتدا مختصر توضیحی در مورد SQLite می دهیم. و سپس به دلیل انتخاب آن می پردازیم.

بر خلاف پایگاه‌های داده مستقل نظیر MySQL یا SQL Server، SQLite هیچ فرایند مستقلی را روی پردازنده اجرا نمی‌کند، بلکه کتابخانه اس‌کیوال لایت به برنامه اصلی پیوند می‌شود و با اجرای برنامه اصلی، SQLiteهم اجرا می‌گردد. برنامه اصلی برای دسترسی به اطلاعات پایگاه داده یا تغییر آن‌ها از رویه‌های موجود در کتابخانه بهره می‌برد و این طراحی باعث کاهش تاخیر در دسترسی به اطلاعات (در مقایسه با استفاده از پایگاه داده‌های مستقل) می‌گردد. به منظور حفظ یکپارچگی اطلاعات ذخیره شده در پرونده، این پرونده در زمان نوشتن قفل می‌شود.

با وجود برتری در سرعت، SQLite یک مشکل بزرگ دارد. هنگام حجم اطلاعات بالا می رود کارایی پایگاه داده کاهش می یابد. ما با استفاده از تکنیکی خاص این مشکل را حل کرده ایم که در فصل دوم به توضیح کامل آن می پردازیم.

D:\Documents\Collage\Project\Source\Images\Tech\121px-Nginx_logo.svg.png

**وب سرور:** چارچوب Flask یک وب سرور[[20]](#footnote-20) کوچک به نام Werkzurg همراه خود نصب می کند. که تمام نیازهای ما را در حد این پروژه برآورده می سازد. هدف از طراحی این وب سرور سهولت طراحی برنامه بوده و برای پروژه های بزرگ گزینه مناسبی نیست. در صورت لذوم می توان از وب سرور Nginx استفاده کرد، که سازگاری کاملی با Python و Flask دارد.



**سیستم عامل:** بسیاری از موسسات و ادارات کشورهای مختلف از دبیان استفاده می‌کنند که فهرست آن‌ها در این نشانی: www.debian.org/users قابل مشاهده‌است. پست‌های الکترونیکی که به فهرست پستی دبیان ارسال می‌شود در کوتاه ترین زمان پاسخ داده می‌شوند. برای مشاهده و عضویت در این فهرست‌ها می‌توانید به این نشانی: www.debian.org/MailingLists مراجعه کنید.

دبیان دارای یکی از بهترین سیستم مدیریت بسته‌[[21]](#footnote-21)های نرم‌افزاری است، این قابلیت به تنهایی برتری دبیان را در برابر دیگر توزیع‌ها اثبات می‌کند. با این ابزار تنومند که APT نام دارد مشکل وابستگی ‌های بسته‌های نرم‌افزاری که در توزیع‌های مبتنی بر ردهت (RPM) دیده می‌شود کاملاً حل شده‌است. با وجود APT انجام ارتقاهای نرم‌افزاری بسیار تسهیل یافته به‌طوری که با یک فرمان می‌توان سیستم‌عامل را به نسخه جدیدتر ارتقا داد. دبیان مفهوم یک بار نصب برای همیشه را عملی می‌کند.

پایداری سیستم‌های دبیان بی‌همتاست، سیستم‌های دبیان ماه‌ها بدون نیاز به راه‌اندازی مجدد کار می‌کنند و دلیل متوقف‌شدن آن قطع برق یا ارتقاهای سخت‌افزاری است.

پیگیری اشکالات نرم‌افزارها و حفره‌های امنیتی دبیان از نشانی bugs.debian.org برای عموم آزاد و قابل دسترس است. دبیان حتی اشکالات موجود را از دید کاربر پنهان نمی‌سارد بلکه با گزارش اشکالات می‌توانید به برطرف‌سازی آن توسط تیم توسعه دبیان سرعت ببخشید.

دبیان دارای امکانات رمزنگاری[[22]](#footnote-22) برای نشت‌های امن[[23]](#footnote-23)، صدها ابزار توسعه نرم‌افزار و زبان‌های برنامه‌نویسی است و در نهایت مسیر دبیان توسط نیاز کاربران خود و فلسفه نرم‌افزارهای آزاد مشخص و هدایت می‌شود.

این موارد برخی از دلایل انتخاب دبیان در بین توزیع‌های گنو/لینوکس بود اما در مقایسه با دیگر سیستم‌عامل‌ها همچون ویندوز می‌توان صدها برتری و مزیت ذکر کرد.



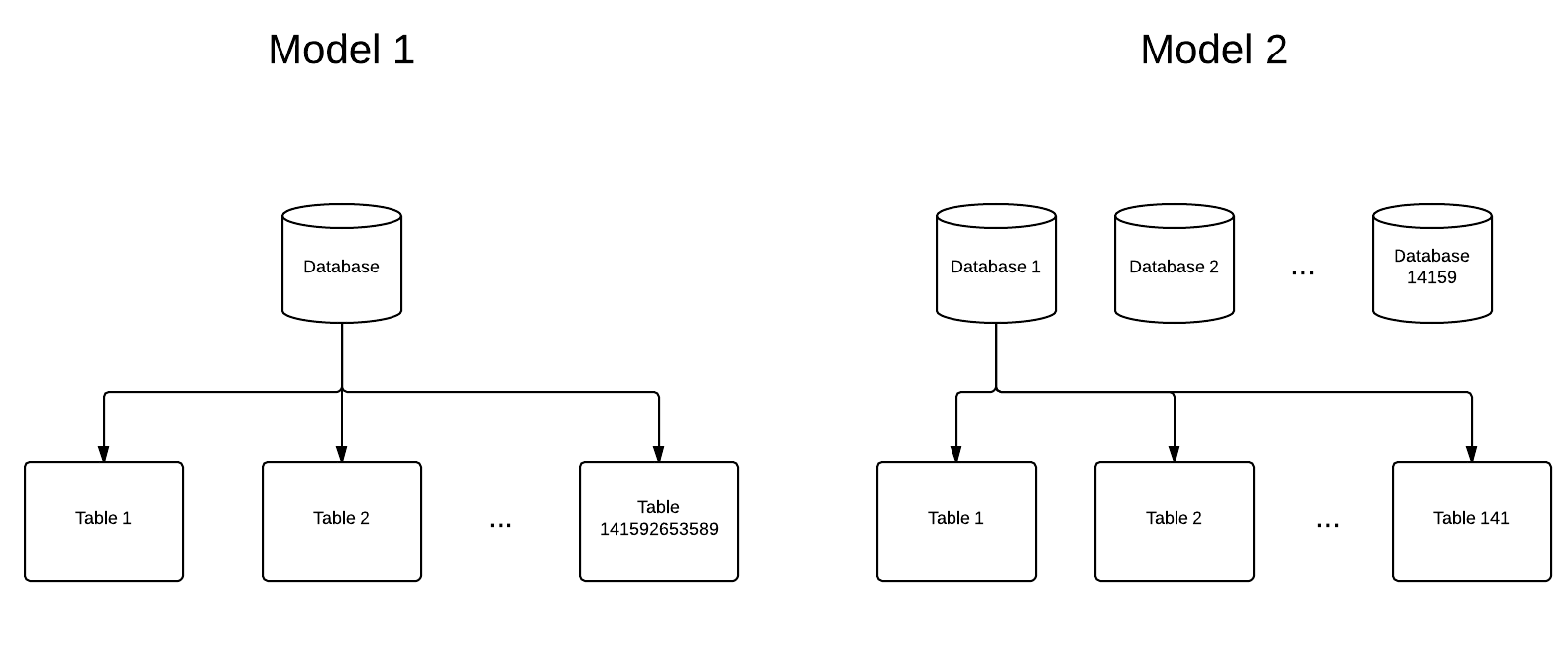
**مجازی ساز:** برای شبیه سازی محیط سرور از نرم افزار مجازی ساز ویرچوال‌باکس استفاده کرده ایم. هر چند وی ام ویر امکانات بیشتری دارد و به خصوص در زمینه شبکه از ویرچوال‌باکس پیشرفته تر عمل می کند، اما ویرچوال‌باکس با محیط ساده باعث سادگی کار با این برنامه شده و حجم کم این نرم افزار باعث سبک شدن این برنامه شده به طوریکه روی هر سیستمی قابل اجرا بوده و در انتها رایگان بودن آن موجب می شود تا به راحتی به توانیم نسخه اصلی آن را نسب کنیم و از آن استفاده کنیم.

# فصل 2: پایگاه داده

در گام اول مدل مناسب برای پایگاه داده خود انتخاب کنیم. با در نظر گرفتن این احتمال که شاید تعداد کاربران ابر به صد ها هزار نفر برسد و هر کدام ممکن است ده ها برنامه را ایجاد کنند. از طرف دیگر همان طور که پیش از این گفته شد یکی از مشکلات SQLite این است که با بالا رفتن حجم پایگاه داده کارایی خود را از دست می دهد.

## انتخاب مدل مناسب

ما در اینجا با روشی پایگاه داده را به قسمت های کوچک تقسیم کرده ایم تا این مشکل را بر طرف سازیم. به دو مدل پیاده سازی پایگاه داده که در زیر آمده است توجه کنید.



تصویر 2: انتخاب مدل پایگاه داده

مدل اول حالت ابتدایی و ساده طراحی پایگاه داده می باشد، که در آن تنها یک پایگاه داده با هزاران ها جدول داریم. در مدل دوم پایگاه داده را به چند قسمت تقسیم می کنیم که با عث می شود تعداد جدول های هر پایگاه داده کاهش می یابد و در نتیجه حجم پایگاه داده نیز کاهش یافته و مشکل حجم در SQLite را می توان به این طریق بر طرف کرد.

## تقسیم پایگاه داده

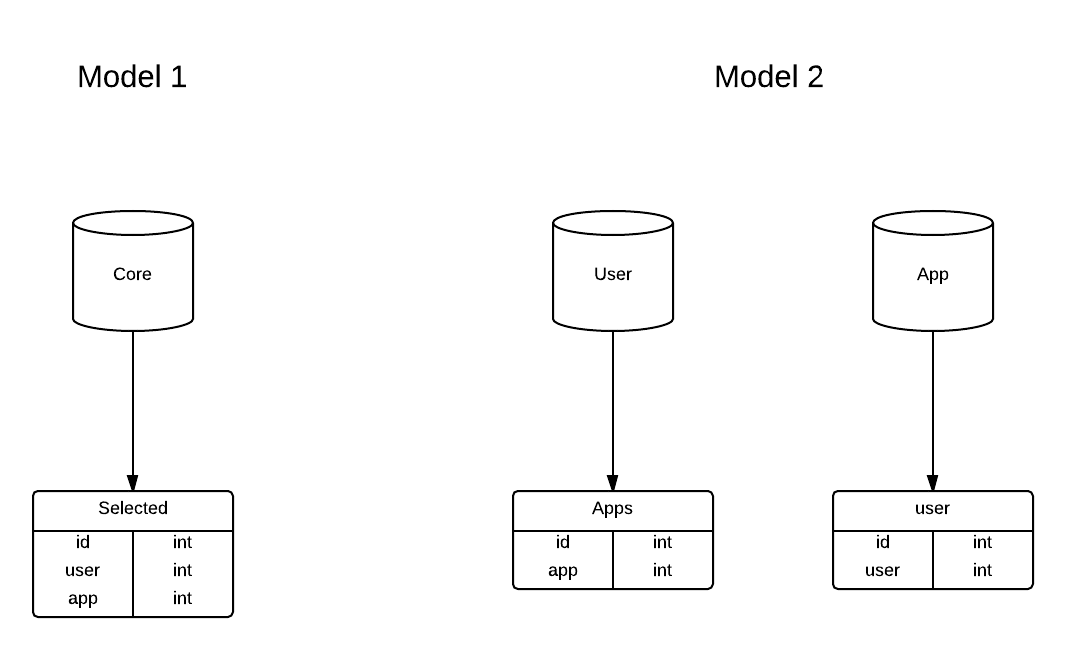
حال که تصمیم بر تقسیم کردن پایگاه داده را داریم، این سوال پیش می آید که به چند قسمت تقسیم کنیم؟



تصویر 3: نمایه کلی پایگاه داده

در این تقسیم بندی ما یک پایگاه داده مرکزی به نام core داریم که هسته برنامه می باشد و اطلاعت کلی در آن قرار می گیرد. سپس برای هر برنامه ای که کاربر ایجاد می کند یک پایگاه داده مجزا تولید می کنیم. سپس سوالی پیش می آید که آیا لازم است برای هر کاربر نیز پایگاه داده ای طراحی کنیم؟ جواب این سوال بستگی به انتخاب مدل پیاده سازی ما دارد. که در بخش بعدی آن را توضیح می دهیم.

