به نام خدا

گزارش تسک عملی

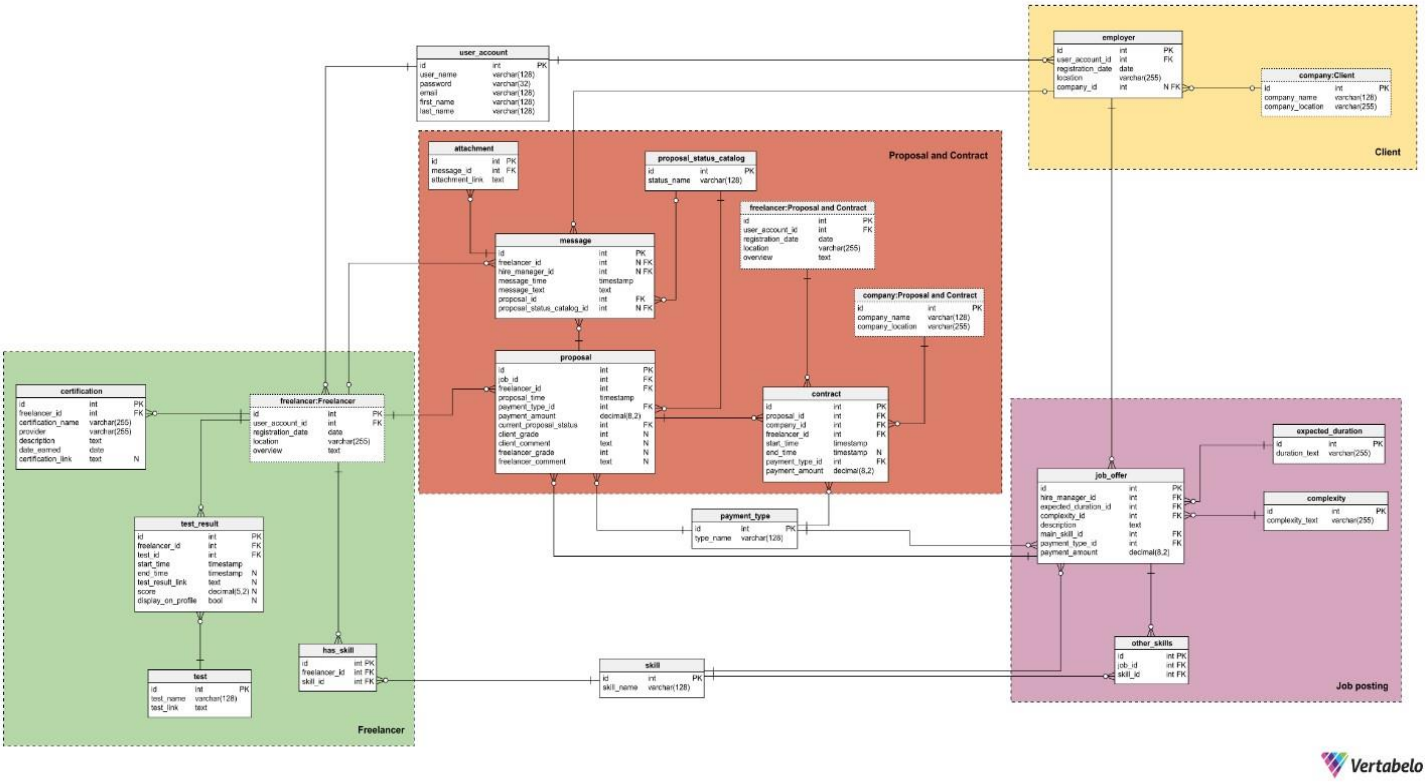
موقعیت شغلی Software Engineer

click to redirect to repository

Jobac (Jobac organization of business and contract)

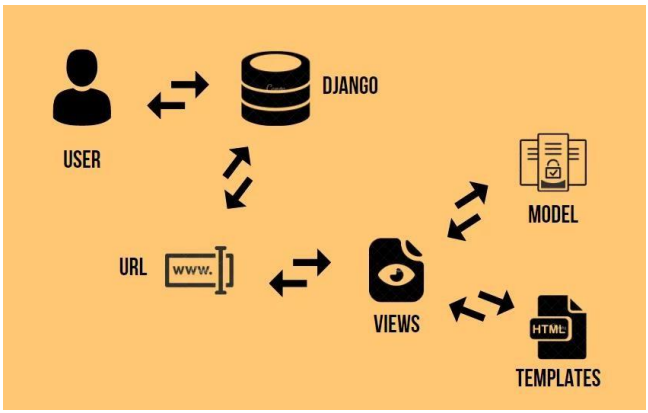
شرکت جابک مخفف "جستوجوی آسان برای کار" است. این شرکت قصد دارد محیطی آنلاین ایجاد کند که کارجویان و کارفرمایان در آن به آسانی یکدیگر را پیدا کنند و طرفین از طریق آن کسب درآمد کنند.

