مقدمه

با پیشرفت فناوری‌اطلاعات 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 رایانش‌ابری مالک زیرساخت فیزیکی ابر نیستند، بلکه برای اجتناب از هزینه‌ی سرمایه‌ای، تنها آن را از عرضه‌کنندگان اجاره می‌کنند. آنها منابع را در قالب سرویس مصرف می‌کنند و تنها بهای منابعی که به کار می‌برند، را می‌پردازند.

رایانش‌ابری در نمودار دوره محبوبت گروه گارتنر در راس دوره محبوبت قرار دارد، در این مقطع رایانش‌ابری در مرکز توجهات بوده اما هنوز کاملاً پتانسیل‌های خود را آشکار نکرده است. طبق پیش‌بینی‌های گارتنر طی سه تا چهار سال آینده رایانش‌ابری پتانسیل واقعی خود را نمایان می‌کند. رایانش‌ابری بطور چشمگیری موانع ورود به تجارت نرم‌افزاری را کاهش می‌دهد و برای شرکت‌ها روش‌های جدید کسب سود را می‌نمایاند. ارائه‌دهندگان خدمات ابر از طریق تسهیم، بهبود دادن و سرمایه‌گذاری بیشتر در نرم‌افزار و سخت‌افزار، به سود دست می‌یابند. 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- به‌عبارتی 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 پایان‌نامه به بررسی چگونگی ساخت یک ابر از نوع بستر به عنوان سرویس می‌پردازیم. برای مطالعه‌ی این پایان‌نامه، لازم است خواننده با مفاهیم رایانش‌ابری و انواع آن، معماری‌های MVC و REST و تکنیک ORM آشنا باشد. توضیح مختصری در مورد هر یک از این مفاهیم در بخش پیوست‌ها آورده شده است. از خواننده تقاضا می‌کنیم، در صورت آشنا نبودن با این مفاهیم، ابتدا به

بخش پیوست‌ها مراجعه نموده. تا در درک مفاهیم و اصطلاحات به کار رفته در این پایان‌نامه دچار سر درگمی نشود. علاوه بر این موارد آشنایی با زبان python و چارچوب flask و sql و همچنین مبانی طراحی سایت با این زبان لازم است.

این پایان‌نامه در شش فصل تنظیم شده است. در فصل یک به بیان کلیات پروژه مانند اهداف و تکنولوژی‌ها پرداخته‌ایم. فصل دو به تحلیل پایگاه‌داده اختصاص دارد. از آن جا که این پروژه از معماری MVC پیروی می‌کند، هر لایه‌ی این معماری را در یک فصل توضیح داده‌ایم. مدل‌ها را در همان فصل پایگاه داده تشریح می‌کنیم و در دو فصل سه و چهار به بررسی کنترل‌گرها و نماها می‌پردازیم. در فصل پنج به نکات امنیتی به کار رفته در ابر می‌پردازیم و در فصل شش پیشنهاداتی را ارائه کرده‌ایم که می‌توان در آینده به این پروژه اضافه کرد. کدهای کامل برنامه نیز به صورت پیوست در انتهای این پایان‌نامه ضمیمه شده‌اند.

فصل ۱: کلیات

در این فصل نگاهی کلی به پروژه می‌اندازیم و اهداف کلی خود را از پروژه مشخص می‌کنیم و در مورد چگونگی پیاده‌سازی اهداف صحبت خواهیم کرد.

اهداف پروژه

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

هدف نهایی، پیاده‌سازی یک نمونه ابر PaaS بسیار ساده می‌باشد. حداقل موارد لازم برای پیاده‌سازی ابر به شرح زیر است:

محیط توسعه[1]: کدهای وارد شده توسط کاربران در اینجا کامپایل می‌شود.

رابط[2]: وظیفه برقراری ارتباط میان کاربر و محیط توسعه را به عهده دارد.

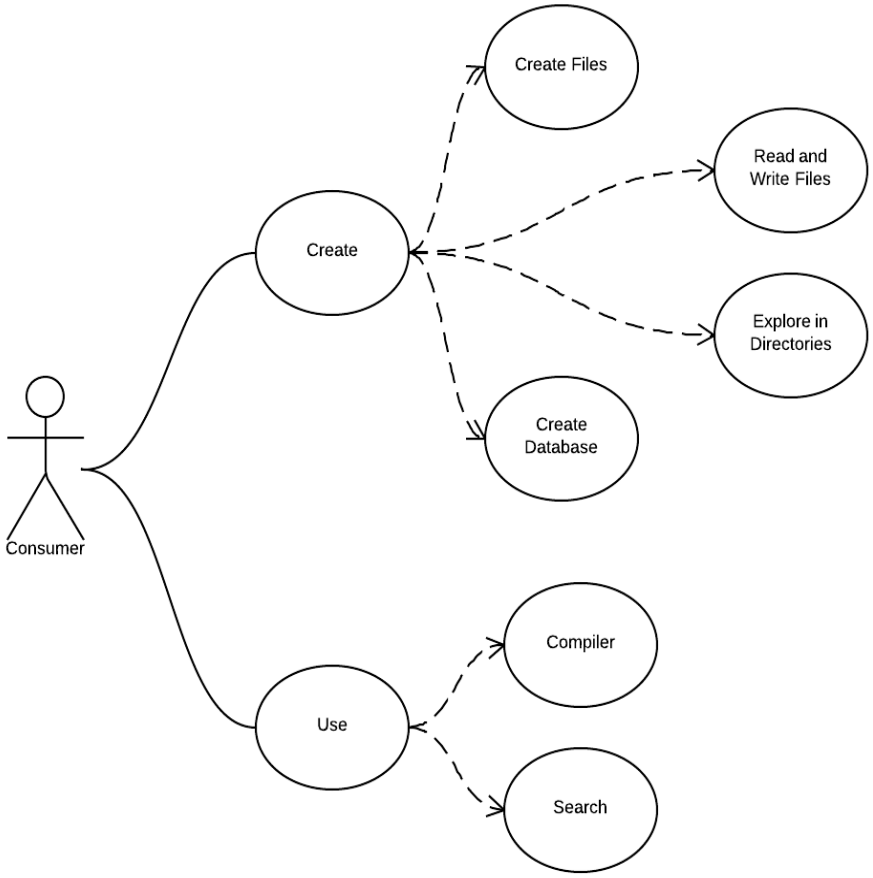
اشکال‌یاب[3]: در صورت وجود مشکل در برنامه آن را به کاربر اطلاع می‌دهد.

انتشار: برای آن که دیگر کاربران بتوانند از برنامه استفاده کنند باید مکانی برای انتشار آن‌ها در نظر بگیریم.

[1] Deployment Environment
[2] Integrated
[3] Debugger

نمودار مورد استفاده (Use Case)

با بررسی اهداف و تجزیه و تحلیل عملکرد ابر PaaS به نمودار زیر، که خواسته‌های کاربر از برنامه را نمایش می‌دهد، می‌رسیم.



تصویر ۱: نمودار حالت استفاده

دو امکان اصلی در اختیار کاربران قرار می‌گیرد. یکی ساخت برنامه و دیگری استفاده از برنامه‌ای که دیگر کاربران ساخته‌اند. هر کدام از این دو امکان، خود از بخش‌هایی تشکیل شده‌اند. برای آن که کاربر بتواند برنامه ایجاد کند، به امکاناتی مانند ساخت فایل، ساخت پایگاه‌داده و امکان خواندن و نوشتن بروی فایل‌ها، که انجام این کارها نیازمند امکان کاوش[۴] نیز بوده، می‌باشد.

[4] Explore

تکنولوژی به کار رفته

	برنامه	نسخه[5]
زبان برنامه نویسی	Python	2.7.3
چارچوب[6]	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 بررسی دلیل انتخاب هر کدام از این برنامه‌ها می‌پردازیم.

زبان برنامه‌نویسی: در حال حاضر سه زبان برنامه‌نویسی PHP، Ruby و Python مطرح‌ترین زبان‌ها در زمینه‌ی طراحی صفحات وب می‌باشند.

کار با PHP به دلیل وجود توابع بسیار زیاد آن راحت و آسان است. اما همین تعداد زیاد توابع باعث سنگین شدن مفسر[7] این زبان شده و سرعت پاسخ‌گویی آن نسبت به دو رقیب خود کاهش قابل توجه‌ای دارد. هرچند در سایت Facebook با استفاده از تکنیکی‌خاص[8] کدهای برنامه را به زبان ماشین کامپایل کرده‌اند و مشکل سرعت به گونه‌ای حل شده است.

---

[5] Version
[6] Framework
[7] Interpreter
[8] HipHop

از آن طرف زبان Ruby با تمرکز بر روی سرعت خود، نام خود را به عنوان سریع‌ترین زبان ثبت کرده است. تعداد ماژول‌ها و کتابخانه‌های آن، نسبت به دو زبان دیگر کمتر بوده و این امر کار با این زبان را کمی دشوار می‌سازد. بهترین مثال برای نمایش قدرت این زبان Twitter است.

زبان Python را می‌توانیم ما بین دو زبان دیگر دانست. سرعت بالایی دارد اما سریع‌ترین نیست و کار با آن راحت است اما آسان‌ترین نیست. سرعت بالا در محاسبات در کنار آسانی طراحی برنامه باعث محبوبیت این زبان شده است. نمونه‌ای عالی از این زبان Youtube است که کاملا با این زبان پیاده‌سازی شده است و در بخش‌هایی از Google نیز از این زبان استفاده کرده‌اند.

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[۹] روان‌تری برخوردار است. که امیدواریم موجب انتقال بهتر مفاهیم شود.

چارچوب[۱۰]: تا به حال چارچوب‌های زیادی برای Python نوشته شده محبوب‌ترین آن‌ها Django و Flask می‌باشند.

Django از معماری MVC پیروی می‌کند و هدف اصلی آن ساخت آسان سایت‌های پیچیده و وابسته به دیتابیس است و بر پایه قابلیت استفاده مجدد[۱۱] و قابل اتصال بودن[۱۲] اجزای مختلف، سریع و اصل خودت تکرار نکن[۱۳] طراحی شده است. برترین مزیت این چارچوب نسبت به سایر

[9] Syntax
[10] Framework
[11] Reusability
[12] Pluggability
[13] Don't Repeat Yourself

چارچوب‌ها بخش ORM آن است که کار با دیتابیس[۱۴] را بسیار ساده کرده و نقطه ضعف آن در این است که توسعه برنامه‌های نوشته شده با Django چندان راحت نیست.