## افزونگی تکنیکی[[24]](#footnote-24)

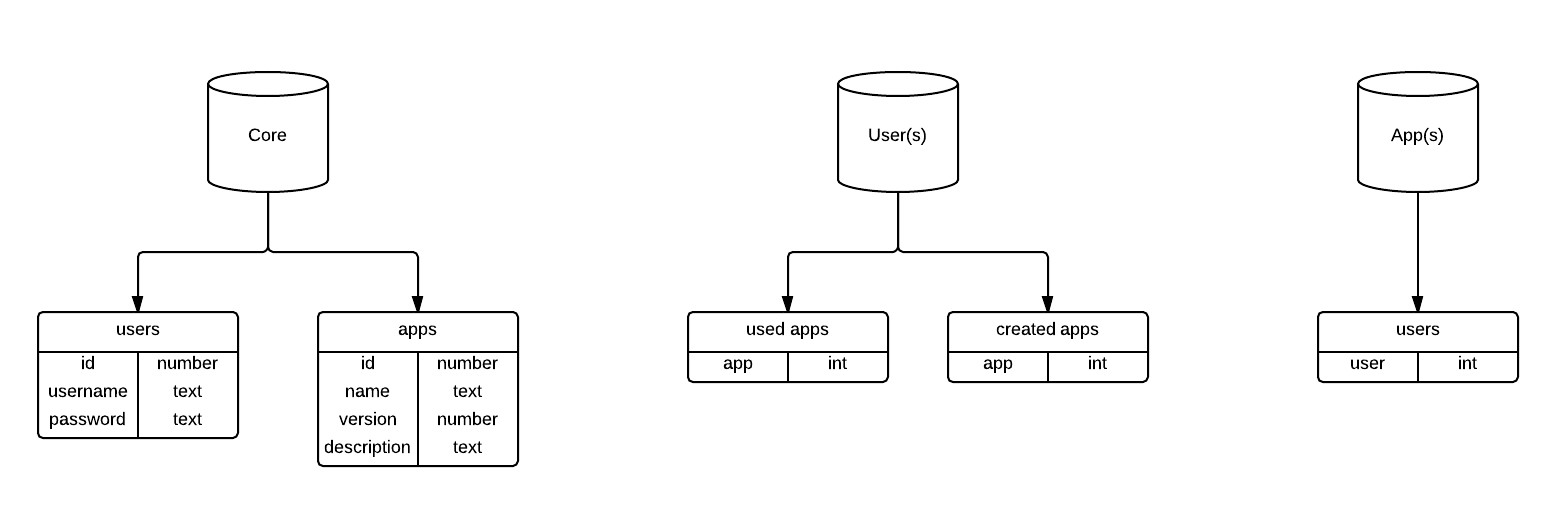
دو مدل زیر را باهم مقایسه می کنیم.

تصویر 4: انتخاب مدل جدول بندی

در مدل اول تنها یک جدول در پایگاه داده core ایجاد می کنیم. از این جدول هم می توان برنامه های انتخاب شده توسط کاربر را تشخیص داد و هم می توان به کاربرانی که در حال استفاده از برنامه خاص هستند دسترسی داشته باشیم. در مدل دوم اطلاعات مربوط به هر کاربر در پایگاه داده اختصاصی آن کاربر ذخیره می شود و با این روش حجم اطلاعات ذخیره شده در هر جدول بسیار کاهش می یابد. از طرافی چون ما اطلاعات را در دوجای مجزا ذخیره می کنیم و از تعداد پایگاه داده بیشتری استفاده می کنیم، که باعث به وجود آمدن یک افزونگی می شود. و در نتیجه حجم کلی اطلاعات بیشتر خواهد شد. با بالا رفتن داده های ذخیره شده سرعت گزارش گیری[[25]](#footnote-25) در مدل اول تساعدی افزایش می یابد. اما در مدل دوم افزایش داده تاثیر چندانی در سرعت گزارش کیری ندارد. بنابر این مدل دوم بسیار بهتر خواهد بود.

## ساختار نهایی پایگاه داده

در انتها با در نظر گرفتن تمام موارد گفته شده، ساختار پایگاه داده به صورت زیر خواهد بود.



تصویر 5: ساختار نهایی پایگاه داده

بعد از تحلیل زمان پیاده سازی است. برای طراحی از یک ابزار ORM (برای آشنایی بیشتر به پیوست ها مراجه کنید) به نام sqlalchemy استفاده می کنیم و جدول ها را به اشیایی در برنامه مپ[[26]](#footnote-26) می کنیم.

## پایگاهی برای هر کاربر

برای آنکه به توانیم برای هر هر کاربر پایگاهی جدا بسازیم، از سیشنی[[27]](#footnote-27) استفاده می کنیم که در هنگام ورود به سایت برای هر کاربر ساخته می شود. نمونه کد برنامه مشابه به شگل زیر خواهد بود:

from sqlalchemy import create\_engine   
  
engine = create\_engine(   
    'sqlite:///' +   
    session['user\_id']  
)

در اینجا از متد create\_engine کلاس sqlalchemy استفاده کرده ایم. با این کد هرگاه کاربری وارد سایت شود، پایگاهی به نام کد کاربری آن فرد ایجاد می شود.

## پایگاهی برای هر برنامه

از آنجایی که پایگاه داده ما sqlite بوده، نیازی به ایجاد پایگاه داده، تعریف کاربر و سایر موارد نیست. برای پایگاه داده هایی مانند MySQL این مراحل الزامی است. پس هر کاربر برای ایجاد پایگاه داده می توانند به روش زیر عمل کنند.

from sqlite3 import connect   
con = connect('test.db')

یا اگر بخواهند از sqlalchemy استفاده کنند:

from sqlalchemy import create\_engine   
engine = create\_engine(

'sqlite:///test.db'

)

## جدول های هسته (Core)

users: نام کاربری و رمز عبور تمام کاربران در اینجا ذخیره می شود و برای ورود به سایت از این جدول استفاده می کنیم. از ذخیره اطلاعاتی شخصی مانند نام، تاریخ تولد و ... به دلیل نداشتن اهمیت برای این پروژه پردهیز کردیم. لیست فیلد ها در زیر آمده است:

|  |  |
| --- | --- |
| نوع فیلد | نام فیلد |
| integer, key, unique, auto | id |
| text | username |
| text | password |

tools: اطلاعات تمام برنامه ها را در خود نگاه می دارد. از این مدل برای جستجو در برنامه ها استفاده می کنیم. لیست فیلد ها در زیر آورده شده:

|  |  |
| --- | --- |
| نوع فیلد | نام فیلد |
| integer, key, unique, auto | id |
| text, unique | name |

## جدول های کاربران (Users)

در اینجا برای هر کاربر یک پایگاه داده جداگانه ایجاد می شود و در هر پیگاه داده جدول هایی به صورت زیر قرار میگیرد.

Selected: این جدول نشان دهنده برنامه هایی می باشد که کاربر از آن ها استفاده می کند. و با یک فیلد نیاز ما را برطرف می کند:

|  |  |
| --- | --- |
| نوع فیلد | نام فیلد |
| integer, key, unique | tool |

Created: این جدول نشان دهنده برنامه هایی می باشد که کاربر از آن ها ایجاد کرده. اینجا نیز تنها یک فیلد کافی بوده:

|  |  |
| --- | --- |
| نوع فیلد | نام فیلد |
| integer, key, unique | tool |

## جدول های برنامه ها (Tools)

بسته به نیاز کاربران می تواند شامل 1 تا بی نهایت جدول باشد. ما هیچ جدولی در نظر نمی گیریم و کاربران در زمان طراحی برنامه خود می توانند جداول را ایجاد کنند.

# فصل 3: کنترل‌گر ها

در این فصل به بررسی کنترل‌گر های معماری MVC (برای آشنایی بیشتر به پیوست ها مراجعه شود) می پردازیم. از آنجایی که تقریبا به ازای هر صفحه سایت یک کنترل گر لازم است، ابتدا نقشه سایت[[28]](#footnote-28) را ترسیم می کنیم.

## Gate: try به عنوان validator

صفحه نخست سایت بوده. و ثبت نام و لاگ این در این صفحه انجام می شود. همچنین در صورت اینکه کاربر قبلا لاگ این کرده باشد، کاربر را به صفحه home منتقل می کند. در این کنترلر دو تابع به نام های register و login وجود دارد که وضیفه ثبت نام و لاگ این را به عهده دارند.کد این تابع register در زیر آورده شده:

def register(username, password):

from kernel.models.core import user\_session, User

new\_user = User(

username,

password

)

try:

user\_session.add(new\_user)

user\_session.commit()

except Exception, e:

user\_session.rollback()

return False

return True

در rgister تنها اطلاعات گرفته شده توسط کاربر در پایگاه ثبت می شود. برای آنکه چک نام کاربری قبلا ثبت نشده است تنها کافیست دستور ثبت را در یک try قرار دهیم. از آنجایی که فیلد username به صورت unique تعریف شده، در صورت وارد کردن مقدار تکراری پایگاه داده یک خطا بر می گرداند. و ما از این خطا استفاده می کنیم و از تکراری بودن فیلد مطلع می شویم و می توانیم خطایی مناسب را به کاربر نمایش دهیم.

و در login بعد از چک کردن فیلد ها در صورت صیح بودن اطلاعات یک سیشن ساخته می شود. کد این تابع در زیر آورده شده:

def login(username, password):

from kernel.models.core import user\_session, User

from flask import session

row = user\_session.query(User.id).\

filter\_by(username = username).\

filter\_by(password = password).first()

if row:

session['id'] = row[0]

return True

return False

این تایع در صورت موفق بودن عملیات مقدار true و در صورت عدم موفقیت مقدار false را بر می گرداند. در زمان استفاده از این تابع، در صورت true بودن مقدار بازگشتی کاربر را به صفحه home هدایت می کنیم. برای دیدن کد های کامل این کنترلر می توانید به بخش پیوست ها مراجعه کنید.

## Home: ارتباط 2 جدول از 2 پایگاه

صفحه خانه، داشبورد یا کنترل پنل کاربر می باشد. که بعد از لاگ این کردن، اولین صفحه ای است که کاربر با آن روبرو می شود. در این صفحه دو لیست وجود دارد. که یکی برنامه هایی است که کاربر آن ها را ایجاد کرده. و دیگری برنامه هایی از آن ها استفاده می کند. این دو لیسا از دو جدول select و create در پایگاه مخصوص هر کاربر گرفته می شود. از آن جا که تنها id برنامه ها در این جداول ذخیره می شود. ما احتیاج به بر قراری ارتباط با جدول tools در پایگاه core را داریم. کد های این بخش به شکل زیر خواهد بود.

from kernel.models.users import select\_session, Select

from kernel.models.users import create\_session, Create

from kernel.models.core import tool\_session, Tool

tool\_list = []

selects = select\_session.query(Select.tool\_id)

creates = create\_session.query(Create.tool\_id)

for i in selects:

tool\_list.append(i[0])

selects = tool\_session.query(Tool.id, Tool.name).filter(

Tool.id.in\_(

tool\_list

)

)

for i in creates:

tool\_list.append(i[0])

creates = tool\_session.query(Tool.id, Tool.name).filter(

Tool.id.in\_(

tool\_list

)

)

تنها راه حل ما این است که ابتدا مقادیر را از جدول ها ی پایگاه اول بگیریم و سپس آن ها را در جدول دوم جستجو کنیم. در پایین صفحه نویگیتوری قرار دارد که کاربر را به صفحات دیگر منتقل می کند. برای دیدن کدهای این کنترلر می توانید به پیوست ها مراجعه کنید.

## Create: یک تابع بازگشتی

در اینجا با گرفتن یک نام یکتا، یک برنامه جدید برای کاربر ایجاد می شود. ایجاد یک برنامه جدید مراحلی دارد که باید به ترتیب انجام شوند:

این مراحل توسط دو تابع register و clone انجام می شود. تابع clone یک تابع بازگشتی است و به صورت زیر عمل می کند:

def clone(source, dest):

from os import listdir

from os.path import isfile

for item in listdir(source):

if isfile(source + item):

source\_file = open(source + item , 'r')

dest\_file = open(dest + item , 'w')

dest\_file.write(source\_file.read())

else:

from os import mkdir

mkdir(dest + item)

clone(source + item + '/', dest + item + '/')

این تابع دو آرگومان دارد. و تمام فایل ها و فولدر ها و زیر فولدر ها source را در محل dest کپی می کند. ابتدا مقایسه می کند ایتم یک فایل است یا یک فولدر در صورت فایل بودن آن را ایجاد می کند و در صورت فولدر بودن یک فلدر ایجاد کرده و دوباره تابع clone را فرا خوانی می کند تا تمام زیر پوشه ها و زیر فایل های آن نیز ایجاد شود. تابع register فقط دو ثبت ساده انجام می دهد که احتیاجی به توضیح ندارد. کد های کامل این کنترلر در بخش پیوست ها موجود است.

## Tool: مفسر کد برنامه ها

این کنترلر بخش اصلی ابر ما بوده و کدهای کاربران را کمپایل می کند و نتیجه را به خروجی می برد. در ابتدا توسط تابع registered چک می کند که آیا این برنامه برای این کاربر ثبت شده است؟ در صورت ثبت نبودن کاربر را به صفحه لیسنس هدایت می کند و کاربر بعد از قبول آن اجازه استفاده از نرم افزار را خواهد داشت. این تایع به صورت زیر خواهد بود:

def registered(tool\_id):

from kernel.models.users import select\_session, Select

row = select\_session.query(Select.tool\_id).\

filter\_by(tool\_id = tool\_id).first()

if row:

return True

return False

بعد از آنکه تایید شد این برنامه برای کاربر ثبت شده است، زمان آن است که کد ها را کمپایل کرده و خروجی را به کاربر نمایش دهیم. برای این منظور ابتدا یک پرسه جدید ایجاد میکنیم. در این پرسه کمپایلر پایتون را فراخوانی می کنیم و کد های نوشته شده توسط کاربر را به کمپایلر ارسال می کنیم. نتیجه نهایی را از کمپایلر دریافت می کنیم و به خروجی می بریم. اگر بخواهیم دیاگرامی از مراحل کامپایل رسم کنیم به صورت زیر خواهد بود:

برای نوشتن کدهای این بخش از یکی از مژول های python به نام subprocess استفاده می کنیم. Popen برای ایجاد یک پرسه جدید، و PIPE برای دریافت خروجی برنامه استفاده می شود. کدها به صورت زیر خواهد بود:

from subprocess import Popen, PIPE

p = Popen(

[

'python',

source\_file

],

stdout = PIPE,

stderr = PIPE

)

output, err = p.communicate()

if err:

return err

return output

در اینجا source\_file آدرس فایلی است که کاربر در آن کدهای خود را قرار داده و output خروجی برنامه و err در صورت وجود خطا آن را نمایش می دهد

## Explorer و Editor: یک محیط توسعه

برای مدریت برنامه نیاز به یک explorer است تا به توانیم به فایل ها و فلدر های برنامه دسترسی داشته باشیم، یا در صورت نیاز یک فایل یا فولدر جدید برای برنامه خود ایجاد کنیم. در کنار این کنترلر دوکنترلر دیگر وجود دارد. یکی برای ایجاد فایل و دیگری برای ایجاد فولدر. در زمان لیست کردن، در صورت اینکه آیتم مورد نظر یک فولدر باشد، مسیر آن را به explorer لینک می دهیم. و در صورت فایل بودن، آن را به editor لینک می کنیم. در editor محتویات فایل نمایش داده می شود و پس از ویرایش می توانیم آن را ذخیره کنیم.