از ما خواسته شده است تا بخش back-end پروژه را در دست بگیریم. به عنوان یک مهندس نرم‌افزار **نیازمندی‌ها** را بررسی کردیم و دیاگرام مورد نظر را استخراج کردیم. (این عکس در یک مقاله در مورد پایگاه داده‌ی سایت فریلنسری قرار داشت. با تغییرات کوچک از آن استفاده کردم)



تمام بخش‌های مشخص شده در دیاگرام از ما خواسته نشده اما برای توسعه در بلند مدت در نظر گرفته می‌شود.

برای چنین پروژه تصمیم گرفتیم از Django استفاده کنیم. جنگو یک فریم ورک سطح بالا و عملگراست. ابزارهایی که جنگو در اختیار ما م یگذارد بسیار مفید است و **سرعت توسعه** را بالا می‌برد. در واقع نیاز به اختراع دوباره چرخ نیست. **مقیاس‌پذیری** و امنیت پروژه نیز به اندازه قابل توجهی افزایش پیدا می‌کند اما این پایان کار نیست و در کنار خود جنگو، از Django REST framework استفاده می‌شود که این پکیج پایتون است و بستری برای ما فراهم می‌کند که می‌توانیم به سرعت یک RESTful API توسعه دهیم و در کلاینت های متفاوت از آن استفاده کنیم. این فریم ورک هندل کردنِ ,serialization permissions, authentication جنگو را برای ما آسان می‌کند. الگوهای طراحی در این مدل از نرم‌افزار بسته به شرایط به میزان مناسبی ثابت و قابل پیشبینی است که generic View هایی برای ما مهیا شده‌است تا طراحی API بسیار آسان شود و کاهش نیاز به تعریف دوباره متدهای عادی مانند post, get و غیره نباشد.



به کمک استفاده از این پکیج‌ها و فریم‌ورک‌ها دیگر نیازی به پیاده سازی اکثر متدها (POST,GET,...) نیست. در اینجا با معماری **MVT** مواجه هستیم. شخصا در شروع کار انتظار پیاده سازی معماری MVC داشتم، اما تفاوت کلیدی جنگو آشکار می‌شود و خود مسئولیت Controller را به عهده می‌گیرد و View ها روی URL خاصی مپ می‌کند و وظیفه‌ی ارتباط با رابط‌کاربری تنها به عهده Templateهاست. مدل‌ها نیز **موجودیت**های(Entity) ما در دیتابیس هستند و **روابط** بین آنها و **صفت‌ها** در آن تعریف‌شده است.

در این پروژه دو اپلیکیشن جنگو ساخته شده است اپلیکیشن User برای مدل‌های یوزر و APIهای ثبت نام کارجو و کارفرما و تغییر پروفایل، و اپلیکیشن Job که برای بین ثبت آگهی و درخواست‌ها و غیره اایجاد شده است. در Views.py کارکرد APIها تعریف شده است. این اپلیکیشن شامل قسمت های دیگری نیز هست. یکی از نکات کلیدی این اپَ بهره‌گیری از کلاس پیش فرض AbstractUser جنگو است که با customization بسیار کوتاه می‌توانیم بخش اعظم authentication را مدیریت کنیم. (در واقع خود جنگو آن را کنترل می‌کند) بخش‌های دیگری در این اپلیکیشن‌های جنگو وجود دارد که در آن serializer ها و permissionها تعریف شده‌اند. با استفاده از serializer ها دریافت و ارسال موجودیت‌های ما آسان می‌شود که در اینجا به فرمت فایل (JSON تبدیل به آن، و تبدیل آن به یک object) استفاده شده است. در permission ها کلاس‌ها و متدهای مربوط به دسترسی هر مدل مدنظر تعریف شده‌اند و در قسمت Views به کمک جنریک ویوها کلاس های دسترسی آن مشخص می‌شود.

لازم به ذکر است که جنگو پا را فراتر گذاشته و پنل ادمین نیز برای ما طراحی می‌کند. برای اینکار تنها لازم بود مدل‌های مورد نظر را در فایل ادمین رجیستر کنیم تا ادمین دسترسی تغییرات مستقیم در دیتابیس را با رابط کاربری عالی داشته باشد.

یکی از توسعه دهندگان جنگو معتقد است که"Code without tests is broken by design."

این مورد به اهمیت تست در توسعه اشاره دارد. دلایل بسیار منطق یای وجود دارد که چرا باید تست نوشت اما در این پروژه به علت زمان کوتاه و عنوان نشدن آن، از این مورد چشم پوشی کردیم.