Flask چارچوبی بسیار سبک است که از معماری REST پیروی می‌کند. اغلب Flask را Microframework صدا می‌زنند. زیرا هسته برنامه را ساده نگه می‌دارد. امکاناتی مانند کار با دیتابیس، اعتبارسنجی فرم و از این قبیل امکانات که معمولا در اکثر چارچوب‌های وب قرار دارد، در Flask وجود ندارد. تمامی این امکانات به صورت کامپونت‌های جداگانه موجود است. حتی پیاده‌سازی مدل MVC نیز امکان پذیر است.

پایگاه داده: در پروژه‌هایی مانند این پروژه معمولا از MySQL استفاده می‌شود. ما ابتدا مختصر توضیحی در مورد SQLite داده و سپس به دلیل انتخاب آن می‌پردازیم.

بر خلاف پایگاه‌های داده مستقل نظیر MySQL یا SQL Server، SQLite هیچ فرآیند مستقلی را روی پردازنده اجرا نکرده و تنها کتابخانه اس‌کیوال‌لایت به برنامه اصلی پیوند خورده و با اجرای برنامه اصلی، SQLite هم 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. هنگامی‌که حجم اطلاعات بالا می‌رود کارآیی پایگاه داده کاهش می‌یابد. ما با استفاده از تکنیکی خاص این مشکل را حل کرده‌ایم که در فصل دوم به توضیح کامل آن می‌پردازیم.

[14] Database

وب سرور: چارچوب Flask یک وب سرور[۱۵] کوچک به نام Werkzurg همراه خود نصب کرده و تمام نیازهای ما را در حد این پروژه برآورده می‌سازد. هدف از طراحی این وب سرور سهولت طراحی برنامه بوده و برای پروژه‌های بزرگ گزینه مناسبی نیست. در صورت لزوم می‌توان 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 سایت دبیان[۱۶] قابل مشاهده‌است. پست‌های الکترونیکی ارسالی به فهرست پستی دبیان در کوتاه‌ترین زمان پاسخ داده می‌شوند. برای مشاهده و عضویت در این فهرست‌ها می‌توانید به آدرس[۱۷] مراجعه کنید.

دبیان دارای یکی از بهترین سیستم مدیریت بسته[۱۸]های نرم‌افزاری بوده که این قابلیت به تنهایی برتری دبیان را در برابر دیگر توزیع‌ها اثبات می‌کند. با این ابزار تنومند که APT نام دارد مشکل وابستگی‌های بسته‌های نرم‌افزاری که در توزیع‌های مبتنی بر ردهت[۱۹] دیده می‌شود کاملاً حل شده‌است. با وجود APT انجام ارتقاءهای نرم‌افزاری بسیار تسهیل یافته به‌طوری که با یک فرمان می‌توان سیستم‌عامل را به نسخه جدیدتر ارتقاء داد. دبیان مفهوم یک بار نصب برای همیشه را عملی می‌کند.

---

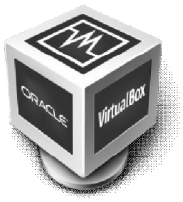
[15] Web Server
[16] www.debian.org/users
[17] www.debian.org/MailingLists
[18] Package Management System
[19] RPM

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

پیگیری اشکالات نرم‌افزارها و حفره‌های امنیتی دبیان در سایت[۲۰] 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[۲۱] برای نشت‌های امن[۲۲]، 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.



مجازی‌ساز: برای شبیه‌سازی محیط سرور از نرم‌افزار مجازی‌ساز ویرچوال‌باکس استفاده کرده‌ایم. هر چند وی‌ام‌ویر امکانات بیشتری داشته و به خصوص در زمینه شبکه از ویرچوال‌باکس پیشرفته‌تر عمل می‌کند، اما ویرچوال‌باکس با محیط ساده باعث سادگی کار با این برنامه شده و حجم کم این نرم‌افزار باعث سبک شدن این برنامه گردیده به طوریکه روی هر سیستمی قابل اجرا بوده و در انتها رایگان بودن آن موجب می‌شود تا به راحتی بتوانیم نسخه اصلی آن را نصب و از آن استفاده کنیم.

---

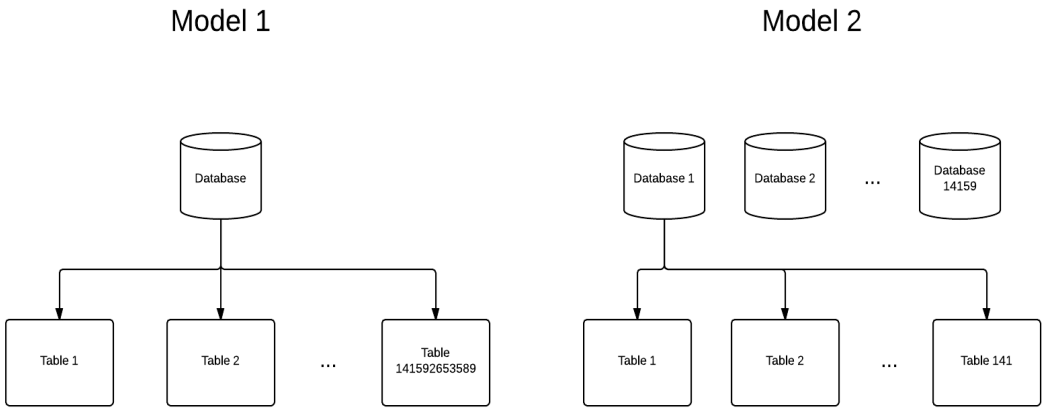
[20] bugs.debian.org
21 Cryptography
[22] Security Leaks

فصل ۲: پایگاه داده

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

انتخاب مدل مناسب

ما در اینجا با روشی پایگاه داده را به قسمت‌های کوچک تقسیم کرده‌ایم تا این مشکل را بر طرف سازیم. به دو مدل پیاده‌سازی پایگاه داده که در زیر آمده است توجه کنید.

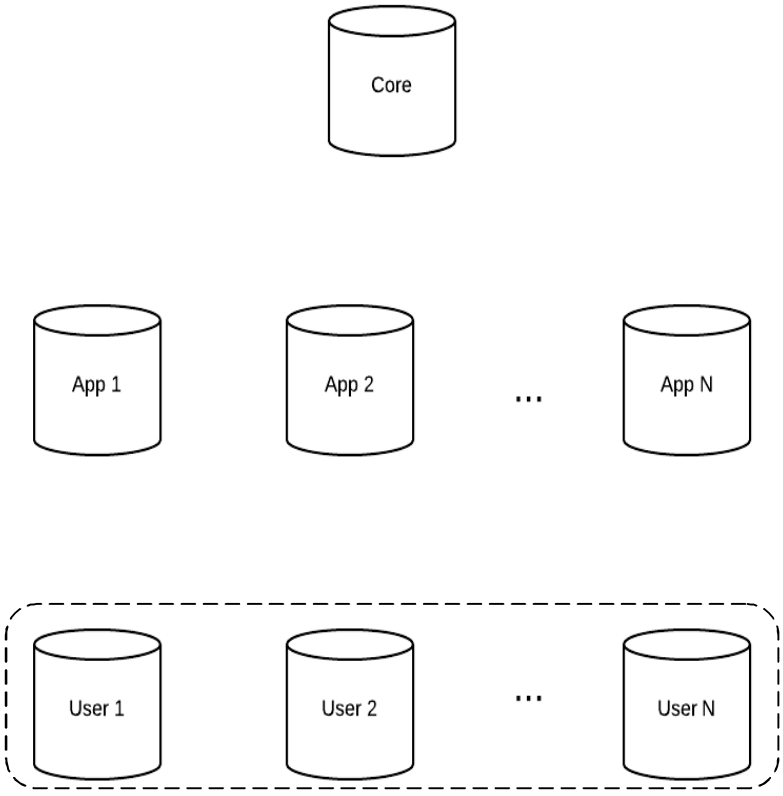


تصویر ۲: انتخاب مدل پایگاه داده

مدل اول حالت ابتدایی و ساده طراحی پایگاه داده بوده که در آن تنها یک پایگاه داده با بی‌شمار جدول داریم. در مدل دوم با تقسیم پایگاه داده به چند قسمت، تعداد جداول پایگاه داده کاهش و در نتیجه حجم پایگاه داده نیز کاهش یافته و مشکل حجم در SQLite را می‌توان به این طریق برطرف کرد.

تقسیم پایگاه داده

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

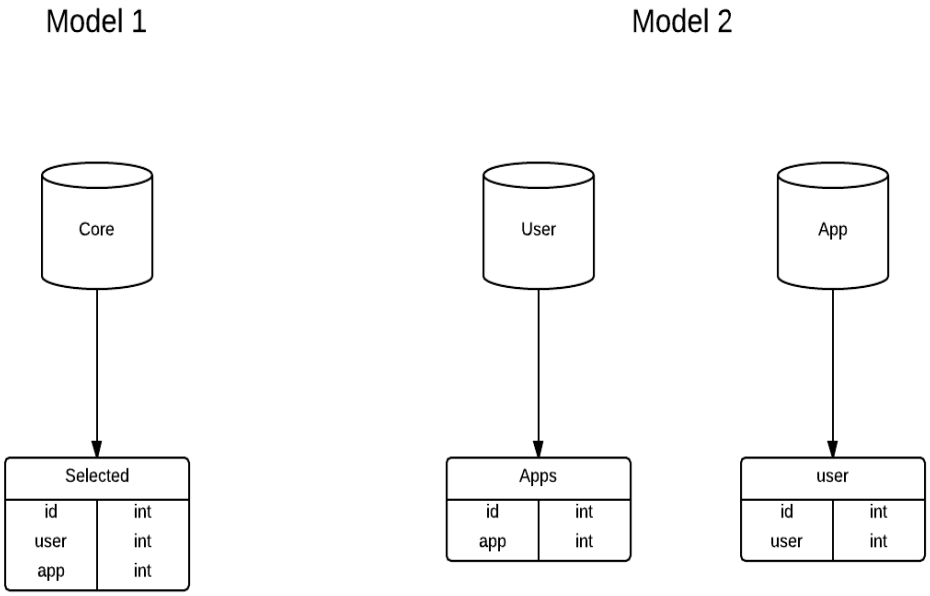


تصویر ۳: نمایه کلی پایگاه داده

در این تقسیم‌بندی ما یک پایگاه داده مرکزی به نام core داشته که هسته برنامه بوده و اطلاعات کلی در آن قرار گرفته و سپس برای هر برنامه‌ای که کاربر ایجاد کرده یک پایگاه داده مجزا تولید می‌کنیم. سپس سوالی پیش می‌آید که آیا لازم است برای هر کاربر نیز پایگاه داده‌ای طراحی کنیم؟ جواب این سوال بستگی به انتخاب مدل پیاده‌سازی ما که در بخش بعدی توضیح خواهیم داد، دارد.