## Tools و Select: دسترسی و مجوز

برای آنکه کاربران به برنامه دیگر کاربران دسترسی داشته باشند، صفحه به نام tools وجود دارد که تمام برنامه های موجود را لیست می کند و کاربران با انتخاب یک برنامه ابتدا به صفحه select منتقل می شوند. در این صفحه لایسنس برنامه نمایش داده می شود. اگر کاربر لایسنس را قبول کند، آن برنامه برای کاربر ثبت می شود. و از آن پس کاربر می تواند از آن برنامه استفاده کند.

# فصل 4: نما ها

نما ها همان صفحات سایت می باشند. متشکل از کد های html، css و js. کدهای این صفحات در بخش پیوست ها موجود است. در اینجا با قرار دادن یک تصویر از هر صفحه به شرح مختصر وظایف آن می پردازیم.

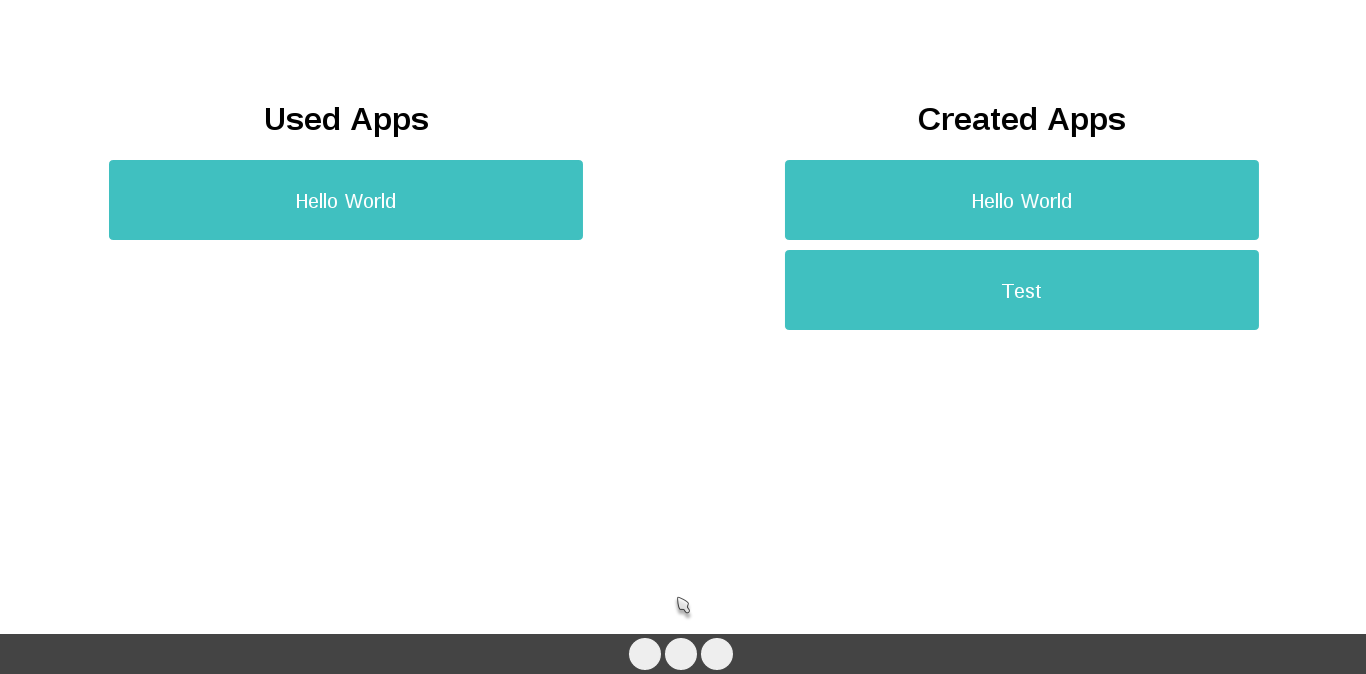
## gate: دروازه ورودی سایت



تصویر 6 صفحه gate

در این صفحه دو دکمه login و register دو عمل لاگین و ثبت نام را انجام می دهند. برای ثبت نام صفحه جدیدی باز نمی شود. زمانی که فیلد های username و password را پر می کنیم، در صورت فشردن دکمه register ابتدا اطلاعات را در پایگاه ذخیره می کند و سپس عمل لاگین را انجام می دهد.

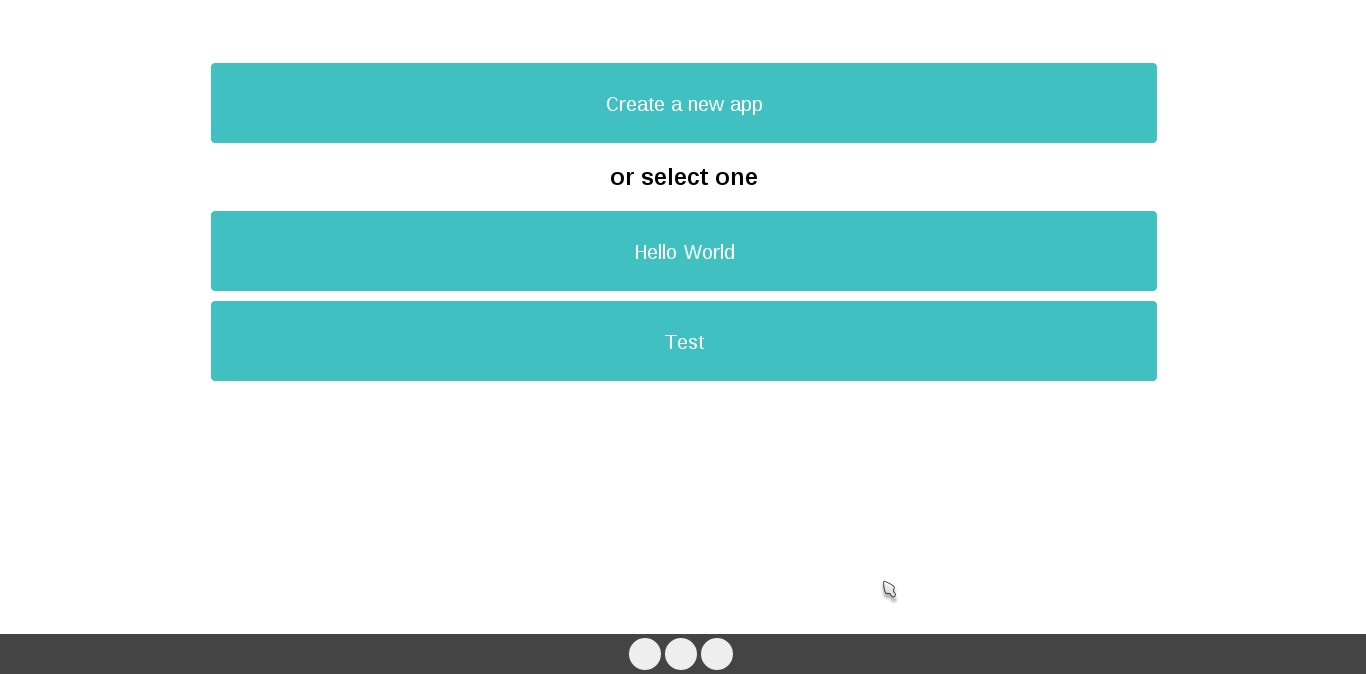
## home: داشبورد کنترل برنامه ها



تصویر 7 صفحه home

در این صفحه دو لیست قرار دارد. یکی لیست برنامه هایی است که کار بر آن ها را ایجاد کرده و دیگری لیست برنامه هایی که کار بر از آنها استفاده می کند.

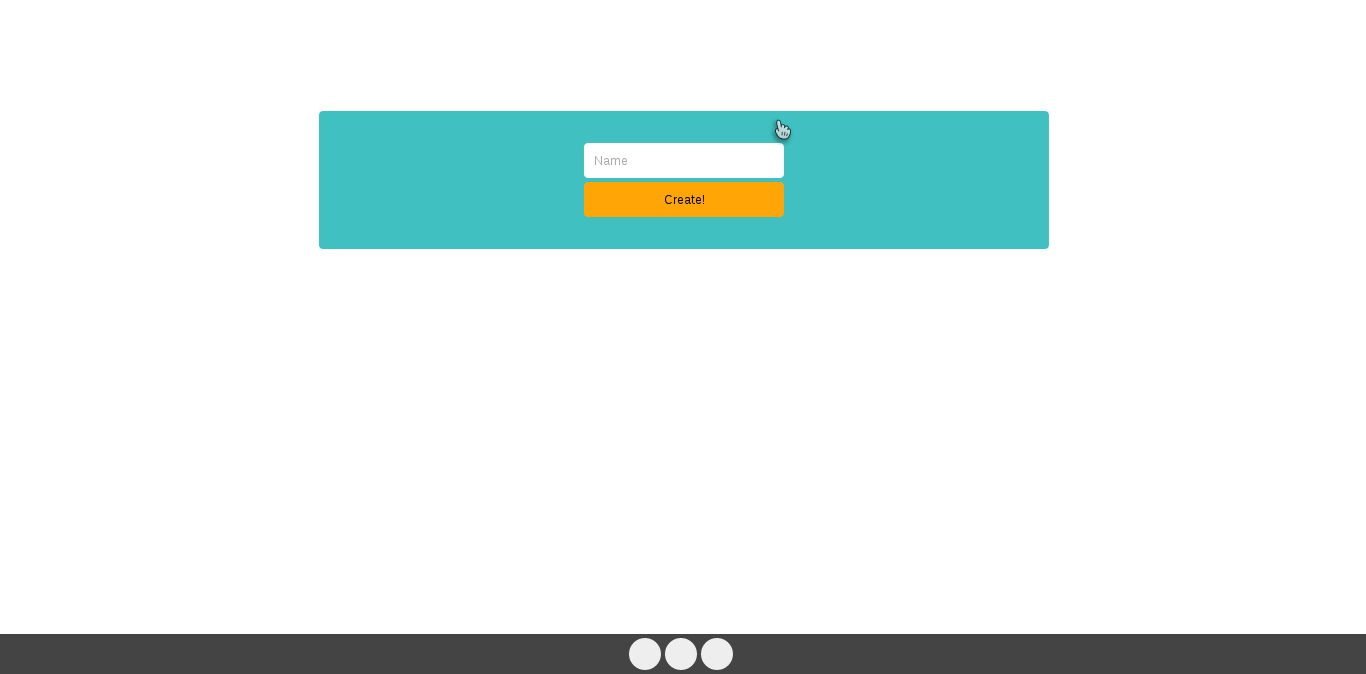
## tools: لیستی از تمام برنامه ها



تصویر 8 صفحه tools

در این صفحه لیستی از تمام برنامه های موجود در سرور در اختیار کار بر قرار می گیرد. و کاربر این امکان دارد تا با کلیک روی نام یکی از آن ها از آن برنامه استفاده کند. علاوه بر این گزینه create a new app به کار بر این امکان را می دهد تا یک برنامه جدید ایجاد کند.

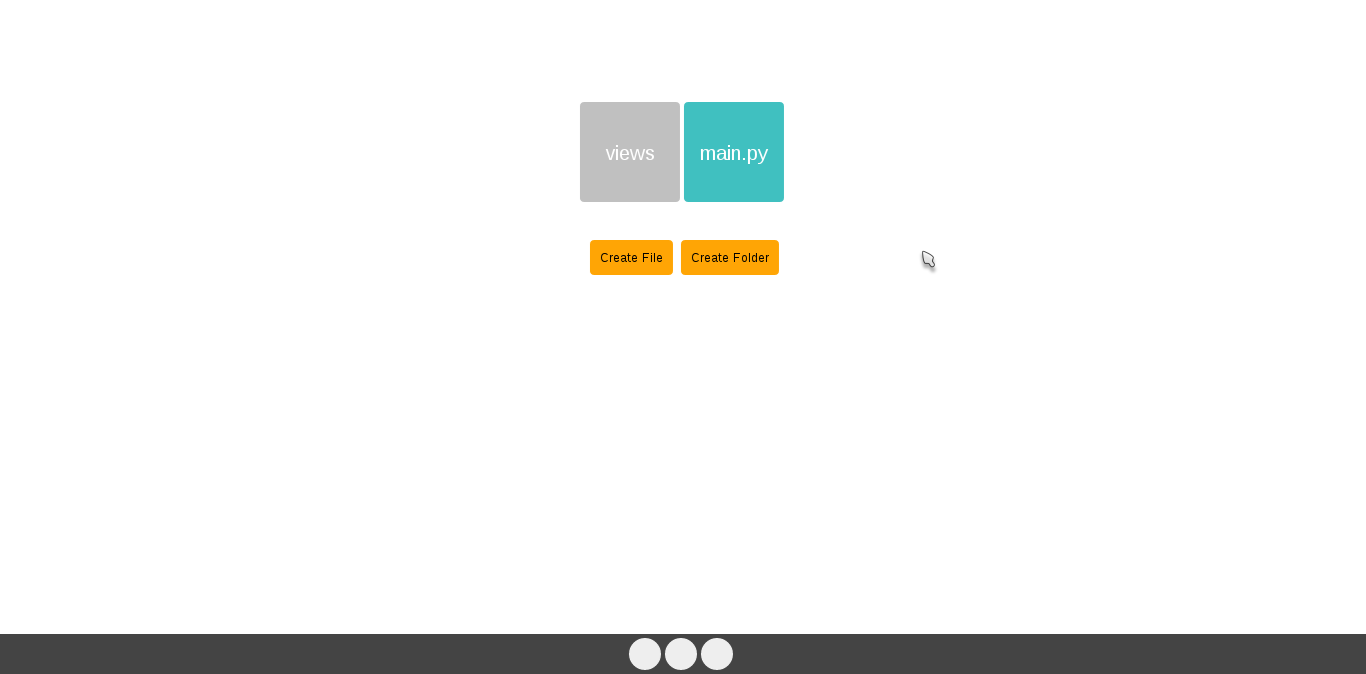
## create: ساخت یک برنامه جدید



تصویر 9 صفحه create

در این صفحه کاربر با انتخاب یک نام برای برنامه خود آن را ایجاد می کند. که پس کلیک بر روی دکمه create به صفحه explorer هدایت می شود.