اپلیکیشن دوم که در این پروژه وجود دارد و کار اصلی را به عهده دارد Job است. که از نظر الگوی طراحی اشتراکات بسیاری وجود دارد. ما به تفاوتهای آن می‌پردازیم.

نکته مهم در طراحی مدل‌های این اپلیکیشن این بود که هر *کاری* که تعریف شده بود، می‌توانست یک لیست از درخواست‌های فریلنسرها داشته باشد. اما این مورد با آنکه دسترسی ما به آن درخواست ها را تسریع می‌کرد (دیگر نیازی به جست‌وجو نبود و مستقیما به لیست مورد نظر دسترسی داشتیم) اما پیچیدگی زیادی به دیتابیس اضافه می‌کرد. پس تصمیم گرفته شد که هر proposal یک رابطه مشخص با فریلنسر و کار درخواست شده داشته باشد.

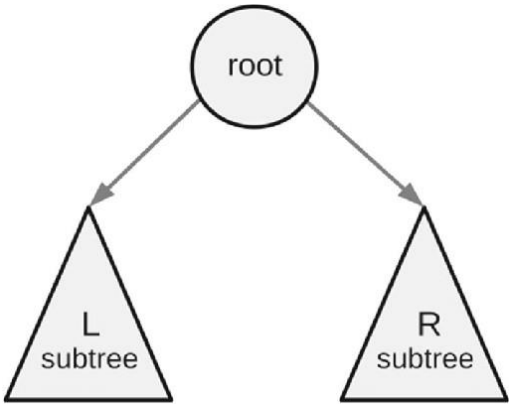
در این اپلیکیشن برای هندل کردن الویت‌هایی برای مشاهده‌ی *کارها* براساس محدویدتهای زمانی و مهارتهای مورد نظر، فیلترهایی طراحی شده اند. فیلتری که ما خود اضافه کردیم فیلتر کردن بازه‌ی قیمت است. از آنجا که ما به فیلترهای دیتابیس دسترسی نداشتیم (طبق فرض پروژه) تمام *کارها* در یک درخت جستوجو ی دودویی ذخیره کردیم. این ساختمان داده در مقیاس بالا به شدت هزینه‌ها را کاهش می‌دهد.

هزینه زمانی فیلتر کردن بازه‌ی قیمت، در این ساختمان داده با الگوریتم نوشته شده $O(h + k)$ است که h ارتفاع درخت و k تعداد عناصر میان این بازه است .در مقیاس بالا که این مقایسه با $o(n)$ صورت می‌گرفت زمان به حد چشمگیری کاهش پیدا می‌کند.

چرا درخت جست‌وجوی دودیی؟

۱. روشی‌ایده‌ال برای ذخیره‌ی داده‌هایی که سلسه مراتب دارند.
۲. بازتاب روابط ساختاریافته ای که در این مجموعه از داده وجود دارد.
۳. اضافه کردن و حذف عناصر در مقایسه با آرایه‌ها به شدت کاهش می‌یابد
٤. بسیار منعطف پذیر است.
٥. در دسترسی بین عناصر از لینک لیست سریع‌تر و از آرایه کندتر است.

در ادامه این نکات، باید بدانیم در مقیاس‌های بالا پیچیدگی زمانی جستوجو نسبت به جستوج وهای خطی به‌صورت نمایی کاهش می‌یابد زیرا پیچیدگی زمانی پیدا کردن عنصر مورد نظر $O(log2(n))$ است یا همان $O(h)$ که ارتفاع درخت است. سرعت این الگوریتم زمانی که شاید ۲۰ عصنر داریم خود را نشان ندهد اما زمانی که بیش از نیم میلیون عنصر داریم کاملا قابل مشاهده است.پیچیدگی زمانی اضافه یا حذف عناصر نیز $O(log2(n))$ است. از آنجا که ما نودهای متفاوتی داریم از نظر حافظه نیز برای مقیاس بالا بسیار مفید است. دسترسی به عنصر میانی $o(1)$ است. و ما م یتوانیم این داده هارا بین سرور تقسیم کنیم که قابلیت مقیاس‌پذیری این ساختمان داده را افزایش می‌دهد.



برای مثلا بخشی از داده در زیر درسخت سمت راست می‌تواند در یک سرور متفاوت باشد.

نکته حائز اهمیت آن است که ما به دلیل سادگی از دیتابیس دیفالت جنگو که sqlite است استفاده کردیم اما برای چنین پروژه‌ای قطعا از دیتابیسی نظیر MySQL, PostreSQL و نظیر آن استفاده خواهیم کرد.

از زمانی که برای خواندن این گزارش در اختیار بنده قرار دادید، سپاسگزارم .

به امید دیدار.

مهدی آذری