افزونگی تکنیکی[۲۳]

دو مدل زیر را باهم مقایسه می‌کنیم.



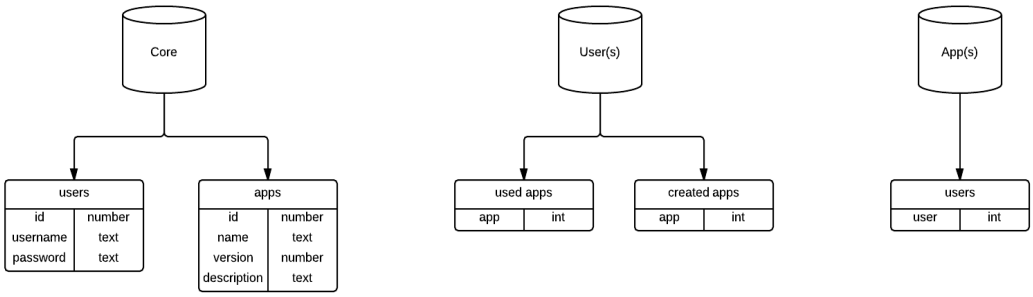
تصویر ۴: انتخاب مدل جدول بندی

در مدل اول تنها یک جدول در پایگاه داده core ایجاد می‌کنیم. از این جدول هم می‌توان برنامه‌های انتخاب شده توسط کاربر را تشخیص و هم می‌توان 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 و حجم اطلاعات ذخیره شده در هر جدول بسیار کاهش می‌یابد. از طرفی چون اطلاعات در دوجای مجزا ذخیره می‌شود، از تعداد پایگاه داده بیشتری استفاده کرده، که باعث به وجود آمدن یک افزونگی و در نتیجه حجم کلی اطلاعات بیشتر خواهد شد. با بالا رفتن داده‌های ذخیره شده سرعت گزارش‌گیری[۲۴] در مدل اول تساعدی افزایش یافته و در مدل دوم افزایش داده تاثیر چندانی در سرعت گزارش‌گیری ندارد. بنابر این مدل دوم بسیار بهتر خواهد بود.

[23] Technical Redundancy
[24] Query

ساختار نهایی پایگاه داده

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



تصویر ۵: ساختار نهایی پایگاه داده

بعد از تحلیل زمان پیاده‌سازی است که برای طراحی از یک ابزار ORM [۲۵] به نام sqlalchemy استفاده می‌کنیم و جدول‌ها را به اشیایی در برنامه مپ[۲۶] می‌کنیم.

پایگاهی برای هر کاربر

برای آنکه بتوانیم برای هر کاربر پایگاهی جدا بسازیم، از سیشنی[۲۷] استفاده می‌کنیم که در هنگام ورود به سایت برای هر کاربر ساخته می‌شود. نمونه کد برنامه مشابه به شکل زیر خواهد بود:

```
from sqlalchemy import create_engine
engine = create_engine(
   'sqlite:///' +
   session['user_id'])
```

در اینجا از متد create_engine کلاس sqlalchemy استفاده کرده‌ایم. با این کد هرگاه کاربری وارد سایت شود، پایگاهی به نام کد کاربری آن فرد ایجاد می‌شود.

[۲۵] برای آشنایی بیشتر به پیوست‌ها مراجعه کنید.

[26] Map
[27] Session

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 داده‌هایی مانند MySQL این مراحل الزامی است. پس هر کاربر برای ایجاد پایگاه داده می‌تواند 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')
```

جداول هسته

users: نام کاربری و رمز عبور تمام کاربران در اینجا ذخیره می‌شود و برای ورود به سایت از این جدول استفاده می‌کنیم. از ذخیره اطلاعاتی شخصی مانند نام، تاریخ تولد و ... به دلیل نداشتن اهمیت برای این پروژه پرهیز کردیم. لیست فیلدها در زیر آمده است:

نام فیلد	نوع فیلد
Id	integer, key, unique, auto
Username	Text
Password	Text

tools: اطلاعات تمام برنامه‌ها را در خود نگاه می‌دارد. از این مدل برای جستجو در برنامه‌ها استفاده می‌کنیم. لیست فیلدها در زیر آورده شده:

نام فیلد	نوع فیلد
Id	integer, key, unique, auto
Name	text, unique

جداول کاربران[۲۸]

در اینجا برای هر کاربر یک پایگاه داده جداگانه ایجاد و در هر کدام جداولی به صورت زیر قرار می‌گیرد.

Selected: این جدول نشان دهنده برنامه‌هایی است که کاربر از آن‌ها استفاده کرده و با یک فیلد نیاز ما را برطرف می‌کند:

نام فیلد	نوع فیلد
Tool	integer, key, unique

Created: این جدول نشان دهنده برنامه‌هایی است که کاربر از آن‌ها ایجاد می‌کند اینجا نیز تنها یک فیلد کافی است.

نام فیلد	نوع فیلد
Tool	integer, key, unique

[28] Users

جداول برنامه‌ها[۲۹]

بسته به نیاز کاربران می‌تواند شامل یک تا بی‌نهایت جدول باشد. ما هیچ جدولی در نظر نمی‌گیریم و کاربران در زمان طراحی برنامه خود می‌توانند جداول را ایجاد کنند.

[29] Tools

فصل ۳: کنترل‌گرها

در این فصل به بررسی کنترل‌گرهای معماری MVC [۳۰] می‌پردازیم. از آنجایی که تقریبا به ازای هر صفحه سایت یک کنترل‌گر لازم است، ابتدا نقشه سایت[۳۱] را ترسیم می‌کنیم.

Gate: try به عنوان validator

صفحه نخست سایت بوده و ثبت‌نام و لاگ‌این در این صفحه انجام می‌شود. همچنین در صورت اینکه کاربر قبلا لاگ‌این کرده باشد، کاربر را به صفحه home منتقل می‌کند. در این کنترلر دو تابع به نام‌های register و login وجود دارد که وظیفه ثبت‌نام و لاگ‌این 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 می‌شود. 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

[۳۰] برای آشنایی بیشتر به پیوست ها مراجعه شود.

[31] Site Map

گردانده و ما از این خطا استفاده و از تکراری بودن فیلد مطلع شده و می‌توانیم خطایی مناسب را به کاربر نمایش دهیم.

در login بعد از چک‌کردن فیلدها در صورت صحیح بودن اطلاعات یک سیشن ساخته می‌شود. 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 می‌گرداند. در زمان استفاده از این تابع، در صورت true بودن مقدار بازگشتی کاربر را به صفحه home هدایت می‌کنیم. برای دیدن کدهای کامل این کنترلر می‌توانید به بخش پیوست‌ها مراجعه کنید.

Home: ارتباط دو جدول از دو پایگاه

صفحه خانه، داشبورد یا کنترل پنل کاربر می‌باشد. که بعد از لاگ‌این کردن، اولین صفحه‌ای است که کاربر با آن روبرو می‌شود. در این صفحه دو لیست وجود دارد. که یکی برنامه‌هایی است که کاربر آن‌ها را ایجاد کرده و دیگری برنامه‌هایی از آن‌ها استفاده می‌کند. این دو لیست از دو جدول select و create در پایگاه مخصوص هر کاربر گرفته می‌شود. از آن‌جا که تنها id برنامه‌ها در این جداول ذخیره می‌شود. ما احتیاج به برقراری ارتباط با جدول tools در پایگاه core را داریم. کدهای 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 جداول پایگاه اول بگیریم و سپس آن‌ها را در جدول دوم جستجو کنیم. در پایین صفحه ناویگیتوری قرار دارد که کاربر را به صفحات دیگر منتقل می‌کند. برای دیدن کدهای این کنترلر می‌توانید به پیوست‌ها مراجعه کنید.

Create: یک تابع بازگشتی

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

این مراحل توسط دو تابع register و clone انجام می‌شود. تابع clone یک تابع بازگشتی است و به صورت زیر عمل می‌کند:

```
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 داشته و تمام فایل‌ها و فولدرها و زیرفولدرها source را در محل dest کپی می‌کند. ابتدا مقایسه می‌کند آیتم یک فایل است یا یک فولدر در صورت فایل بودن آن را ایجاد می‌کند و در صورت فولدر بودن یک فولدر ایجاد کرده و دوباره تابع clone را فراخوانی می‌کند تا تمام زیر پوشه‌ها و زیر فایل‌های آن نیز ایجاد شود. تابع register فقط دو ثبت ساده انجام می‌دهد که احتیاجی به توضیح ندارد. کدهای کامل این کنترلر در بخش پیوست‌ها موجود است.

Tool: مفسر کد برنامه‌ها

این کنترلر بخش اصلی ابر ما بوده و کدهای کاربران را کامپایل می‌کند و نتیجه را به خروجی می‌برد. در ابتدا توسط تابع registered چک می‌کند که آیا این برنامه برای این کاربر ثبت شده است؟ در صورت ثبت نبودن کاربر را به صفحه لایسنس هدایت می‌کند و کاربر بعد از قبول آن اجازه استفاده از نرم‌افزار 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 کدها را کامپایل کرده و خروجی را به کاربر نمایش دهیم. برای این منظور ابتدا یک پرسه جدید ایجاد می‌کنیم. در این پروسه کامپایلر پایتون را فراخوانی می‌کنیم و کدهای نوشته شده توسط کاربر را به کامپایلر ارسال می‌کنیم. نتیجه نهایی را از کامپایلر دریافت و به خروجی می‌بریم. اگر بخواهیم دیاگرامی از مراحل کامپایل رسم کنیم به صورت زیر خواهد بود:

برای نوشتن کدهای این بخش از یکی از ماژول‌های python به نام subprocess استفاده می‌کنیم. Popen برای ایجاد یک پرسه جدید، و PIPE برای دریافت خروجی برنامه استفاده می‌شود. 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 می‌دهد

Explorer و Editor: یک محیط توسعه

برای مدیریت برنامه نیاز به یک explorer است تا بتوانیم به فایل‌ها و فلدرهای برنامه دسترسی داشته باشیم، یا در صورت نیاز یک فایل یا فولدر جدید برای برنامه خود ایجاد کنیم. در کنار این کنترلر دوکنترلر دیگر یکی برای ایجاد فایل و دیگری برای ایجاد فولدر وجود دارد. در زمان لیست کردن، در صورت اینکه آیتم مورد نظر یک فولدر باشد، مسیر آن را به explorer لینک داده و در صورت فایل بودن، آن را به editor لینک می‌کنیم. در editor محتویات فایل نمایش داده می‌شود و پس از ویرایش می‌توانیم آن را ذخیره کنیم.