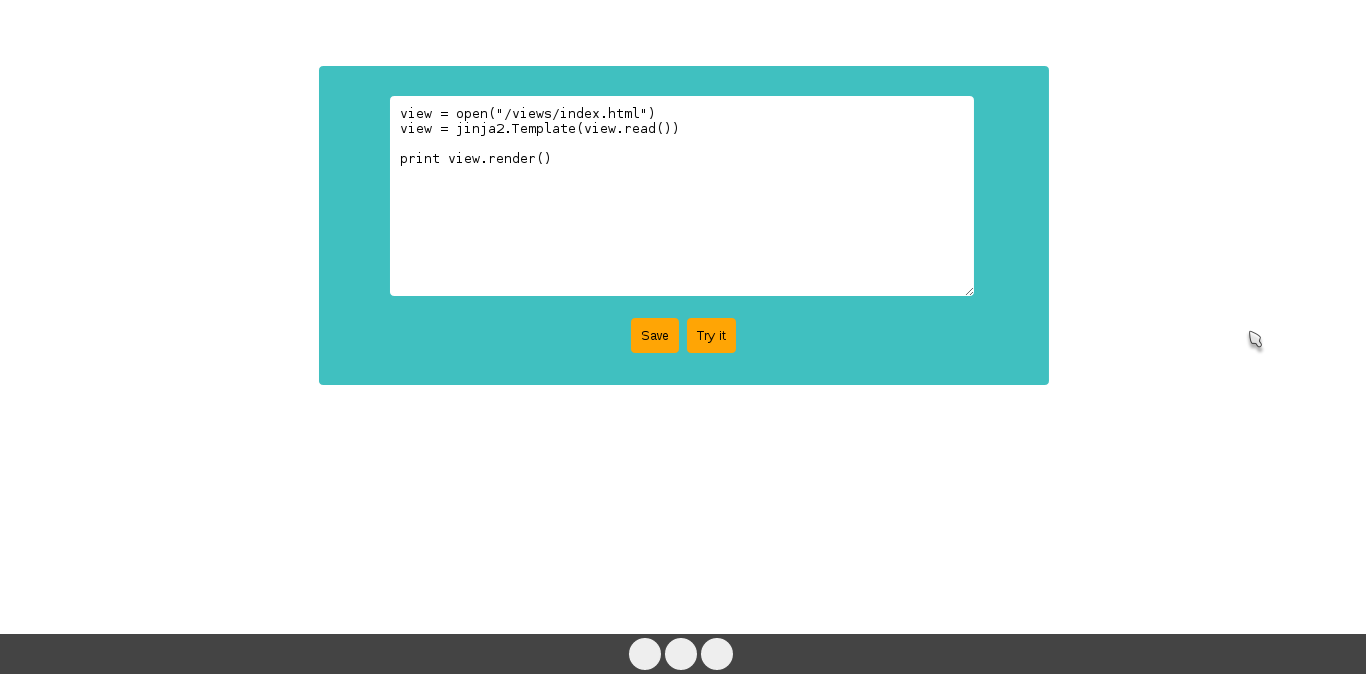
## explorer: یک مرورگر ساده



تصویر 10 صفحه explorer

این صفحه یک فایل منیجر ساده است. که لیستی از تمام فایل ها و فولدر های برنامه را به کاربر نشان می دهد. تفاوت فایل ها و فولدر ها با رنگ های مختلف مشخص می شود. و دو گزینه creat file و create folder برای ساخت فایل و فولدر استفاده می شود.

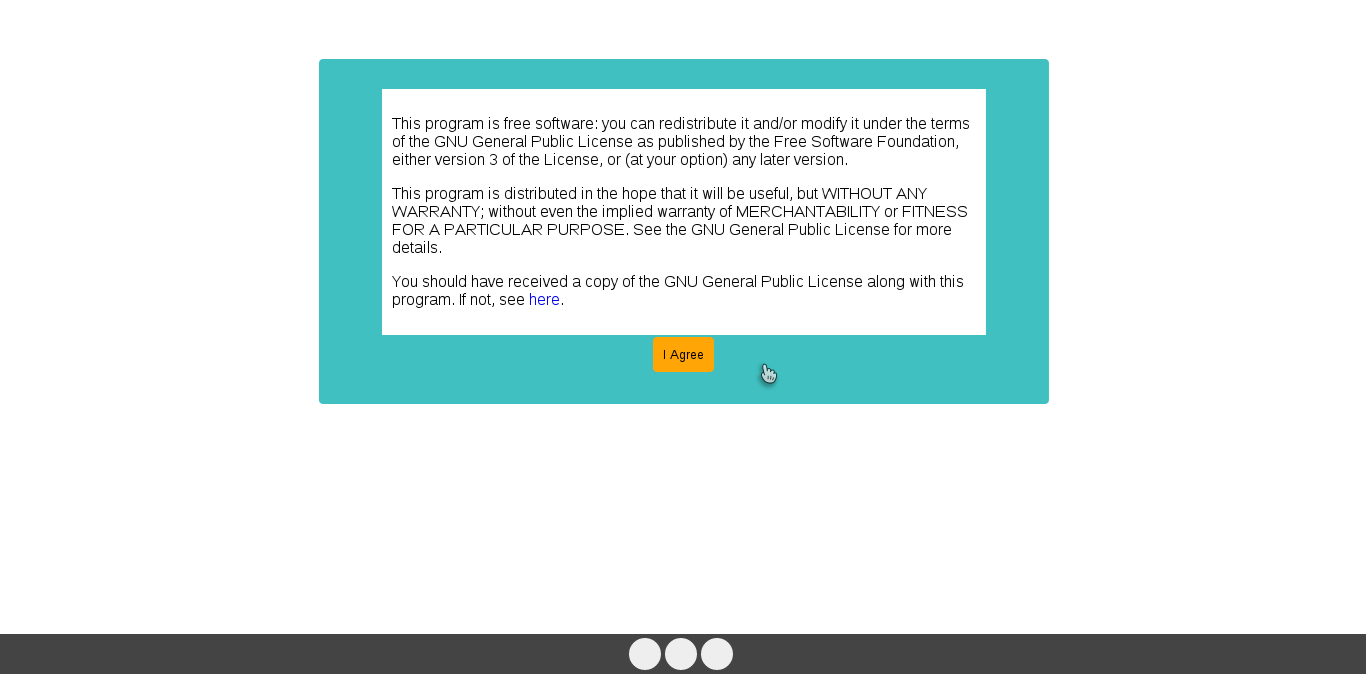
## editor: یک ویرایشگر ساده



تصویر 11 صفحه editor

دراینجا کاربر می تواند به ویرایش فایل های خود به پردازد. دو گزینه save و try برای ذخیره و امتحان کردن برنامه استفاده می شود.

## select: انتخاب یک برنامه جدید



تصویر 12 صفحه select

در صفحه به کاربر لایسنس برنامه نمایش داده می شود و کاربر پس از پذیرفتن آن به صفحه اصلی برنامه منتقل می شود و می تواند از برنامه استفاده کند.

# فصل 5: امنیت

بی شک مهمترین چالش ابر های رایانه ای امنیت آن ها است. و قتی قرار است یک نفر یا یک شرکت، تمام اطلاعات خود در اختیار یک شرکت اینترنتی قرار دهد. قطعا باید تضمین خوبی وجود داشته باشد تا به توانند به این شرکت اعتماد کنند. هر چند موضوع امنیت ابر ها بحث وسیعی است و خود این موضوع به تنهای عنوان اصلی صدها مقاله بوده، اما ذکر چند نکته امنیتی خالی از لطف نخواهد بود.

## جعبه امن[[29]](#footnote-29) (زندان chroot)

هنگامی که به کاربران اجازه اجرای یک کد روی سرور داده می شود، یعنی به آن ها اجازه انجام هر کاری را داده ایم. با یک کد ساده می توان تمام اطلاعات یک سرور را خالی کرد، یا تمام اطلاعات را از بین برد و یا حتی سرور را از کار انداخت. یکی از اهداف هکر ها در اکثر حملات، اجرای یک اسکریپت به روی سرور است. حال در این مدل ابر ما خود به کاربران امکان اجرای نه تنها اسکریپت بلکه نوشتن یک برنامه کامل را می دهیم. در این صورت ما حتما باید چاره ای بی اندیشیم تا هر کاربر تنها بتواند در برنامه خود تقیر ایجاد کند. و در کار سرور یا در برنامه دیگران اخلالی ایجاد نکند.

خوشبختانه لازم نیست ما همه کار را خودمان انجام دهیم. ابزار های برای این منظور ساخته شده. یکی از تکنیک های ساده و قدرتمند که در سیستم عامل های شبه unix وجود دارد، chroot نام دارد. این ابزار به این صورت عمل می کند که با تقیر مسیر ریشه، باعث می شود برنامه یا کاربر احساس کند مسیری که همکنون در آن قرار دارد مسیر ریشه است. اینگونه به اصطلاح یک جعبه امن ایجاد می کند که کاربر یا برنامه اجازه خروج از آن را ندارد.

برای ایجاد زندان chroot در python به صورت زیر باید عمل کنیم:

from os import chroot   
  
chroot(tool\_dir)

البته اگر این کد را در برنامه اصلی استفاده کنیم، با تغیر مسیر ریشه باعث ایجاد اختلال در کار برنامه شده. همان طور که قبلا گفتیم (به فصل کنترگرها مراجعه شود) برای اجرای برنامه ها ما ابتدا یک زیر پرسه مجزا ایجاد می کنیم. با اجرای این کد در زیر پرسه ما می توانیم این محدودیت را محدود به یک پرسه کنیم. این گونه سایر اجزای برنامه بدون مشکل کار خود را انجام می دهند.

## محدودیت منابع

در تکنیک قبل ما دسترسی کاربر را به یک پوشه محدود کردیم. اما حال مشکل دیگر وجود دارد. یک کاربر می تواند با اجرای یک حلقه بی نهایت، پردازنده سرور را اشغال کند. یا ممکن است با ساخت چندین متغیر تمام خانه های حافظ را پر کند. و با این کار مانع اجرای سایر برنامه ها شود. و یا حتی ممکن است بتواند در کار سیستم عامل نیز اخلال ایجاد کند.

برای حل این مشکل لازم است پرسه را که وضیفه اجرای برنامه ها را دارد، محدود به استفاده از حد مشخصی از منابع سیستم کنیم. این تکنیک به ما این امکان را می دهد تا اگر قسط کسب درآمد از ابر را داریم، با دریافت هزینه ای منابع بیشتری در اختیار کاربر قرار دهیم.

ما می توانیم محدودیت های مختلف را ایجاد کنیم. اما مهمترین محدودیت هایی که باید اعمال شوند، محدودیت در زمان استفاده از cpu و محدودیت در استفاده از فضای حافظ ram سرور می باشند.

برای ایجاد محدودیت منابع در python از ماژول resource استفاده می کنیم:

from resource import setrlimit, RLIMIT\_CPU, RLIMIT\_AS

setrlimit(RLIMIT\_CPU, (4, -1))

Setrlimit(RLIMIT\_AS, (1024, -1))

RLIMIT\_CPU برای ایجاد محدودیت در زمان استفاده از cpu بوده که بر حصب ثانیه می باشد. در این مثال یک برنامه اجازه ندارد بیش از 4 ثانیه cpu در اختیار بگیرد. و RLIMIT\_AS برای ایجاد محدودیت در تعداد خانه های حافظه ای است که یک برنامه می تواند در اختیار بگیرد. از آن جایی که هر خانه حافظه 8 بیت یا 1 بایت می باشد. پس می توان گفت واحد آن بایت است. در این یک برنامه تنها می تواند 1024 بایت معادل 1 کیلوبایت فضا حافظه استفاده کند.

# فصل 6: پیشنهاد ها و برنامه های آینده

این پروژه دارای پتانسیل بالایی است. شاید صدها صفحه دیگر به توان در مورد این موضوع نوشت. کارهای بسیاری بودند که مایل بودیم به روی این پروژه پیاده سازی کنیم. اما به علت محدودیت زمان به همین مقدار بسنده کردیم. در این فصل می خواهیم تعداد از این موارد را برشماریم.