Tools و Select: دسترسی و مجوز

برای آنکه کاربران به برنامه دیگر کاربران دسترسی داشته باشند، صفحه‌ای به نام tools وجود دارد که تمام برنامه‌های موجود را لیست می‌کند و کاربران با انتخاب یک برنامه ابتدا به صفحه select منتقل می‌شوند. در این صفحه لایسنس برنامه نمایش داده می‌شود. اگر کاربر لایسنس را قبول کند، آن برنامه برای کاربر ثبت و از آن پس کاربر می‌تواند از آن برنامه استفاده کند.

فصل ۴: نماها

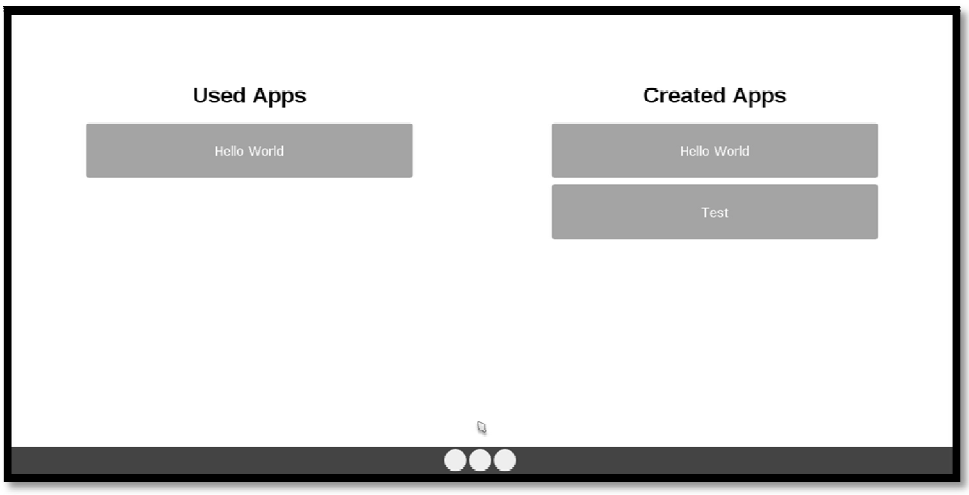
نماها همان صفحات سایت متشکل از کدهای html، css و js می‌باشند. کدهای این صفحات در بخش پیوست‌ها موجود است. در اینجا با قرار دادن یک تصویر از هر صفحه به شرح مختصر وظایف آن می‌پردازیم.

gate: دروازه ورودی سایت

تصویر ۶ صفحه gate

در این صفحه دو دکمه login و register دو عمل لاگ‌این و ثبت‌نام را انجام می‌دهند. برای ثبت‌نام صفحه جدیدی باز نمی‌شود. زمانی که فیلدهای username و password را پر می‌کنیم، در صورت فشردن دکمه register ابتدا اطلاعات را در پایگاه ذخیره می‌کند و سپس عمل لاگین را انجام می‌دهد.

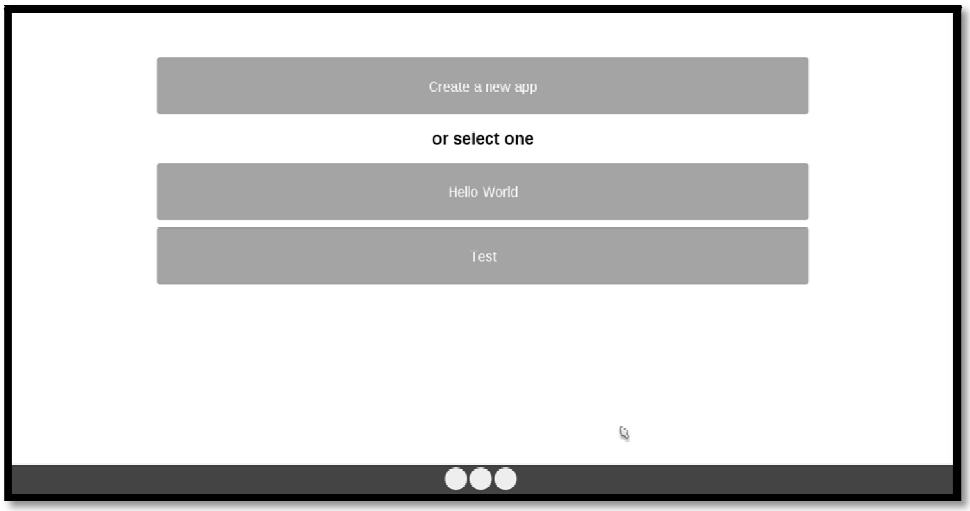
home: داشبورد کنترل برنامه‌ها



تصویر ۷ صفحه home

در این صفحه دو لیست قرار دارد. یکی لیست برنامه‌هایی است که کاربر آن‌ها را ایجاد کرده و دیگری لیست برنامه‌هایی که کاربر از آن‌ها استفاده می‌کند.

tools: لیستی از تمام برنامه‌ها

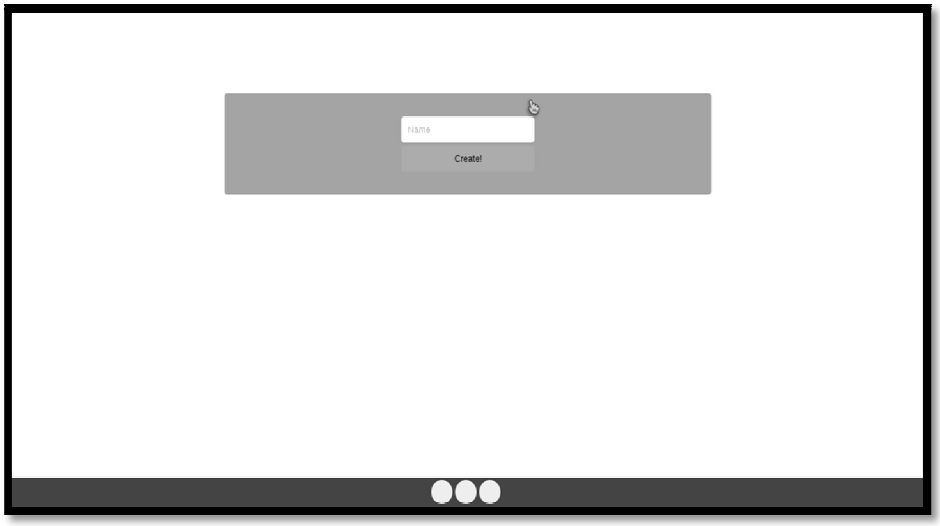


تصویر ۸ صفحه tools

در این صفحه لیستی از تمام برنامه‌های موجود در سرور در اختیار کاربر قرار می‌گیرد و کاربر این امکان دارد تا با کلیک روی نام یکی از آن‌ها از آن برنامه استفاده کند.

علاوه بر این گزینه create a new app به کاربر این امکان را می‌دهد تا یک برنامه جدید ایجاد کند.

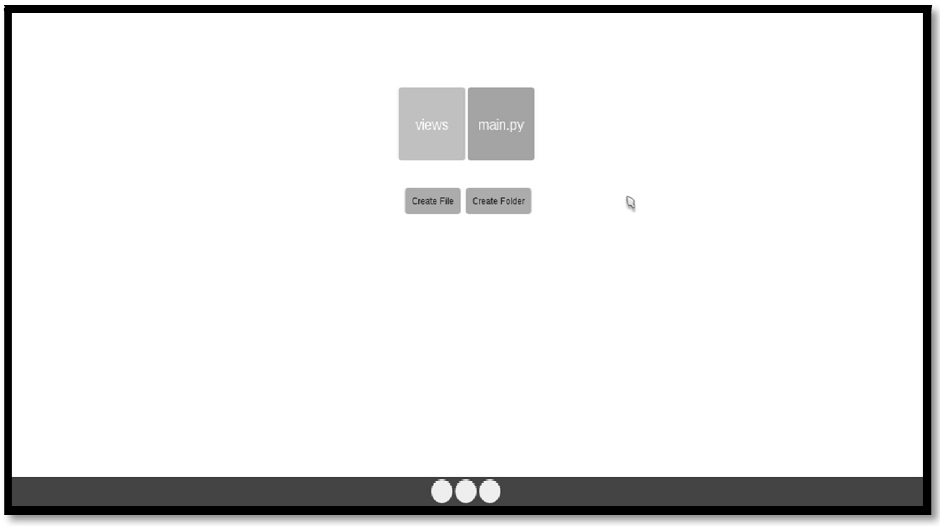
create: ساخت یک برنامه جدید



تصویر ۹ صفحه create

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

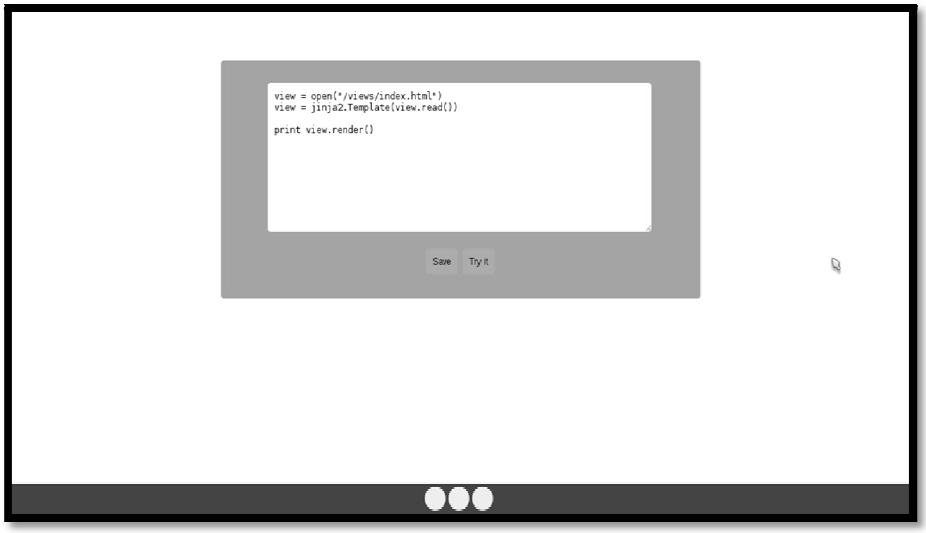
explorer: یک مرورگر ساده



تصویر ۱۰ صفحه explorer

این صفحه یک فایل منیجر ساده است که لیستی از تمام فایل‌ها و فولدرهای برنامه را به کاربر نشان می‌دهد. تفاوت فایل‌ها و فولدرها با رنگ‌های مختلف مشخص می‌شود و دو گزینه creat file و create folder برای ساخت فایل و فولدر استفاده می‌شود.

editor: یک ویرایشگر ساده



تصویر ۱۱ صفحه editor

دراینجا کاربر می‌تواند به ویرایش فایل‌های خود بپردازد. دو گزینه save و try برای ذخیره و امتحان کردن برنامه استفاده می‌شود.

select: انتخاب یک برنامه جدید

تصویر ۱۲ صفحه select

در صفحه به کاربر لایسنس برنامه نمایش داده می‌شود و کاربر پس از پذیرفتن آن به صفحه اصلی برنامه منتقل می‌شود و می‌تواند از برنامه استفاده کند.

فصل ۵: امنیت

بی‌شک مهم‌ترین چالش ابرهای رایانه‌ای امنیت آن‌ها است. وقتی 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 بتوانند به این شرکت اعتماد کنند. هر چند موضوع امنیت ابرها بحث وسیعی است و خود این موضوع به تنهایی عنوان اصلی صدها مقاله بوده، اما ذکر چند نکته امنیتی خالی از لطف نخواهد بود.

جعبه امن[۳۲] (زندان chroot)

هنگامی که به کاربران اجازه اجرای یک کد روی سرور داده می‌شود، یعنی به آن‌ها اجازه انجام هر کاری را داده‌ایم. با یک کد ساده می‌توان 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 هکرها 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 می‌دهیم. در این صورت ما حتما باید چاره‌ای بیاندیشیم تا هر کاربر تنها بتواند در برنامه خود تغییر ایجاد کند و در کار سرور یا در برنامه دیگران اخلالی ایجاد نکند.

خوشبختانه لازم نیست ما همه کار را خودمان انجام دهیم. ابزارهایی برای این منظور ساخته شده. یکی از تکنیک‌های ساده و قدرتمند که در سیستم عامل‌های شبه unix وجود دارد، chroot نام دارد. این ابزار به این صورت عمل می‌کند که با تغییر مسیر ریشه، باعث می‌شود برنامه یا کاربر احساس کند مسیری که هم اکنون 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:

[32] sandbox

```
from os import chroot
chroot(tool_dir)
```

البته اگر این کد را در برنامه اصلی استفاده کنیم، با تغییر مسیر ریشه باعث ایجاد اختلال در کار برنامه خواهد شد. همان‌طور که قبلا گفتیم (به فصل کنترگرها مراجعه شود) برای اجرای برنامه‌ها ما ابتدا یک زیر پروسه مجزا ایجاد می‌کنیم. با اجرای این کد در زیر پرسه ما می‌توانیم این محدودیت را محدود به یک پروسه کنیم. این گونه سایر اجزای برنامه بدون مشکل کار خود را انجام می‌دهند.

محدودیت منابع

در تکنیک قبل ما دسترسی کاربر را به یک پوشه محدود کردیم. اما حال مشکل دیگر وجود دارد. یک کاربر می‌تواند با اجرای یک حلقه بی‌نهایت، پردازنده سرور را اشغال کند. یا ممکن است با ساخت چندین متغیر تمام خانه‌های حافظ را پرکند و با این کار مانع اجرای سایر برنامه‌ها شود و یا حتی ممکن است بتواند در کار سیستم عامل نیز اخلال ایجاد کند.

برای حل این مشکل لازم است پروسه را که وظیفه اجرای برنامه ها را دارد، محدود به استفاده از حد مشخصی از منابع سیستم کنیم. این تکنیک به ما این امکان را می‌دهد تا اگر قصد کسب درآمد از ابر را داریم، با دریافت هزینه‌ای منابع بیشتری در اختیار کاربر قرار دهیم.