## پشتیبانی از چند زبان برنامه نویسی

در این پروژه تنها نوشتن برنامه با زبان python امکان پزیر است. اما با انجام چند تغیر کوچک می توانیم کدهای زبان های مختلف را کمپایل کنیم. تنها کافیت در هنگام ثبت برنامه گزینه ای جهت انتخاب زبان برنامه قرار دهیم. در زمان کمپایل بجای ارسال کدها به کمپایلر python آن ها را به کمپایلر زبانی می فرستیم که کاربر در زمان ثبت انتخاب کرده بوده. کدهای این بخش تقریبا به صورت زیر خواهد بود:

from subprocess import Popen, PIPE

p = Popen(

[

compiler,

source\_file

],

stdout = PIPE,

stderr = PIPE

)

output, err = p.communicate()

if err:

return err

return output

که در آن compiler متغیری است که در آن نام کمپایلر که از داخل جدول برنامه ها گرفته شده قرار دارد.

## توسعه برنامه ها به کمک git

همانطور که می دانید git یک ابزار قدرتمند برای توسعه برنامه ها است. اکثر ابرهای کنونی git را پشتیبانی می کنند. حتی در بعضی از ابر ها تنها راه توسعه برنامه git می باشد. در این پروژه برای کاربران برای توسعه برنامه خود از یک explorer و یک editor ساده استفاده می کنند. که ابزار ها بسیاری از نیازهای کاربران را برآورده نخواهند کرد. به عنوان مثل اگر گروهی از کابران به طور همزان قصد کار روی یک پروژه را داشته باشند. یا اگر بخواهند برنامه خود را ابتدا در سیستم خود طراحی کنند و در انتها به سرور منتقل کنند. git این کارها را به راحتی و فقط با وارد کردن چند فرمان ساده انجام می دهد.

ارتباط با git کار سختی نیست. کافی است ابتدا کاربر را به یک سایت git مانند github ارجاع دهیم. کاربر بعد از ساخت اکانت ایجاد پروژه در آن سایت یک لینک دریافت می کند. ما از کاربر آن لینک را در خواست می کنیم. سپس به کمک فرمان های git یک کپی از برنامه را روی سرور خود قرار می دهیم. حال پس از هر با ویرایش برنامه کار باید دکمه به روز رسانی فاشار دهد تا برنامه به روز شود.

## استفاده از ssh برای ارتباط با سورس کد

یک ابزار دیگر که می تواند به توسعه برنامه ها کمک کند ssh بوده. در ابر های مختلفی می بینیم که به صورت های مختلفی از ssh استفاده می کنند. اگر به خواهیم برای هر کاربر یک اکانت ssh به سازیم لازم است در هنگام ثبت نام به ازای هر کاربر یک یسر در سرور بسازیم و برای هر کار یک اکانت ssh تعریف کنیم تا از این تریق بتوانند برنامه های خود را مدیریت کنند.

## ساخت یک زیردامنه برای هر برنامه

ساخت زیردامنه برای برنامه ها هیچ تفاوتی در عمل کرد آن ها ندارد. این کار تنها برای دسترسی آسان تر و برای ساخت url هایی به اصطلاح تمیز تر به کار می رود. به خصوص زمانی که بخواهیم برای برنامه ها route engine تعریف کنیم، استفاده از دامنه متفاوت برای هر برنامه می تواند بسیار مفید باشد.

## سازگاری با مجوز های مختلف

در این پروژه در حال حاضر تنها یک مجوز(لایسنس) وجود دارد و کاربران حق انتخاب مجوز متفاوتی برای برنامه خود ندارد. می توانیم با قردادن گزینه در زمان ساخت برنامه به کار حق انتخاب مجوز را بدهیم.

## اعمال محدودیت پهنای باند برای هر برنامه

محدودیت در پهنای باند یک ضرورت مهم است. در صورت وجود نداشتن چنین محدودیتی ممک یک برنامه تمام پهنای باند را درگیر کند و باعث اختلال در عمل کرد کل سیستم بشود. همچنین با باز گزاشتن پهنای باند ما راه را برای هکر ها باز گذاشته ایم تا به توانند حملاتی از نوع DoS را به راحتی انجام دهند. علاوه بر این می توان قابلیتی فراهم کرد تا کاربران با پرداخت بهایی به توانند پهنای باند بیشتری دراختیار داشته باشند.

# پیوست: کد برنامه

ساختار فایل های برنامه به صورت زیر می باشد:

/run.py

from flask import Flask

from kernel.controllers import \*

app = Flask(\_\_name\_\_)

app.register\_blueprint(blueprint)

app.secret\_key = 'the\_secret\_key'

if \_\_name\_\_ == "\_\_main\_\_":

app.debug = True

app.run('192.168.56.40', port = 80)

/kernel/\_\_init\_\_.py

def logged():

from flask import session

try:

if session['id']:

return True

except Exception:

return False

/kernel/config.py

DATA\_DIR = '/srv/data/'

VIEWS\_DIR = '/srv/kernel/views/'

TOOLS\_DIR = '/srv/tools/'

/kernel/models/\_\_init\_\_.py

#this file is empty

/kernel/models/core.py

from sqlalchemy import create\_engine

from sqlalchemy.orm import sessionmaker

from sqlalchemy.ext.declarative import declarative\_base

from kernel.config import DATA\_DIR

core\_engine = create\_engine(

'sqlite:///' +

DATA\_DIR +

'core.sqlite'

)

user\_session = sessionmaker(bind = core\_engine)

tool\_session = sessionmaker(bind = core\_engine)

user\_session = user\_session()

tool\_session = tool\_session()

Base = declarative\_base()

class User(Base):

from sqlalchemy import Column, Integer, String

\_\_tablename\_\_ = 'users'

id = Column(Integer, primary\_key = True)

username = Column(String, nullable = False, unique = True)

password = Column(String, nullable = False)

def \_\_init\_\_(self, username, password):

self.username = username

self.password = password

class Tool(Base):

from sqlalchemy import Column, Integer, String, Binary

\_\_tablename\_\_ = 'tools'

id = Column(Integer, primary\_key = True)

name = Column(String, nullable = False, unique = True)

creator = Column(Integer, nullable = False)

describe = Column(String)

license = Column(String)

def \_\_init\_\_(self, name, creator, describe,

license = 'GPL'):

self.name = name

self.creator = creator

self.describe = describe

self.license = license

Base.metadata.create\_all(core\_engine)

/kernel/models/users.py

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker

from sqlalchemy import create\_engine

from kernel.config import DATA\_DIR

from flask import session

Base = declarative\_base()

user\_engine = create\_engine(

'sqlite:///' +

DATA\_DIR +

'users/' +

str(session['id']) +

'.sqlite'

)

select\_session = sessionmaker(bind = user\_engine)

create\_session = sessionmaker(bind = user\_engine)

select\_session = select\_session()

create\_session = create\_session()

class Select(Base):

from sqlalchemy import Column, Integer

\_\_tablename\_\_ = "selects"

tool\_id = Column(

Integer,

nullable = False,

unique = True,

primary\_key = True

)

def \_\_init\_\_(self, tool\_id):

self.tool\_id = tool\_id

class Create(Base):

from sqlalchemy import Column, Integer, ForeignKey

\_\_tablename\_\_ = "creates"

tool\_id = Column(

Integer,

nullable = False,

unique = True,

primary\_key = True

)

def \_\_init\_\_(self, tool\_id):

self.tool\_id = tool\_id

Base.metadata.create\_all(user\_engine)

/kernel/controllers/\_\_init\_\_.py

from flask import Blueprint

from kernel.config import VIEWS\_DIR

blueprint = Blueprint(

'blueprint',

\_\_name\_\_,

template\_folder= VIEWS\_DIR

)

from os import path

from glob import glob

\_\_all\_\_ = [

path.basename(f)[:-3]

for f in glob(path.dirname(\_\_file\_\_) + "/\*.py")

]

\_\_all\_\_.append('blueprint')

/kernel/controllers/create.py

from kernel.controllers import blueprint

@blueprint.route("/create", methods=['GET', 'POST'])

def create():

from kernel import logged

if not logged():

from flask import redirect

return redirect('/')

from flask import request

try:

name = request.args['name']

create = request.args['create']

except Exception:

name = None

create = None

if name and create:

from flask import session

creator = session['id']

tool\_id = register(name, creator)

if not tool\_id:

return 'error'

from kernel.config import TOOLS\_DIR

template\_folder = TOOLS\_DIR + 'template/'

new\_tool\_folder = TOOLS\_DIR + str(tool\_id) + '/'

from os import mkdir

mkdir(new\_tool\_folder)

clone(template\_folder, new\_tool\_folder)

from flask import redirect

return redirect('/explorer?id=' + str(tool\_id))

from flask import render\_template

return render\_template(

'create.html',

tool = '',

path = ''

)

def register(name, creator):

from kernel.models.core import tool\_session, Tool

from kernel.models.users import create\_session, Create

new\_tool = Tool(

name,

creator,

''

)

try:

tool\_session.add(new\_tool)

tool\_session.commit()

except Exception, e:

tool\_session.rollback()

return None

row = tool\_session.query(Tool.id).\

filter\_by(name = name).first()

tool\_id = row[0]

new\_create = Create(

tool\_id

)

try:

create\_session.add(new\_create)

create\_session.commit()

except Exception, e:

create\_session.rollback()

return False

return tool\_id

def clone(source, dest):

from os import listdir

from os.path import isfile

for item in listdir(source):

if isfile(source + item):

source\_file = open(source + item , 'r')

dest\_file = open(dest + item , 'w')

dest\_file.write(source\_file.read())

else:

from os import mkdir

mkdir(dest + item)

clone(source + item + '/', dest + item + '/')

/kernel/controllers/editor.py

from kernel.controllers import blueprint

@blueprint.route('/editor', methods=['GET'])

def edit():

from kernel import logged

if not logged():

from flask import redirect

return redirect('/')

from flask import request

try:

path = request.args['path']

tool = request.args['id']

except Exception:

return 'invalid parameter(s)'

try:

code = request.args['code']

save = request.args['save']

except Exception:

code = None

save = None

from kernel.config import TOOLS\_DIR

from os.path import exists

full\_path = TOOLS\_DIR + tool + '/' + path

if(not exists(full\_path)):

return 'invalid path'

if save and code:

f = open(full\_path, 'w')

f.write(code)

from flask import redirect

return redirect(

'/editor?id=' +

tool +

'&path=' +

path

)

f = open(full\_path, 'r')

content = f.read()

from flask import render\_template

return render\_template(

'editor.html',

tool = tool,

path = path,

content = content

)

/kernel/controllers/explorer.py

from kernel.controllers import blueprint

@blueprint.route('/explorer', methods=['GET', 'POST'])

def explorer():

from kernel import logged

if not logged():

from flask import redirect

return redirect('/')

from flask import request

try:

path = request.args['path']

except Exception:

path = ''

try:

tool = request.args['id']

except Exception:

return 'invalid parameter(s)'

from kernel.config import TOOLS\_DIR

from os import listdir

from os.path import isfile, exists

full\_path = TOOLS\_DIR + tool + '/' + path

if(not exists(full\_path)):

return 'invalid path'

dirs = []

files = []

for item in listdir(full\_path):

if isfile(full\_path + '/' + item):

files.append(item)

else:

dirs.append(item)

if path == '/' or path == '':

back = None

else:

back = path[:path.rfind('/')]

from flask import render\_template

return render\_template(

'explorer.html',

path = path,

tool = tool,

dirs = dirs,

files = files,

back = back

)

@blueprint.route(

'/explorer/create\_file',

methods=['GET', 'POST']

)

def explorer\_create\_file():

from kernel import logged

if not logged():

from flask import redirect

return redirect('/')

from flask import request

try:

tool = request.args['id']

path = request.args['path']

except Exception:

return 'invalid parameter(s)'

try:

name = request.args['name']

create = request.args['name']

except Exception:

name = None

create = None

if name and create:

from kernel.config import TOOLS\_DIR

new\_file = TOOLS\_DIR + tool + '/' + path + '/' + name

open(new\_file, 'w').close()

from flask import redirect

return redirect('/explorer?id=' + tool)

from flask import render\_template

return render\_template(

'create.html',

tool = tool,

path = path

)

@blueprint.route(

'/explorer/create\_folder',

methods=['GET', 'POST']

)

def explorer\_create\_folder():

from kernel import logged

if not logged():

from flask import redirect

return redirect('/')

from flask import request

try:

tool = request.args['id']

path = request.args['path']

except Exception:

return 'invalid parameter(s)'

try:

name = request.args['name']

create = request.args['name']

except Exception:

name = None

create = None

if name and create:

from kernel.config import TOOLS\_DIR

from os import mkdir

from os.path import exists

new\_directory = TOOLS\_DIR + tool + '/' + path + '/' + name

if not exists(new\_directory):

mkdir(new\_directory)

from flask import redirect

return redirect('/explorer?id=' + tool)

from flask import render\_template

return render\_template(

'create.html',

tool = tool,

path = path

)

/kernel/controllers/gate.py

from kernel.controllers import blueprint

@blueprint.route("/", methods=['GET', 'POST'])

def gate():

from kernel import logged

if logged():

from flask import redirect

return redirect('/home')

from flask import request

if request.method == 'POST':

if request.form['username'] and request.form['password']:

username = request.form['username']

password = request.form['password']

button = request.form['submit']

else:

return 'error'

if(button == 'Register'):

if not register(username, password):

return 'Error'

if login(username, password):

from flask import redirect

return redirect('/home')

if(button == 'Login'):

if login(username, password):

from flask import redirect

return redirect('/home')

return 'Wrong Username or Password'

from flask import render\_template

return render\_template('gate.html')

def register(username, password):

from kernel.models.core import user\_session, User

new\_user = User(

username,

password

)

try:

user\_session.add(new\_user)

user\_session.commit()

except Exception, e:

user\_session.rollback()

return False

return True

def login(username, password):

from kernel.models.core import user\_session, User

from flask import session

row = user\_session.query(User.id).\

filter\_by(username = username).\

filter\_by(password = password).first()

if row:

session['id'] = row[0]

return True

return False

/kernel/controllers/home.py

from kernel.controllers import blueprint

@blueprint.route("/home")

def home():

from kernel import logged

if not logged():

from flask import redirect

return redirect('/')

from kernel.models.users import select\_session, Select

from kernel.models.users import create\_session, Create

from kernel.models.core import tool\_session, Tool

tool\_list = []

selects = select\_session.query(Select.tool\_id)

creates = create\_session.query(Create.tool\_id)

for i in selects:

tool\_list.append(i[0])

selects = tool\_session.query(Tool.id, Tool.name).filter(

Tool.id.in\_(

tool\_list

)