ما می‌توانیم محدودیت‌های مختلف را ایجاد کنیم. اما مهم‌ترین محدودیت‌هایی که باید اعمال شوند، محدودیت در زمان استفاده از cpu و محدودیت در استفاده از فضای حافظ ram سرور می‌باشند. برای ایجاد محدودیت منابع در python از ماژول resource استفاده می‌کنیم:

```
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 حسب ثانیه می‌باشد. در این مثال یک برنامه اجازه ندارد بیش از ۴ ثانیه cpu در اختیار بگیرد و RLIMIT_AS برای ایجاد محدودیت در تعداد خانه‌های حافظه‌ای است که یک برنامه می‌تواند در اختیار بگیرد. از آن‌جایی که هر خانه حافظه ۸ بیت یا ۱ بایت می‌باشد. پس می‌توان گفت واحد آن بایت است. در این یک برنامه تنها می‌تواند ۱۰۲۴ بایت معادل ۱ کیلوبایت فضا حافظه استفاده کند.

فصل شش: پیشنهادها و برنامه‌های آینده

این پروژه دارای پتانسیل بالایی است. شاید صدها صفحه دیگر بتوان در مورد این موضوع نوشت. کارهای بسیاری بودند که مایل بودیم روی این پروژه پیاده‌سازی کنیم. اما به علت محدودیت زمان به همین مقدار بسنده کردیم. در این فصل می‌خواهیم تعدادی از این موارد را برشماریم.

پشتیبانی از چند زبان برنامه‌نویسی

در این پروژه تنها نوشتن برنامه با زبان python امکان پذیر است. اما با انجام چند تغیر کوچک می‌توانیم کدهای زبان‌های مختلف را کامپایل کنیم. تنها کافی است در هنگام ثبت برنامه گزینه‌ای جهت انتخاب زبان برنامه قرار دهیم. در زمان کامپایل بجای ارسال کدها به کامپایلر python آن‌ها را به کامپایلر زبانی می‌فرستیم 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 کامپایلر که از داخل جدول برنامه‌ها گرفته شده قرار دارد.

توسعه برنامه‌ها به کمک git

همانطور که می‌دانید git یک ابزار قدرتمند برای توسعه برنامه‌ها است. اکثر ابرهای کنونی git را پشتیبانی می‌کنند. حتی در بعضی از ابرها تنها راه توسعه برنامه git می‌باشد. در این پروژه برای کاربران برای توسعه برنامه خود از یک explorer و یک editor ساده استفاده می‌کنند. که ابزارها بسیاری از نیازهای کاربران را برآورده نخواهند کرد. به عنوان مثال اگر گروهی از کاربران به طور هم‌زمان 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 می‌دهد.

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 می‌کند. 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 می‌دهیم. حال پس از هر با ویرایش برنامه کار باید دکمه به روز رسانی فشار دهد تا برنامه به روز شود.

استفاده از ssh برای ارتباط با سورس کد

یک ابزار دیگر که می‌تواند به توسعه برنامه‌ها کمک کند ssh بوده. در ابرهای مختلفی می‌بینیم که به صورت‌های مختلفی از ssh استفاده می‌کنند. اگر بخواهیم برای هر کاربر یک اکانت ssh بسازیم لازم است در هنگام ثبت‌نام به ازای هر کاربر یک یسر در سرور بسازیم و برای هر کار یک اکانت ssh تعریف کنیم تا از این طریق بتوانند برنامه‌های خود را مدیریت کنند.

ساخت یک زیردامنه برای هر برنامه

ساخت زیردامنه برای برنامه‌ها هیچ تفاوتی در عملکرد آن‌ها ندارد. این کار تنها برای دسترسی آسان‌تر و ساخت url هایی به اصطلاح تمیزتر، به خصوص زمانی که بخواهیم برای برنامه‌ها route

engine تعریف کنیم، به کار می‌رود. استفاده از دامنه متفاوت برای هر برنامه می‌تواند بسیار مفید باشد.

سازگاری با مجوزهای مختلف

در این پروژه در حال حاضر تنها یک مجوز[۳۳] وجود دارد و کاربران حق انتخاب مجوز متفاوتی برای برنامه خود ندارد. می‌توانیم با قراردادن گزینه در زمان ساخت برنامه به کار حق انتخاب مجوز را بدهیم.

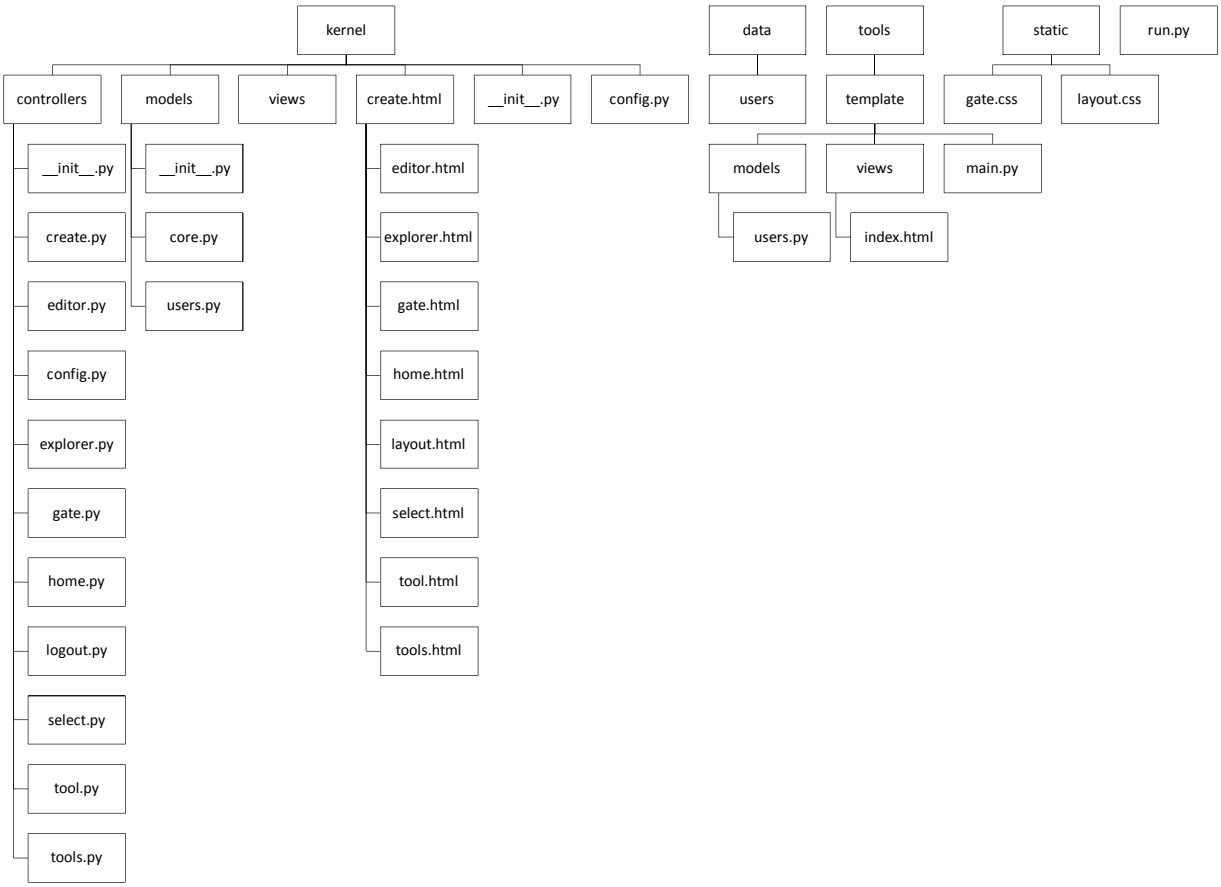
اعمال محدودیت پهنای باند برای هر برنامه

محدودیت در پهنای باند یک ضرورت مهم است. در صورت وجود نداشتن چنین محدودیتی ممکن است یک برنامه تمام پهنای باند را درگیر کند و باعث اختلال در عملکرد کل سیستم بشود. همچنین با باز گذاشتن پهنای باند ما راه را برای هکرها باز گذاشته‌ایم تا بتوانند حملاتی از نوع DoS را به راحتی انجام دهند. علاوه بر این می‌توان قابلیتی فراهم کرد تا کاربران با پرداخت بهایی بتوانند پهنای باند بیشتری دراختیار داشته باشند.

[۳۳] لایسنس

پیوست یک: کد برنامه

ساختار فایل‌های برنامه به صورت زیر می‌باشد:



/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 دو: آشنایی با رایانش‌ابری

رایانش‌ابری، یک مفهوم کاملا جدید نمی‌باشد؛ بلکه از فناوری پیشین رایانش‌توزیع‌یافته در مقیاس بزرگ نشأت گرفته است. با این حال، این تکنولوژی دگرگون‌کننده که سومین انقلاب در صنعت IT به شمار می‌رود، نشان دهنده‌ی رویکرد توسعه‌ای صنعت IT از سخت‌افزار به نرم‌افزار، از نرم‌افزار به سرویس‌ها و از سرویس‌های توزیع‌یافته به سرویس‌های متمرکز می‌باشد. رایانش‌ابری همچنین، شکل جدیدی از رایانش‌تجاری بوده که بطور گسترده در آینده نزدیک مورد استفاده قرار خواهد گرفت. مفهوم اصلی رایانش‌ابری، گاه بار پردازشی بر درگاه‌های 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 می‌سازد. همه اینها از طریق یک اتصال اینترنتی ساده با استفاده از یک مرورگر استاندارد یا سایر اتصالات میسر می‌شود. با این حال، هنوز هم مشکلات فراوانی در رایانش‌ابری کنونی وجود دارد؛ یک تحقیق تازه نشان می‌دهد که امنیت داده و خطرات حریم خصوصی، به نگرانی اصلی برای مردمی که می‌خواهند به رایانش‌ابری متوسل شوند، تبدیل شده است.

رایانش‌ابری چیست؟

ابر یک ائتلاف مشترک از منابع‌رایانه‌ای است که می‌تواند:

- انواعی از بارهای کاری مختلف از جمله دسته‌ای از عملیات پشت‌زمینه‌ای و برنامه‌های تعاملی کاربرمحور را مدیریت کند.
- به سرعت بار کاری را با استفاده از ارائه سریع ماشین‌های فیزیکی یا ماشین‌های مجازی، آرایش داده و افزایش دهد.

- از تکرار اطلاعات میان فایل‌های گوناگون، خودترمیمی و مدل برنامه‌نویسی شدیدا مقیاس‌پذیر، بگونه‌ای که بتوان از خرابی‌های سخت‌افزاری/ نرم‌افزاری غیرقابل اجتناب رهایی یافت، پشتیبانی می‌کند،
- بصورت بلادرنگ بر استفاده از منابع نظارت کرده و تخصیص منابع در هنگام نیاز را مجددا متوازن کند.

مدل سرویس

نرم‌افزار به‌عنوان‌یک‌سرویس[۳۴]: نرم‌افزار به‌عنوان‌یک‌سرویس، نرم‌افزاری است که در اینترنت گسترش یافته تا پشت دیوار آتشین در شبکه محلی یا رایانه‌ی شخصی شما اجرا شود. این یک مدل، «پرداخت آنی» است و در اصل بطور گسترده برای اتوماسیون نیروی فروش و مدیریت رابطه مشتریان[۳۵] بکار گرفته می‌شود.

پلتفرم به‌عنوان‌یک‌سرویس[۳۶]: پلتفرم به‌عنوان‌یک‌سرویس، ضمن این‌که یک نوع SaaS بوده، نوعی رایانش‌ابری است که محیط توسعه را به عنوان یک سرویس فراهم می‌کند و می‌توانید از تجهیزات واسطه برای توسعه‌ی برنامه خود و انتقال آن به کاربران از طریق اینترنت یا سرورها استفاده کنید.

زیرساخت به‌عنوان‌یک‌سرویس[۳۷]: زیرساخت به‌عنوان‌یک‌سرویس یک محیط مجازی‌سازی پلتفرم به‌عنوان‌یک‌سرویس است. بجای خرید سرور، نرم‌افزار، فضای پایگاه داده یا تجهیزات شبکه، مشتریان آن منابع را به عنوان یک سرویس کامل از جانب منابع خارجی می‌خرند.

سخت‌افزار به‌عنوان‌یک‌سرویس[۳۸]: مطابق با نظر نیکولاس کار -ایده خرید سخت‌افزار IT یا حتی یک پایگاه داده کامل به عنوان بخشی از خدمات اشتراک پرداخت آنی است که به منظور

---

[34] **SaaS**
[35] CRM
[36] **PaaS**
[37] **IaaS**

برآورده کردن نیازهایتان، مقیاس آن افزایش یا کاهش یافته و در نتیجه پیشرفت‌های سریع در مجازی‌سازی سخت‌افزاری، اتوماسیون IT و سنجش و قیمت‌گذاری مصرف، تصور می‌کنم که مفهوم سخت‌افزار به‌عنوان‌یک‌سرویس ممکن است در نهایت برای زمان آغازین آماده باشد.- این مدل برای کاربران تجاری به خاطر عدم نیاز به سرمایه‌گذاری در ساخت و مدیریت پایگاه‌های داده. مفید است.

مدل آرایش و گسترش

ابر عمومی: در ابرهای عمومی، مشتریان متعدد منابع رایانه‌ای ارائه شده توسط یک سرویس دهنده واحد را به اشتراک می‌گذارند، مشتریان می‌توانند به سرعت به این منابع دسترسی یافته 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[۳۹] وجود دارد.

ابر خصوصی: در ابر خصوصی، منابع رایانه‌ای توسط یک شرکت تجاری خصوصی مورد استفاده و نظارت قرار می‌گیرد. این ابر معمولا در پایگاه داده شرکت استفاده شده و توسط پرسنل داخلی یا سرویس دهنده مدیریت می‌شود. مهم‌ترین مزیت این مدل آن است که امنیت، سازگاری و کیفیت سرویس تحت کنترل و نظارت شرکت‌های تجاری است.

ابر مرکب[۴۰]: نوع سوم می‌تواند ابر مرکب باشد که ترکیب نوعی از ابر عمومی و خصوصی است. این ابر، شرکت تجاری را قادر می‌سازد تا بار کاری حالت پایداری در ابر خصوصی اجرا کرده و زمانی که بار کاری ماکزیمم رخ دهد، از ابر عمومی تقاضای منابع محاسباتی گسترده کرده و سپس در صورتی‌که بیش از آن نیاز نباشد، باز می‌گردد.