)

for i in creates:

tool\_list.append(i[0])

creates = tool\_session.query(Tool.id, Tool.name).filter(

Tool.id.in\_(

tool\_list

)

)

from flask import render\_template

return render\_template(

'home.html',

selects = selects,

creates = creates

)

/kernel/controllers/logout.py

from kernel.controllers import blueprint

@blueprint.route("/logout")

def logout():

from flask import redirect, session

session.clear()

return redirect('/')

/kernel/controllers/select.py

from kernel.controllers import blueprint

@blueprint.route('/select')

def select():

from kernel import logged

if not logged():

from flask import redirect

return redirect('/')

from flask import request

agree = None

try:

tool = request.args['id']

try:

agree = request.args['agree']

except Exception:

pass

except Exception:

return 'invalid parameter(s)'

if agree:

if registered(tool):

from flask import redirect

return redirect('/tools/' + tool)

else:

return 'Error'

from kernel.models.core import tool\_session, Tool

license = tool\_session.query(Tool.license).filter\_by(id = tool).first()

if not license:

return 'Not found'

from flask import render\_template

return render\_template(

'select.html',

tool = tool,

license = license

)

def registered(tool\_id):

from kernel.models.users import select\_session, Select

new\_select = Select(

tool\_id

)

try:

select\_session.add(new\_select)

select\_session.commit()

except Exception, e:

select\_session.rollback()

return False

return True

/kernel/controllers/tool.py

from kernel.controllers import blueprint

@blueprint.route('/tools/<tool\_id>', methods = ['GET', 'POST'])

def tool(tool\_id = None):

from kernel import logged

if not logged():

from flask import redirect

return redirect('/')

if not registered(tool\_id):

from flask import redirect

return redirect('/select?id=' + str(tool\_id))

from kernel.config import TOOLS\_DIR

tool\_folder = TOOLS\_DIR + str(tool\_id) + '/'

template\_folder = 'templates/'

preload\_file = open(TOOLS\_DIR + 'preload.py', 'r')

main\_file = open(tool\_folder + 'main.py', 'r')

code = preload\_file.read() + main\_file.read()

from subprocess import Popen, PIPE

p = Popen(

[

'python',

'-c ' + code,

tool\_id

],

stdout = PIPE,

stderr = PIPE

)

output, err = p.communicate()

if err:

return '<pre>' + err + '</pre>'

return output

from flask import render\_template

return render\_template(

'tool.html',

view = output

)

def registered(tool\_id):

from kernel.models.users import select\_session, Select

row = select\_session.query(Select.tool\_id).\

filter\_by(tool\_id = tool\_id).first()

if row:

return True

return False

/kernel/controllers/tools.py

from kernel.controllers import blueprint

@blueprint.route('/tools')

def tools():

from kernel.models.core import tool\_session, Tool

ids = []

names = []

rows = tool\_session.query(Tool.id, Tool.name)

for row in rows:

ids.append(row[0])

names.append(row[1])

from flask import render\_template

return render\_template(

'tools.html',

count = len(ids),

ids = ids,

names = names

)

/kernel/views/layout.html

<link rel="stylesheet" type="text/css" href="/static/layout.css">

<body>

<div id="content">

{% block content %}{% endblock %}

</div>

<div class="space"></div>

<div id="menu">

<div id="buttons">

<a href="/tools"><span class="button"></span></a>

<a href="/home"><span class="button"></span></a>

<a href="/logout"><span class="button"></span></a>

</div>

</div>

</body>

/kernel/views/create.html

{% extends "layout.html" %}

{% block content %}

<title>Cloud | Create</title>

<style>

textarea, input

{

width: 200px;

}

textarea

{

height: 100px;

}

</style>

<form method="get" class="form-1 center">

<input name="id" type="hidden" value="{{ tool }}">

<input name="path" type="hidden" value="{{ path }}">

<input name="name" placeholder="Name"><br>

<input name="create" type="submit" value="Create!" class="button-1">

</form>

{% endblock %}

/kernel/views/editor.html

{% extends "layout.html" %}

{% block content %}

<title>Cloud | Editor</title>

<style>

textarea{

height: 200px;

width: 80%;

}

</style>

<div class="form-1 center">

<form id="editor">

<input name="id" type="hidden" value="{{ tool }}">

<input name="path" type="hidden" value="{{ path }}">

<textarea name="code">{{ content }}</textarea>

<br>

</form>

<input name="save" type="submit" form="editor" value="Save" class="button-1">

<a href="tools/{{ tool }}" target="\_blank">

<button class="button-1">Try it</button>

</a>

</div>

{% endblock %}

/kernel/views/explorer.html

{% extends "layout.html" %}

{% block content %}

<title>Cloud | Explorer</title>

<div class="center">

{% if back != None %}

<a href="?id={{ tool }}&path={{ back }}">

<div class="folder">..</div>

</a>

{% endif %}

{% for dir in dirs %}

<a href="?id={{ tool }}&path={{ path }}/{{ dir }}">

<div class="folder">{{ dir }}</div>

</a>

{% endfor %}

{% for file in files %}

<a href="/editor?id={{ tool }}&path={{ path }}/{{ file }}">

<div class="file">{{ file }}</div>

</a>

{% endfor %}

<br><br><br>

<a href="explorer/create\_file?id={{ tool }}&path={{ path }}">

<button class="button-1">Create File</button>

</a>

<a href="explorer/create\_folder?id={{ tool }}&path={{ path }}">

<button class="button-1">Create Folder</button>

</a>

</div>

{% endblock %}

/kernel/views/gate.html

<title>Cloud | Gate</title>

<link rel="stylesheet" type="text/css" href="static/gate.css">

<div id="holder" class="center">

<div class="verical-center">

<div id="gate">

<h1>Wellcome</h1><br>

<form method="post">

<input name="username" placeholder="Username">

<br>

<input name="password" type="password" placeholder="Password">

<br>

<input name="submit" type="submit" value="Login">

<input name="submit" type="submit" value="Register">

</form>

</div>

</div>

</div>

/kernel/views/home.html

{% extends "layout.html" %}

{% block content %}

<title>Cloud | Home</title>

<div class="column-2">

<h1 class="center top">Used Apps</h1>

{% for tool in selects %}

<a href="tools/{{ tool[0] }}"><div class="box-1 center">{{ tool[1] }}</div></a>

{% endfor %}

</div>

<div class="column-2">

<h1 class="center top">Created Apps</h1>

{% for tool in creates %}

<a href="explorer?id={{ tool[0] }}"><div class="box-1 center">{{ tool[1] }}</div></a>

{% endfor %}

</div>

{% endblock %}

/kernel/views/select.html

{% extends "layout.html" %}

{% block content %}

<title>Cloud | Selcet</title>

<style>

#license

{

width: 80%;

background-color: #FFFFFF;

margin: 0 auto 0 auto;

text-align: left;

padding: 10px;

}

</style>

<div class="form-1 center">

<div id="license">

<p>This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.</p>

<p>This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.</p>

<p>You should have received a copy of the GNU General Public License along with this program. If not, see <a href="http://www.gnu.org/licenses">here</a>.</p>

</div>

<a href="?id={{ tool }}&agree=true"><button class="button-1">I Agree</button></a>

</div>

{% endblock %}

/kernel/views/tool.html

{% extends "layout.html" %}

{% block content %}

{{ view }}

{% endblock %}

/kernel/views/tools.html

{% extends "layout.html" %}

{% block content %}

<title>Cloud | Tools</title>

<a href="/create"><div class="box-1 center">Create a new app</div></a>

<h2 class="center">or select one</h2>

{% for i in range(0, count) %}

<a href="tools/{{ ids[i] }}"><div class="box-1 center">{{ names[i] }}</div></a>

{% endfor %}

{% endblock %}

/static/gate.css

body

{

display: table;

height: 90%;

width: 99%;

font-size: 100%;

font-family: sans-serif;

}

#holder{

display: table;

width: 400px;

height: 100%;

text-align: center;

}

#gate{

border-radius: 100%;

background-color: #777;

color: #fff;

padding: 20%;

}

#gate:hover{

background-color: #788;

}

.center{

margin: auto;

}

.verical-center{

display: table-cell;

vertical-align: middle;

}

/static/layout.css

body

{

display: table;

height: 90%;

width: 99%;

font-size: 100%;

font-family: sans-serif;

}

a

{

text-decoration: none;

}

textarea, input, button

{

border: none;

border-radius: 4px;

padding: 10px;

}

#content

{

display: table-cell;

vertical-align: middle;

}

/\* Menu \*/

#menu

{

width: 100%;

height: 40px;

position: fixed;

margin: 0px;

left: 0;

bottom: 0;

background-color: #444;

}

#menu #buttons

{

display: table;

margin-left: auto;

margin-right: auto;

}

#menu .button

{

display: table-cell;

float: left;

width: 32px;

height: 32px;

background-color: #eee;

margin-top: 4px;

margin-right: 4px;

border-radius: 50%;

}

#menu .button:hover

{

background-color: #ddd;

}

/\* classes \*/

.center

{

text-align: center;

}

.space

{

display: table-row;

height: 40px;

}

.box-1

{

display: block;

width: 70%;

height: 50px;

padding-top: 30px;

font-size: 20px;

border-radius: 4px;

margin: 0 auto 10px auto;

background-color: #40c0c0;

color: #FFFFFF;

overflow: visible;

}

.box-1:hover

{

background-color: #70C4D4;

}

.form-1

{

margin: 0 auto 0 auto;

padding: 30px 0 30px 0;

border-radius: 4px;

background-color: #40C0C0;

width: 54%;

}

.folder, .file

{

display: inline-block;

border-radius: 4px;

color: #FFFFFF;

padding-top: 40px;

font-size: 20px;

width: 100px;

height: 60px;

}

.folder

{

background-color: #C0C0C0;

}

.file

{

background-color: #40C0C0;

}

.button-1

{

background-color: #FFA505;

}

.column-2

{

float: left;

width: 50%;

}

.column-3

{

float: left;

width: 33%;

}

.cloumn-4

{

float: left;

width: 25%;

}

/tools/preload.py

# پیوست: آشنایی با رایانش ابری

رایانش ابری، یک مفهوم کاملا جدید نمی باشد؛ بلکه از فناوری پیشین رایانش توزیع یافته در مقیاس بزرگ نشأت گرفته است. با این حال، یک تکنولوژی دگرگون کننده خواهد بود که سومین انقلاب در صنعت IT به شمار می رود که نشان دهنده رویکرد توسعه ای صنعت IT از سخت افزار به نرم افزار، از نرم افزار به سرویس ها، و از سرویس های توزیع یافته به سرویس های متمرکز است. رایانش ابری همچنین، شکل جدیدی از رایانش تجاری است که بطور گسترده در آینده نزدیک مورد استفاده قرار خواهد گرفت. مفهوم اصلی رایانش ابری، کاه بار پردازشی بر درگاه های کاربری از طریق بهبود مستمر قابلیت اداره «ابر» است که در نهایت درگاه کاربری را به یک دستگاه ورودی و خروجی تبدیل کرده، و در هنگام تقاضا، به ظرفیت رایانشی قدرتمند ابر مجهز می سازد. همه اینها از طریق یک اتصال اینترنتی ساده با استفاده یک مرورگر استاندارد یا سایر اتصالات میسر می شود. با این حال، هنوز هم مشکلات فراوانی در رایانش ابری کنونی وجود دارد؛ یک تحقیق تازه نشان می دهد که امنیت داده و خطرات حریم خصوصی، به نگرانی اصلی برای مردمی که می خواهند به رایانش ابری متوسل شوند، تبدیل شده است.

## رایانش ابری چیست

### تعریف:

«ابر» یک ائتلاف مشترک از منابع رایانه ای است که می تواند:

• انواعی از بارهای کاری مختلف از جمله دسته ای از عملیات پشت زمینه ای و برنامه های تعاملی کاربر محور را مدیریت کند.

• به سرعت بار کاری را با استفاده از ارائه سریع ماشین های فیزیکی یا ماشین های مجازی، آرایش داده و افزایش دهد.

• از تکرار اطلاعات میان فایل های گوناگون، خود ترمیمی و مدل برنامه نویسی شدیدا مقیاس پذیر پشتیبانی می کند، بگونه ای که بتوان از خرابی های سخت افزاری/ نرم افزاری غیرقابل اجتناب رهایی یافت.

• بصورت بلادرنگ بر استفاده از منابع نظارت کرده، تخصیص منابع در هنگام نیاز را مجددا متوازن کند.

### مدل سرویس:

**نرم افزار به عنوان یک سرویس (SaaS):** نرم افزار به عنوان یک سرویس، نرم افزاری است که در اینترنت گسترش یافته تا پشت دیوار آتشین در شبکه محلی یا رایانه شخصی شما اجرا شود. این یک مدل، «پرداخت آنی» است و در اصل بطور گسترده برای اتوماسیون نیروی فروش و مدیریت رابطه مشتریان (CRM) بکار گرفته می شود.

**پلتفرم به عنوان یک سرویس (PaaS):** پلتفرم به عنوان یک سرویس، یک نوع SaaS است، نوعی رایانش ابری است که محیط توسعه را به عنوان یک سرویس فراهم می کند و می توانید از تجهیزات واسطه به منظور توسعه برنامه خود و انتقال آن به کاربران از طریق اینترنت یا سرورها استفاده کنید.

**زیرساخت به عنوان یک سرویس (IaaS):** زیرساخت به عنوان سرویس یک محیط مجازی سازی پلتفرم به عنوان سرویس است. بجای خرید سرور، نرم افزار، فضای پایگاه داده یا تجهیزات شبکه، مشتریان آن منابع را به عنوان یک سرویس کاملا از جانب منابع خارجی می خرند.

**سخت افزار به عنوان یک سرویس (HaaS):** مطابق با نظر نیکولاس کار، «ایده خرید سخت افزار IT یا حتی یک پایگاه داده کامل به عنوان بخشی از خدمت اشتراک پرداخت آنی است که به منظور ارضاء نیازهایتان، مقیاس آن افزایش یا کاهش می یابد. اما در نتیجه پیشرفت های سریع در مجازی سازی سخت افزاری ، اتوماسیون IT، و سنجش و قیمت گذاری مصرف، تصور می کنم که مفهوم سخت افزار به عنوان یک سرویس یا همان HaaS ممکن است در نهایت برای زمان آغازین آماده باشد». این مدل برای کاربران تجاری مفید است زیرا لازم نیست در ساختن و مدیریت پایگاه های داده سرمایه گذاری کنند.

### مدل آرایش و گسترش:

**ابر عمومی:** در ابرهای عمومی، مشتریان متعدد منابع رایانه ای ارائه شده توسط یک سرویس دهنده واحد را به اشتراک می گذارند، مشتریان می توانند به سرعت به این منابع دسترسی یابند و هزینه مربوط به منابع عملیاتی را پرداخت کنند. اگرچه ابر عمومی مزایای تأثیرگذاری دارد، لیکن خطر نهفته امنیت، تسلیم در برابر مقررات و کیفیت سرویس (QoS) وجود دارد.

**ابر خصوصی:** در ابر خصوصی، منابع رایانه ای توسط یک شرکت تجاری خصوصی مورد استفاده و نظارت قرار می گیرد. این ابر معمولا در پایگاه داده شرکت استفاده شده و توسط پرسنل داخلی یا سرویس دهنده مدیریت می شود. مهمترین مزیت این مدل آن است که امنیت، سازگاری و QoS تحت کنترل و نظارت شرکت های تجاری است.

**ابر مرکب (هیبرید):** نوع سوم می تواند ابر مرکب باشد که ترکیبی نوعی از ابر عمومی و خصوصی است. این ابر شرکت تجاری را قادر می سازد بار کاری حالت پایداری در ابر خصوصی اجرا کرده و زمانی که بار کاری ماکزیمم رخ می دهد، از ابر عمومی تقاضای منابع محاسباتی گسترده می کند و سپس در صورتی که بیش از آن نیاز نباشد، باز می گردد.

**ابر انجمنی:** سازمان های متعددی بطور مشترک، یک زیرساخت ابری یکسان ساخته و به همراه مواردی نظیر حریم خصوصی، مقررات، ارزش ها و ملاحظات به اشتراک می گذارند. انجمن ابری به درجه ای از مقیاس پذیری و تعادل دموکراتیک می رسد. زیرساختی ابری این ابر توسط یک فروشنده ثالث، یا در محدوده یکی از سازمان های داخل انجمن میزبانی می شود.

## امور مربوط به رایانش ابری

در چند سال اخیر، رایانش ابری از یک ایده تجاری امیدبخش به یکی از سریعترین بخش های در حال رشد از صنعت IT تبدیل شده است. حال، کمپانی های آسیب خورده از بحران اقتصادی، روز به روز بیشتر در حال درک این واقعیت هستند که تنها با بهره گیری از «ابرها» می توانند موفق به دسترسی سریع به برنامه های تجاری بهترین مدل شوند یا شدیدا منابع زیرساختی خود را تقویت کنند و همه اینها را با هزینه ای ناچیز انجام دهند. اما هرچه اطلاعات بیشتر و بیشتری از افراد و شرکت ها در ابر قرار می گیرد، نگرانی ها درباره این که یک محیط تا چه حد ایمن است، آغاز می شود.

### امنیت

داده های شما در کجا از امنیت بیشتری برخوردارند، در هارد درایو شبکه محلی تان سا در سرورهای شدیدا ایمن موجود در ابر؟ برخی افراد ادعا می کنند که داده های مشتریان هنگامی که بصورت داخلی مدیریت می شود، ایمن تر است در حالیکه برخی دیگر مدعی هستند که تأمین کنندگان ابر از انگیزه قوی برای حفظ اعتماد برخوردار هستند و بدین ترتیب از سطح امنیتی بالاتری استفاده می کنند. با این حال، در ابر، فارغ از اینکه آرشیو داده اصلی شما نهایتا در کجا ذخیره شده است، داده های شما در میان این رایانه های فردی توزیع می شود. هکرهای زبردست و کوشا می توانند در عمل هر سروری را مورد تهاجم قرار دهند و آماری وجود دارد که نشان می دهد یک سوم رخنه های امنیتی ناشی از رایانه های قابل حمل و دیگر دستگاه هایی که به سرقت رفته یا گم شده اند یا ناشی از این است که کارمندان بطور تصادفی داده ها را در تماس با اینترنت قرار داده اند و تقریبا 16% نیز بدلیل سرقت داخلی بوده است.

### حریم خصوصی

رایانش ابری که متفاوت از مدل رایانش سنتی است، از تکنولوژی رایانش مجازی بهره برده و داده های شخصی کاربران ممکن است بجای آنکه در همان موقعیت فیزیکی یکسان بماند، در پایگاه های داده مجازی متعدد پخش شده و یا حتی به آن سوی مرزهای کشور برود که در همین زمان، حفاظت از حریم خصوصی داده ها با مناقشه بر سر سیستم های حقوقی مختلف مواجه خواهد شد. از سوی دیگر ممکن است کاربران در هنگام دسترسی به سرویس های رایانش ابری منجر به نشت اطلاعات شوند. مهاجمان می توانند وظیفه بحرانی را بر حسب وظیفه رایانه ای ثبت شده توسط کاربران تحلیل کنند.

### قابلیت اطمینان

سرورهای موجود در ابر با مشکلاتی همچون مشکلات سرورهای محل سکونت شما مواجه هستند. سرورهای ابر همچنین زمان های از کار افتادگی و افت سرعت هایی را تجربه می کنند که فرق آنها در این است که در مدل رایانش ابری، کاربران وابستگی بیشتری به یک تأمین کننده سرور ابری (CSP) دارند. هنگامی که یک CSP خاص را انتخاب می کنید، تفاوت های زیادی بین مدل سرویس CSP وجود دارد که ممکن است اسیر آن شوید و خطر امنیتی بالقوه ای برایتان به همراه داشته باشد.

### مسائل حقوقی

فارغ از تلاش های صورت گرفته به منظور به جریان انداختن موقعیت قانونی، از سال 2009، تهیه کنندگانی نظیر سرویس های وب آمازون، با توسعه جاده محصور و شبکه ریلی و دادن اجازه انتخاب «مناطق قابل دسترس » به کاربران، به بازارهای بزرگی دست یافته اند. از سوی دیگر، نگرانی های درباره تدابیر امنیتی و محرمانه بودن اطلاعات از سوی فرد در تمامی سطوح قانونی وجود دارد.

### استاندارد باز

استانداردهای باز برای پیشرفت رایانش ابری ضروری هستند. اکثر تهیه کنندگان ابری، API هایی را نمایش می دهند که عموما بخوبی مستند می گردند ولی همچنین در اجرا نیز منحصر بفرد هستند و بدین ترتیب قادر به کار در سیستم های مختلف هستند. برخی از فروشندگان از API های دیگری استفاده می کنند و تعدادی استاندارد باز تحت ساخت نیز همانند واسط رایانش ابری باز OGF وجود دارد. کنسرسیوم ابر باز (OCC) مشغول فعالیت به منظور ایجاد توافق عمومی درباره استانداردها و تکنیک های اولیه رایانش ابری است.

### سازگاری

مقررات متعددی که به ذخیره سازی و استفاده از داده ها مربوط می شود مستلزم گزارش دهی منظم و پیگیری است، تهیه کنندگان ابر می بایست به مشتریان خود این قابلیت را بدهند تا بطور صحیح با این مقررات، سازگاری داشته باشند. مدیریت سازگاری و امنیت رایانش ابری منجر به درکی درباره این مطلب می شود که نگاه بالا به پایین تمامی منابع IT در یک موقعیت ابر محور چگونه می تواند مدیریت و اعمال قانون قویتری در سیاست های سازگاری داشته باشد. علاوه بر مقرراتی که مشتریان را مخاطب قرار می دهد، مراکز داده تحت حمایت تهیه کنندگان ابر نیز ممکن است در معرض الزامات سازگاری باشند.

### آزادی

رایانش ابری به کاربران اجازه برخورداری فیزیکی از ذخیره سازی داده نمی دهد و کنترل و ذخیره سازی داده را در دستان تهیه کندگان ابر قرار می دهد. مشتریان ادعا می کنند که این مسأله ای کاملا بنیادی است که به آنها توانایی حفظ نسخه های خود از داده ها را به شکلی که آزادی انتخاب ایشان حفظ شده و آنها را در برابر برخی مسائل خاص خارج از کنترل آنها حفاظت کند، می دهد در حالیکه مزایای بی اندازه زیاد رایانش ابری را نیز به همراه دارد.

### امکان دوام بلند مدت

شما می بایست اطمینان داشته باشید که داده هایی که در ابر قرار می دهید هرگز باطل و بی اعتبار نمی شوند، حتی اگر تهیه کننده ابر شما با شکست مواجه شده یا شرکتی بزرگتر، آن را به چنگ آورده و در خود ادغام کند. گارتنر می گوید «از تهیه کنندگان احتمالی بپرسید که چنانچه داده هایتان در قالبی باشد که بتوانید آن را به یک برنامه جایگزینی وارد کنید، چگونه آنها را دریافت خواهید کرد».

### راه حل

به منظور پیشرفت رایانش ابری، جامعه می بایست تدابیری پیشگیرانه اتخاذ کند تا امنیت آن تضمیم شود. راه حل مقاله برکلی، رمزدار کردن داده ها است. پیش از ذخیره سازی داده در موقعیت مجازی، آن را با قفل های خود رمزدار کرده و اطمینان بیابید که فروشنده برای ارائه گواهی های امنیتی و ممیزی های خارجی آماده است. مدیریت هویت، کنترل دسترسی، گزارش دهی وقایع امنیتی، مدیریت لایه فیزیکی و پرسنل می بایست پیش از انتخاب یک CSP مورد ارزیابی قرار گیرند. و می بایست اطلاعات شخصی فرستاده شده به ابر و ذخیره شده در آن را به حداقل برسانید. CSP می بایست کنترل کاربر را به بیشترین میزان رسانده و بازخورد ارائه دهد. سازمان ها می بایست برنامه ها و انتقال داده را در ابر خصوصی خود اجرا کرده و سپس آن را به ابری عمومی تغییرشکل دهند. اگرچه مسائل حقوقی زیادی در ارتباط با رایانش ابری وجود دارد، پیمان امنیت ابری می بایست به سرعت استانداردهای مرتبطی طراحی کند.

# پیوست: آشنایی با معماری MVC

MVC یک معماری سه لایه است که در سال ۱۹۷۰ و برای زبان smalltalk ایجاد شد. هدف از ایجاد این معماری این بود که قسمت کد نویسی یا منطق برنامه را از قسمت طراحی آن جدا کند. مزایایی که این کار دربر داشت این بود که نگهداری و تغییر در کدها را در آینده بسیار راحت می کرد، کدهایی نوشته شده قابل استفاده مجدد بودند و مهمترین قابلیت این بود که یک طراح و یک برنامه نویس می توانستند روی یک پروژه در آن واحد کار کنند بدون اینکه اختلالی در کار هم به وجود بیاورند.با فراگیر تر شدن وب برنامه نویسان وب تصمیم به استفاده از این معماری در طراحی برنامه های خود گرفتند.معماری سه لایه از سه قسمت Model، View، Controllerتشکیل میشود. که در زیر به شرح مختصری در مورد هر کدوم از لایه ها می پردازیم.

## لایه Model

این لایه برای ارتباط با دیتابیس استفاده میشه. این لایه باید اجازه دسترسی، تغییر یا اضافه کردن داده ها رو بده. این لایه در واقع یک پل بین لایه View و لایه Controller می باشد. یکی از مهمترین خاصیت های این لایه این است که "نابیناست" به این معنی که مدل نمیدونه وقتی که داد هارو به View یا کنترلر ارسال کرد قرار چه اتفاقی براشون بیافته و به دنبال پاسخی از Controller یا View نیست.تنها هدفش اینه که داده هارو ذخیره کنه یا زمانی که درخواستی از بقیه لایه ها ارسال شد تغییرشون بده

در خیلی از جاها گفته شده که لایه مدل نباید با لایه ویو ارتباط داشته و فقط و فقط باید با لایه کنترلر در ارتباط باشه.ولی چیزی که در مورد این لایه مهمه اینه که این لایه مسئول ارتباط با دیتابیسه. حذف، اضافه و ویرایش داده های دیتابیس توی این لایه انجام میشه.

## لایه View

این لایه جاییه که داده ها از مدل گرفته میشن و به صورت خروجی به کاربر نمایش داده میشن.در برنامه های وب این لایه جاییه که کدهای HTML ساخته و نمایش داده میشن.توی این لایه ارتباط با کاربر انجام میشه و با کنترلر درخواست کاربر رو انجام میده.به عنوان مثال یک button رو درنظر بگیرید که وقتی روش کلیک شد یکی از action های کنترلر رو صدا میزنه.