ابر انجمنی: سازمان‌های متعددی بطور مشترک، یک زیرساخت ابری یکسان ساخته و به همراه مواردی نظیر حریم خصوصی، مقررات، ارزش‌ها و ملاحظات به اشتراک می‌گذارند. انجمن ابری

[38] HaaS
[39] QoS
[40] Hibrid

به درجه‌ای از مقیاس‌پذیری و تعادل دموکراتیک می‌رسد. زیرساخت ابری این مدل توسط یک فروشنده ثالث، یا در محدوده یکی از سازمان‌های داخل انجمن میزبانی می‌شود.

امور مربوط به رایانش‌ابری

در چند سال اخیر، رایانش‌ابری از یک ایده تجاری امیدبخش به یکی از سریع‌ترین بخش‌های در حال رشد از صنعت IT تبدیل شده است. حال، کمپانی‌های آسیب خورده از بحران اقتصادی، روزبه‌روز بیشتر در حال درک این واقعیت هستند که تنها با بهره‌گیری از «ابرها» می‌توانند موفق به دسترسی سریع به برنامه‌های تجاری بهترین مدل شده و منابع زیرساختی خود را بیشتر تقویت و همه اینها را با هزینه‌ای ناچیز انجام دهند. اما هرچه اطلاعات بیشتر و بیشتری از افراد و شرکت‌ها در ابر قرار می‌گیرد، نگرانی‌ها درباره این که یک محیط تا چه حد ایمن است، آغاز می‌شود.

امنیت

آیا داده‌های شما در هارد درایو شبکه محلی‌تان و یا در سرورهای بسیار ایمن موجود در ابر از امنیت بیشتری برخوردار خواهند بود؟ برخی افراد ادعا کرده، داده‌های مشتریان هنگامی که بصورت داخلی مدیریت شوند، ایمن‌تر است. در حالی‌که برخی دیگر مدعی هستند که تأمین‌کنندگان ابر از انگیزه قوی برای حفظ اعتماد برخوردار بوده و بدین ترتیب از سطح امنیتی بالاتری استفاده می‌کنند. با این حال، در ابر، فارغ از اینکه آرشیو داده اصلی شما نهایتا در کجا ذخیره شده، داده‌های شما در میان رایانه‌های فردی توزیع می‌شود. هکرهای زبردست و کوشا در عمل توانایی تهاجم به هر سروری را داشته و آماری وجود دارد که نشان می‌دهد یک سوم رخنه‌های امنیتی ناشی از رایانه‌های قابل حمل و دیگر دستگاه‌هایی که به سرقت رفته یا گم شده‌اند یا ناشی از این بوده که کارمندان بطور تصادفی داده‌ها را در تماس با اینترنت قرار داده و تقریبا ۱۶٪ نیز بدلیل سرقت داخلی بوده است.

حریم خصوصی

رایانش‌ابری که متفاوت از مدل رایانش سنتی است، از تکنولوژی رایانش‌مجازی بهره برده و داده‌های شخصی کاربران ممکن است بجای آنکه در همان موقعیت فیزیکی یکسان بماند، در پایگاه‌های داده مجازی متعدد پخش شده و یا حتی به آن سوی مرزهای کشور برود که در همین زمان، حفاظت از حریم خصوصی داده‌ها با مناقشه بر سر سیستم‌های حقوقی مختلف مواجه خواهد شد. از سوی دیگر ممکن است کاربران در هنگام دسترسی به سرویس‌های رایانش‌ابری منجر به نشت اطلاعات شوند. مهاجمان می‌توانند وظیفه بحرانی را بر حسب وظیفه رایانه‌ای 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 زمان‌های از کار افتادگی و افت سرعت‌هایی را تجربه می‌کنند که فرق آن‌ها در این است که در مدل رایانش‌ابری، کاربران وابستگی بیشتری به یک تأمین کننده سرور ابری[۴۱] دارند. هنگامی که یک CSP خاص را انتخاب می‌کنید، تفاوت‌های زیادی بین مدل سرویس CSP وجود دارد که ممکن است اسیر آن شوید و خطر امنیتی بالقوه‌ای برایتان به همراه داشته باشد.

مسائل حقوقی

فارغ از تلاش‌های صورت گرفته به منظور به جریان انداختن موقعیت قانونی، از سال ۲۰۰۹، تهیه‌کنندگانی نظیر سرویس‌های وب آمازون، با توسعه جاده محصور و شبکه ریلی و دادن اجازه انتخاب مناطق قابل دسترس به کاربران، به بازارهای بزرگی دست یافته‌اند. از سوی دیگر، نگرانی‌های درباره تدابیر امنیتی و محرمانه بودن اطلاعات از سوی فرد در تمامی سطوح قانونی وجود دارد.

[41] CSP

استاندارد باز

استانداردهای باز برای پیشرفت رایانش‌ابری ضروری هستند. اکثر تهیه‌کنندگان ابری، API هایی را نمایش می‌دهند که عموما بخوبی مستند می‌گردند ولی همچنین در اجرا نیز منحصر بفرد هستند و بدین ترتیب قادر به کار در سیستم‌های مختلف هستند. برخی از فروشندگان از API های دیگری استفاده می‌کنند و تعدادی استاندارد باز تحت ساخت نیز همانند واسط رایانش‌ابری باز OGF وجود دارد. کنسرسیوم ابر باز[۴۲] مشغول فعالیت به منظور ایجاد توافق عمومی درباره استانداردها و تکنیک‌های اولیه رایانش ابری است.

سازگاری

مقررات متعددی که به ذخیره‌سازی و استفاده از داده‌ها مربوط می‌شود مستلزم گزارش‌دهی منظم و پیگیری است، تهیه‌کنندگان ابر می‌بایست 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 می‌شود که نگاه بالا به پایین تمامی منابع IT در یک موقعیت ابرمحور چگونه می‌تواند مدیریت و اعمال قانون قویتری در سیاست‌های سازگاری داشته باشد. علاوه بر مقرراتی که مشتریان را مخاطب قرار می‌دهد، مراکز داده تحت حمایت تهیه‌کنندگان ابر نیز ممکن است در معرض الزامات سازگاری باشند.

آزادی

رایانش‌ابری به کاربران اجازه برخورداری فیزیکی از ذخیره‌سازی داده نمی‌دهد و کنترل و ذخیره‌سازی داده را در دستان تهیه‌کنندگان ابر قرار می‌دهد. مشتریان ادعا می‌کنند که این مسأله‌ای کاملا بنیادی است که به آنها توانایی حفظ نسخه‌های خود از داده‌ها را به شکلی که آزادی انتخاب ایشان حفظ شده و آنها را در برابر برخی مسائل خاص خارج از کنترل آنها حفاظت کند، می‌دهد در حالی‌که مزایای بی اندازه زیاد رایانش ابری را نیز به همراه دارد.

---

[42] OCC

امکان دوام بلند مدت

شما می‌بایست اطمینان داشته باشید که داده‌هایی که در ابر قرار می‌دهید هرگز باطل و بی‌اعتبار نمی‌شوند، حتی اگر تهیه کننده ابر شما با شکست مواجه شده یا شرکتی بزرگتر، آن را به چنگ آورده و در خود ادغام کند. گارتنر می‌گوید «از تهیه‌کنندگان احتمالی بپرسید که چنانچه داده‌هایتان در قالبی باشد که بتوانید آن را به یک برنامه جایگزینی وارد کنید، چگونه آنها را دریافت خواهید کرد».

راه حل

به منظور پیشرفت رایانش‌ابری، جامعه می‌بایست تدابیری پیشگیرانه اتخاذ کرده تا امنیت آن تضمین شود. راه حل مقاله برکلی، رمزدارکردن داده‌ها است. پیش از ذخیره‌سازی داده در موقعیت مجازی، آن را با قفل‌های خود رمزدار کرده و اطمینان بیابید که فروشنده برای ارائه گواهی‌های امنیتی و ممیزی‌های خارجی آماده است. مدیریت هویت، کنترل دسترسی، گزارش‌دهی وقایع امنیتی، مدیریت لایه فیزیکی و پرسنل می‌بایست پیش از انتخاب یک CSP مورد ارزیابی قرار گیرند و می‌بایست اطلاعات شخصی فرستاده شده به ابر و ذخیره شده در آن را به حداقل برسانید.

CSP می‌بایست کنترل کاربر را به بیشترین میزان رسانده و بازخورد ارائه دهد. سازمان‌ها می‌بایست برنامه‌ها و انتقال داده را در ابر خصوصی خود اجرا کرده و سپس آن را به ابری عمومی تغییر شکل دهند. اگرچه مسائل حقوقی زیادی در ارتباط با رایانش‌ابری وجود دارد، پیمان امنیت ابری می‌بایست به سرعت استانداردهای مرتبطی طراحی کند.

پیوست سه: 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 کدنویسی 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 می‌کرد، کدهایی نوشته شده قابل استفاده مجدد بودند و مهم‌ترین قابلیت این بود که یک طراح و یک برنامه‌نویس می‌توانستند روی یک پروژه در آن واحد کار کنند بدون اینکه اختلالی در کار هم به وجود بیاورند. با فراگیرتر شدن وب برنامه‌نویسان وب تصمیم به استفاده از این معماری در طراحی برنامه‌های خود گرفتند. معماری سه لایه از سه قسمت Model، View، Controller تشکیل می‌شود. که در زیر به شرح مختصری در مورد هر کدام از لایه‌ها می‌پردازیم.

لایه Model

این لایه برای ارتباط با دیتابیس استفاده می‌شود. این لایه باید اجازه دسترسی، تغییر یا اضافه کردن داده‌ها را بدهد. این لایه در واقع یک پل بین لایه View و لایه Controller می‌باشد. یکی از مهم‌ترین خاصیت‌های این لایه این است که "نابیناست" به این معنی که مدل نمی‌داند وقتی که داده‌ها را به View یا کنترلر ارسال کرد قرار است چه اتفاقی برای آن‌ها بیافتد و به دنبال پاسخی از Controller یا View نیست. تنها هدفش این است که داده‌ها را ذخیره کند یا زمانی که درخواستی از بقیه لایه‌ها ارسال شد تغییرشان بدهد.

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 باشد ولی چیزی که در مورد این لایه مهم است این است که این لایه مسئول ارتباط با دیتابیس است. حذف، اضافه و ویرایش داده‌های دیتابیس توی این لایه انجام می‌شود.

لایه View

این لایه جایی که داده‌ها از مدل گرفته می‌شوند و به صورت خروجی به کاربر نمایش داده می‌شوند. در برنامه‌های وب این لایه جایی که کدهای HTML ساخته و نمایش داده می‌شوند. توی این لایه ارتباط با کاربر انجام می‌شود و با کنترلر درخواست کاربر را انجام می‌دهد. به عنوان مثال یک button را درنظر بگیرید که وقتی روی آن کلیک شد یکی از action های کنترلر را صدا می‌زند.