یک سری تصورات غلط در مورد این لایه در برنامه نویسان وب وجود داره. یکی از این تصورات اشتباه اینه که لایه View نباید هیچ ارتباطی با لایه Model داشته باشه و باید همه اطلاعات رو از لایه Controller بگیره.مثلا در فریم ورک Cakephp و اکثر فریم ورکهای PHP این اشتباه وجود داره. در واقع این تفکر نادیده گرفتن کامل تئوریه پشت معماری سه لایه است.

این نکته خیلی مهمه که توجه داشته باشید برای پیاده سازی درست معماری MVC لایه View نباید با لایه Model ارتباط داشته باشه و منطق برنامه باید فقط در لایه Controller انجام بشه.

از اونجا که بین علما اختلاف نظر وجود داره تحقیق بشتری کردم و در این مقاله به نقل از گروه Gang of four در مورد معماری MVC اینطور نوشته :

MVC شامل مدل ها و ویو هاییه که میتونن با هم ارتباط داشته باشن. یک View باید مطمئن باشه که ظاهری که الان میخواد به خودش بگیره باید حالتی از model باشه.هر وقت که دیتا در لایه model تغییر کرد یک پیغام به ویو میفرسته که بسته به اون تغییر کنه.

بازم چیزی که در مورد این لایه مهمه اینه که این لایه وظیفه نمایش داده ها و گرفتن و ارسال اون رو به لایه Controller داره.ما هم زیاد سخت نمیگیریم و هر طوری که راحتیم فریم ورک خودمون رو مینویسم چون قرار نیست کسی به خاطر اینکه آیا این دولایه باید باهم ارتباط داشته باشن یا نه ، مارو مجازات کنه.

## لایه Controller

این لایه منطق برنامه رو کنترل میکنه.این لایه ورودی ها رو میگیره و درخواست های کاربر رو انجام میده ، اگر نیاز باشه از دیتابیس مقادیری برای کاربر ارسال بشه رو از لایه Model میگیره و به لایه View ارسال میکنه.

به هر حال در باره کارکرد این سه لایه باهم نقل قول های زیادی شده که هر کی هرجور دلش خواسته باهاش بخورد کرده.شما با هر کدوم راحت ترید کار کنید ولی اینو بدونید که اصل موضوع چیه.

# پیوست: آشنایی با تکنیک ORM

ORM یا Object Relational Mapping یک تکنیک برنامه نویسی برای تبدیل ارتباطات در دیتابیس به مفاهیم شی گرایی[[30]](#footnote-30) در برنامه نویسی است. در واقع می‌توان گفت که ORM کلا‌س ها را به جدول[[31]](#footnote-31) ها مپ[[32]](#footnote-32) می‌کند.

ORM برای این کار، یک Framework تهیه می‌کند. و توصیه می کند که کاربرها هنگام پیاده سازی از یک Framework استفاده نمایند. وقتی که شما می‌خواهید به دیتابیس دسترسی پیدا کنید، یا اطلاعاتی را ذخیره کنید، این کارها را مستقیما بر‌ روی اشیاء[[33]](#footnote-33) انجام می‌دهید. در واقع ORM با کپسوله کردن مراحل دسترسی به دیتابیس، دشواری‌های کار با پایگاه داده را از دید کاربران و برنامه‌نویسان پنهان می‌کند.

مزیت بکارگیری ORM مدیریت ساده داده[[34]](#footnote-34) در برنامه نویسی شی گرا است. و اینکه اگر قرار به عوض شدن پایگاه داده باشد، این تبدیل تنها با عوض کردن یک بخش از Framework که وظیفه برقراری ارتباط با دیتابیس را بر عهده دارد (در برخی از برنامه‌های موجود با تغییر خصوصیت نام پایگاه داده) صورت می‌پذیرد.

از مشکلاتی که ORM می‌تواند بوجود آورد میتوان به این مورد اشاره کرد که به خاطر فضای زیادی که اشغال می‌کند، باعث پایین آمدن Performance در سیستم می‌شود. ولی به خاطر مزایایی که دارد (که بخصوص در پروژه‌های بزرگ حس می‌شود) استفاده از آن توصیه می‌شود.

زمان به ما ثابت کرده که پایگاه داده‌های رابطه‌ای ساختار مناسبی برای ذخیره‌سازی داده‌ای هستند، همچنین به این نتیجه رسیده‌ایم که برنامه‌نویسی شیءگرا یک رویکرد بسیار خوب و قوی برای پیاده‌سازی سیستم‌های بسیار پیچیده می‌باشد.

ORM عملا یک لایه مترجم بین زبان برنامه‌نویسی و پایگاه داده رابطه‌ای است که این دو را به هم تبدیل می‌کند و در عمل باعث می‌شود که این دو حیطه کاملا متفاوت زبان یکدیگر را به خوبی بشناسند و با هم تبادل اطلاعات داشته باشند. این مفهوم که مانند یک پل بین این دو حیطه می‌ماند قابلیت‌های زیادی را برای ما تهیه می‌نماید.

به طور اختصار می‌توان گفت که وظیفه ORM پایدار کردن خودکار آبجکت‌های موجود در یک برنامه روی جداول در پایگاه داده رابطه‌ای می‌باشد که برای این کار از متادیتاهایی برای نگاشت بین آبجکت‌ها و پایگاه داده استفاده می‌نماید. در ادامه به فوايد استفاده از ORM ها خواهيم پرداخت:

## كاهش زمان تحويل پروژه

اولين و مهمترين دليلي كه بر اساس آن در يك پروژه، استفاده از ORM حائز اهميت مي شود، بحث بالا بردن سرعت برنامه نويسي و كاهش زمان تحويل پروژه به مشتري است. اين كاهش زمان بسته به نوع پروژه بين 20 تا 50 درصد مي تواند خود را بروز دهد.

بديهي است ابزارهاي ORM كار شگفت انگيزي را قرار نيست انجام دهند و شما مي توانيد تمام آن عمليات را دستي هم به پايان رسانيد؛ اما اجازه دهيد يك مثال كوتاه را با هم مرور كنيم.

براي پياده سازي يك برنامه متداول با حدود 15 تا 20 جدول، حدودا به 30 شيء براي مدل سازي سيستم نياز خواهد بود و برنامه نويسي اين مجموعه بين 5000 تا 10000 سطر كد را به خود اختصاص خواهد داد. بديهي است برنامه نويسي و آزمايش اين سيستم چندين هفته يا ماه به طول خواهد انجاميد.

اما با استفاده از يك ORM ، عمده وقت شما به طراحي سيستم و ايجاد ارتباطات بين اشياء و ديتابيس در طي يك تا دو روز صرف خواهد شد. ايجاد كد بر اساس اين مجموعه و با كمك ابزارهاي ORM، آني است و با چند كليك صورت مي گيرد.

## كدي با طراحي بهتر

ممكن است شما بگوئيد كه كد نويسي من بي نظير است و از من بهتر كسي را نمي توانيد پيدا كنيد! به تمامي زواياي كار خود مسلطم و نيازي هم به اين گونه ابزارها ندارم!

عده اي از شما به طور قطع اينگونه ايد؛ اما نه همه. در يك تيم متوسط، همه نوع برنامه نويس با سطوح مختلفي را مي توانيد پيدا كنيد و تمامي آن ها برنامه نويس ها و يا طراح هاي آنچنان قابلي هم نيستند. بنابراين امكان رسيدن به كدهايي كه مطابق اصول دقيق برنامه نويسي شيء گرا نيستند و در آن ها الگوهاي طراحي به خوبي رعايت نشده، بسيار محتمل است. همچنين در يك تيم زمانيكه از يك الگوي يكسان پيروي نمي شود، نتايج نهايي بسيار ناهماهنگ خواهند بود. در مقابل استفاده از ORM هاي طراحي شده توسط برنامه نويس هاي قابل (senior (architect level) engineers)، كدهايي را بر اساس الگوهاي استاندارد و پذيرفته شده ي شيء گرا توليد ميكنند و همواره يك روند كاري مشخص و هماهنگ را در يك مجموعه به ارمغان خواهند آورد.

## نياز به تخصص کم تر

قسمت دسترسي به داده ها يكي از اجزاي كليدي كارآيي برنامه شما است. اگر طراحي و پياده سازي آن ضعيف باشد، كل برنامه را زير سؤال خواهد برد. براي طراحي و پياده سازي دستي اين قسمت از كار بايد به قسمت هاي بسياري از مجموعه ي دات نت فريم ورك مسلط بود. اما هنگام استفاده از يك ORM مهمترين موردي را كه بايد به آن تمركز نمائيد بحث طراحي منطقي كار است و ايجاد روابط بين اشياء و ديتابيس و امثال آن. مابقي موارد توسط ORM انجام خواهد شد و همچنين مي توان مطمئن بود كه پياده سازي خودكار انجام شده اين قسمت ها، بر اساس الگوهاي طراحي شيء گرا است.

## كاهش زمان آزمايش

بديهي است اگر قسمت دسترسي به داده ها را خودتان طراحي و پياده سازي كرده باشيد، زمان قابل توجهي را نيز بايد به بررسي و آزمايش صحت عملكرد آن بپردازيد و الزامي هم ندارد كه اين پياده سازي مطابق بهترين تجربيات كاري موجود بوده باشد. اما هنگام استفاده از كدهاي توليد شده توسط يك ابزار ORM مي توان مطمئن بود كه كدهاي توليدي آن كه بر اساس يك سري الگوي ويژه توليد مي شوند، كاملا آزمايش شده هستند و همچنين صدها و يا هزارها نفر در دنيا هم اكنون دارند از اين پايه در پروژه هاي موفق خود استفاده مي كنند و همچنين بازخوردهاي خود را نيز به تيم برنامه نويسي آن ابزار ORM ارائه مي دهند و اين مجموعه مرتبا در حال بهبود و به روز شدن است.

# پیوست: آشنایی معماری REST

REST يك مدل معماري براي طراحي برنامه‌هاي كاربردي شبكه است كه ترکیبی از چند مدل معماری مبتنی بر شبکه مي‌باشد و محدودیت‌هايی جهت تعریف يك واسط اتصال یکنواخت براي آن درنظر گرفته شده است.

REST مخفف عبارت Representational State Transfer است و متكي بر يك پروتكل ارتباطي بدون حالت، كلاينت سرور و با قابليت cache كردن مي‌باشد كه در اكثر موارد پروتكل HTTP مورد استفاده قرار مي‌گيرد.

ايده اصلي معماري REST اين است كه به جاي استفاده از مكانيزم هاي پيچيده اي مانند CORBA، RPC يا SOAP براي اتصال ماشين ها ازHTTP ساده براي برقراري ارتباط بين ماشين‌ها استفاده شود.

مدل REST شش قيد براي معماری برنامه‌هاي شبكه تعريف مي‌كند:

* Client-Server: كلاينت سرور باشد.
* Stateless: بدون حالت باشد.
* Cacheable: قابليت cache داشته باشد.
* Layered System: سيستم لايه‌بندي شده داشته باشد.
* Uniform Interface: واسط يكنواخت داشته باشد.
* Code on Demand: داراي قابليت كد در صورت نياز باشد. (كه اين محدوديت اختياري مي‌باشد)

به سيستمي كه اين قيود را رعايت نمايد، RESTful مي‌گويند. از لحاظ رويكرد برنامه نويسي REST جايگزيني ساده براي سرويس‌هاي وب است. توسعه‌پذيري در تعاملات ميان اجزا، عموميت واسط ها، توسعه مستقل اجزا و استفاده از واسطه ها از كليدي ترين اهداف معماري REST مي‌باشد و همچنين استفاده از معماري REST در برنامه‌نويسي كارايي، سادگي، انعطاف‌پذيري، امكان مشاهده و نظارت، قابليت حمل و قابليت اطمينان را افزايش مي دهد.

1. Computing [↑](#footnote-ref-1)
2. Grid Computing [↑](#footnote-ref-2)
3. Distributed Computing [↑](#footnote-ref-3)
4. Parallel Computing [↑](#footnote-ref-4)
5. Platform as a Service (PaaS) [↑](#footnote-ref-5)
6. Web Application [↑](#footnote-ref-6)
7. Deployment Environment [↑](#footnote-ref-7)
8. Integrated [↑](#footnote-ref-8)
9. Debugger [↑](#footnote-ref-9)
10. Explore [↑](#footnote-ref-10)
11. Version [↑](#footnote-ref-11)
12. Framework [↑](#footnote-ref-12)
13. Interpreter [↑](#footnote-ref-13)
14. Syntax [↑](#footnote-ref-14)
15. Framework [↑](#footnote-ref-15)
16. Reusability [↑](#footnote-ref-16)
17. Pluggability [↑](#footnote-ref-17)
18. Don't Repeat Yourself [↑](#footnote-ref-18)
19. Database [↑](#footnote-ref-19)
20. Web Server [↑](#footnote-ref-20)
21. Package Management System [↑](#footnote-ref-21)
22. Cryptography [↑](#footnote-ref-22)
23. Security Leaks [↑](#footnote-ref-23)
24. Technical Redundancy [↑](#footnote-ref-24)
25. Query [↑](#footnote-ref-25)
26. Map [↑](#footnote-ref-26)
27. Session [↑](#footnote-ref-27)
28. Site Map [↑](#footnote-ref-28)
29. sandbox [↑](#footnote-ref-29)
30. Object Oriented [↑](#footnote-ref-30)
31. Table [↑](#footnote-ref-31)
32. Map [↑](#footnote-ref-32)
33. Object [↑](#footnote-ref-33)
34. Data [↑](#footnote-ref-34)