یک سری تصورات غلط در مورد این لایه در برنامه‌نویسان وب وجود دارد. یکی از این تصورات اشتباه این است که لایه View نباید هیچ ارتباطی با لایه Model داشته باشد و باید همه اطلاعات را از لایه Controller بگیرد. مثلا در فریم‌ورک Cakephp و اکثر فریم‌ورک‌های PHP این اشتباه وجود دارد. در واقع این تفکر نادیده گرفتن کامل تئوری پشت معماری سه لایه است.

این نکته خیلی مهم است که توجه داشته باشید برای پیاده‌سازی درست معماری MVC لایه View نباید با لایه Model ارتباط داشته باشد و منطق برنامه باید فقط در لایه Controller انجام شود.

از آنجا که بین علما اختلاف نظر وجود دارد تحقیق بیشتری کردم. به نقل از گروه Gang of four در مورد معماری MVC اینطور نوشته:

MVC شامل مدل‌ها و ویوهایی است که می‌توانند با هم ارتباط داشته باشند. یک View باید مطمئن باشد که ظاهری که الان می‌خواهد به خودش بگیرد باید حالتی از model باشد. هر وقت که دیتا در لایه model تغییر کرد یک پیغام به ویو می‌فرستد که بسته به آن تغییر کند. موضوع مهم این است که این لایه وظیفه نمایش داده‌ها و گرفتن و ارسال آن‌ها را به لایه Controller دارد. ما هم فریم‌ورک را آن طوری‌که مایلیم می‌نویسم.

لایه Controller

این لایه منطق برنامه را کنترل می‌کند. این لایه ورودی‌ها را می‌گیرد و درخواست‌های کاربر را انجام می‌دهد، اگر نیاز باشد از دیتابیس مقادیری برای کاربر ارسال شود را از لایه Model می‌گیرد و به لایه View ارسال می‌کند.

به هر حال در باره کارکرد این سه لایه باهم نقل قول‌های زیادی شده که هر کسی هرجور خواسته با آن برخورد کرده است. شما با هر کدام راحت‌ترید کار کنید ولی این را بدانید که اصل موضوع چیست.

پیوست چهار: آشنایی با تکنیک ORM

ORM [۴۳] یک تکنیک برنامه‌نویسی برای تبدیل ارتباطات در دیتابیس به مفاهیم شی‌گرایی[۴۴] در برنامه‌نویسی است. در واقع می‌توان گفت که ORM کلاس‌ها را به جدول[۴۵] ها مپ[۴۶] می‌کند.

ORM برای این کار، یک Framework تهیه و توصیه می‌کند که کاربرها هنگام پیاده‌سازی 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[۴۷] 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[۴۸] در برنامه‌نویسی شی‌گرا 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.

---

[43] Object Relational Mapping
[44] Object Oriented
[45] Table
[46] Map
[47] Object
[48] Data

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 فواید استفاده از ORM ها خواهیم پرداخت.

کاهش زمان تحویل پروژه

اولین و مهم‌ترین دلیلی که بر اساس آن در یک پروژه، استفاده از ORM حائز اهمیت می‌شود، بحث بالابردن سرعت برنامه‌نویسی و کاهش زمان تحویل پروژه به مشتری است. این کاهش زمان بسته به نوع پروژه بین ۲۰ تا ۵۰ درصد می‌تواند خود را بروز دهد.

بدیهی است ابزارهای ORM کار شگفت‌انگیزی را قرار نیست انجام دهند و شما می‌توانید تمام آن عملیات را دستی هم به پایان رسانید؛ اما اجازه دهید یک مثال کوتاه را با هم مرور کنیم.

برای پیاده‌سازی یک برنامه متداول با حدود ۱۵ تا ۲۰ جدول، حدودا به ۳۰ شیء برای مدل‌سازی سیستم نیاز خواهد بود و برنامه‌نویسی این مجموعه بین ۵۰۰۰ تا ۱۰۰۰۰ سطر کد را به خود اختصاص خواهد داد. بدیهی است برنامه‌نویسی و آزمایش این سیستم چندین هفته یا ماه به طول خواهد انجامید.

اما با استفاده از یک ORM ، عمده وقت شما به طراحی سیستم و ایجاد ارتباطات بین اشیاء و دیتابیس در طی یک تا دو روز صرف خواهد شد. ایجاد کد بر اساس این مجموعه و با کمک ابزارهای ORM، آنی است و با چند کلیک صورت می‌گیرد.

کدی با طراحی بهتر

ممکن است شما بگوئید که کدنویسی من بی‌نظیر است و از من بهتر کسی را نمی‌توانید پیدا کنید! به تمامی زوایای کار خود مسلطم و نیازی هم به این گونه ابزارها ندارم!

عده‌ای از شما اما نه همه، به طور قطع اینگونه‌اید. در یک تیم متوسط، همه نوع برنامه‌نویس با سطوح مختلفی را می‌توانید پیدا کنید و تمامی آن‌ها برنامه‌نویس‌ها و یا طراح‌های آنچنان قابلی هم نیستند. بنابراین امکان رسیدن به کدهایی که مطابق اصول دقیق برنامه نویسی شیء گرا نیستند و در آن‌ها الگوهای طراحی به خوبی رعایت نشده، بسیار محتمل است. همچنین در یک تیم، زمانی‌که از یک الگوی یکسان پیروی نمی‌شود، نتایج نهایی بسیار ناهماهنگ خواهند بود. در مقابل استفاده از ORM های طراحی شده توسط برنامه‌نویس‌های قابل[۴۹]، کدهایی را براساس الگوهای استاندارد و پذیرفته شده‌ی شیءگرا تولید می‌کنند و همواره یک روندکاری مشخص و هماهنگ را در یک مجموعه به ارمغان خواهند آورد.

نیاز به تخصص کم‌تر

قسمت دسترسی به داده‌ها یکی از اجزای کلیدی کارآیی برنامه شما است. اگر طراحی و پیاده‌سازی آن ضعیف باشد، کل برنامه را زیر سؤال خواهد برد. برای طراحی و پیاده‌سازی دستی این قسمت از کار باید به قسمت‌های بسیاری از مجموعه‌ی دات‌نت فریم‌ورک مسلط بود. اما هنگام استفاده از یک ORM مهم‌ترین موردی را که باید به آن تمرکز نمائید بحث طراحی منطقی کار و ایجاد روابط بین اشیاء و دیتابیس و امثال آن است. مابقی موارد توسط ORM انجام خواهد شد و همچنین می‌توان مطمئن بود که پیاده‌سازی خودکار انجام شده این قسمت‌ها، بر اساس الگوهای طراحی شیءگرا است.

[49] senior (architect level) engineers

کاهش زمان آزمایش

بدیهی است اگر قسمت دسترسی به داده‌ها را خودتان طراحی و پیاده‌سازی کرده باشید، زمان قابل توجهی را نیز باید به بررسی و آزمایش صحت عملکرد آن بپردازید و الزامی هم ندارد که این پیاده‌سازی مطابق بهترین تجربیات کاری موجود بوده باشد. اما هنگام استفاده از کدهای تولید شده توسط یک ابزار ORM می‌توان مطمئن بود که کدهای تولیدی آن که بر اساس یک سری الگوی ویژه تولید می‌شوند، کاملا آزمایش شده هستند و همچنین صدها و یا هزارها نفر در دنیا هم اکنون دارند از این پایه در پروژه‌های موفق خود استفاده می‌کنند و همچنین بازخوردهای خود را نیز به تیم برنامه‌نویسی آن ابزار ORM ارائه می‌دهند و این مجموعه مرتبا در حال بهبود و به روز شدن است.

پیوست پنج: آشنایی معماری REST[50]

REST یک مدل معماری برای طراحی برنامه‌های کاربردی شبکه است که ترکیبی از چند مدل معماری مبتنی بر شبکه می‌باشد و محدودیت‌هایی جهت تعریف یک واسط اتصال یکنواخت برای آن درنظر گرفته شده است.

REST متکی بر یک پروتکل ارتباطی بدون حالت، کلاینت سرور و با قابلیت cache کردن می‌باشد که در اکثر موارد پروتکل HTTP مورد استفاده قرار می‌گیرد.

ایده اصلی معماری REST این است که به جای استفاده از مکانیزم‌های پیچیده‌ای مانند CORBA، RPC یا SOAP برای اتصال ماشین‌ها ازHTTP ساده برای برقراری ارتباط بین ماشین‌ها استفاده شود.

مدل REST شش قید برای معماری برنامه‌های شبکه تعریف می‌کند:

- Client-Server: کلاینت سرور باشد.
- Stateless: بدون حالت باشد.
- Cacheable: قابلیت cache داشته باشد.
- Layered System: سیستم لایه‌بندی شده داشته باشد.
- Uniform Interface: واسط یکنواخت داشته باشد.
- Code on Demand: دارای قابلیت کد در صورت نیاز باشد. (که این محدودیت اختیاری می‌باشد.)

به سیستمی که این قیود را رعایت نماید، RESTful می‌گویند. از لحاظ رویکرد برنامه‌نویسی REST جایگزینی ساده برای سرویس‌های وب است. توسعه‌پذیری در تعاملات میان اجزاء، عمومیت واسط‌ها، توسعه مستقل اجزا و استفاده از واسطه‌ها از کلیدی‌ترین اهداف معماری REST می‌باشد و

[50] Representational State Transfer

همچنین استفاده از معماری REST در برنامه‌نویسی کارایی، سادگی، انعطاف‌پذیری، امکان مشاهده و نظارت، قابلیت حمل و قابلیت اطمینان را افزایش می‌دهد.

منابع

http://en.wikipedia.org/wiki/Cloud_computing
http://en.wikipedia.org/wiki/Platform_as_a_service
https://developers.google.com/appengine/training/intro/whatiscc
http://dcvizcayno.wordpress.com/2012/04/13/cloud-computingtip/finaial industy /tips-for-financial-industry /
http://www.forbes.com/sites/emc/2014/02/03/cloud-computingfor-the-enterprise-its-about-service
http://bestclipartblog.com/25-cloud-clip-art.html/cloud-clip-art-6
http://en.wikipedia.org/wiki/Model%E2%80% 93view%E2%80%93controller
http://www.servebox.org/actionscriptfoundry/actionscript-foundrydocumentation/a-mvc-framework
https://www.lucidchart.com
http://www.publicpolicy.telefonica.com/blogs/blog/2012/11/20/making-europe-cloudactive
http://en.wikipedia.org/wiki/Platform_as_a_service
http://en.wikipedia.org/wiki/Debian
http://en.wikipedia.org/wiki/Nginx
http://en.wikipedia.org/wiki/Python_(programming_language)
http://en.wikipedia.org/wiki/SQLite
http://en.wikipedia.org/wiki/Flask_(web_framework)
http://www.aperfectworld.org/weather.html