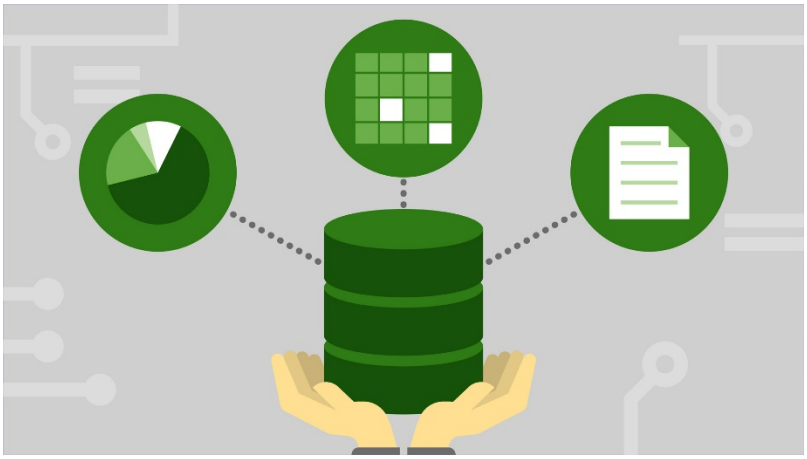
به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر



آزمایشگاه پایگاه‌داده

دستورکار شماره ۲

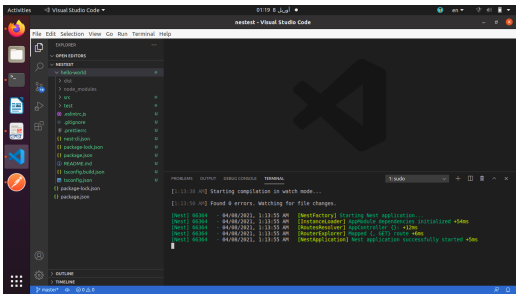
شماره دانشجویی

۸۱۰۱۹۶۴۲۳

گزارش فعالیت‌های انجام شده

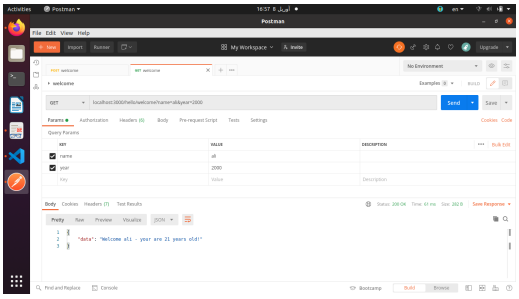
- **قسمت اول**

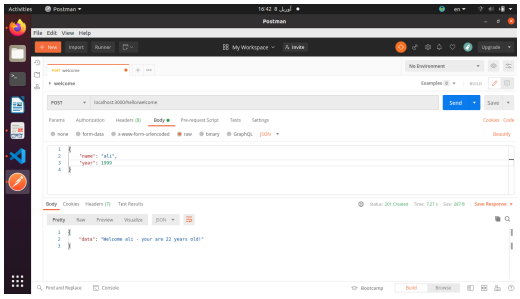
در تصویر مقابل، موفق آمیز بودن ساخت پروژه hello-world مشخص می‌باشد.



سایر گام‌های اجرای پروژه مطابق توضیحات دستور پروژه اجرا شد.

در دو تصویر زیر نیز دو API نهایی از طریق postman صدا زده شده‌اند و نتایج مشخص می‌باشد.





تمام کدهای این پروژه بر روی فولدر Lab2 قرار دارد.

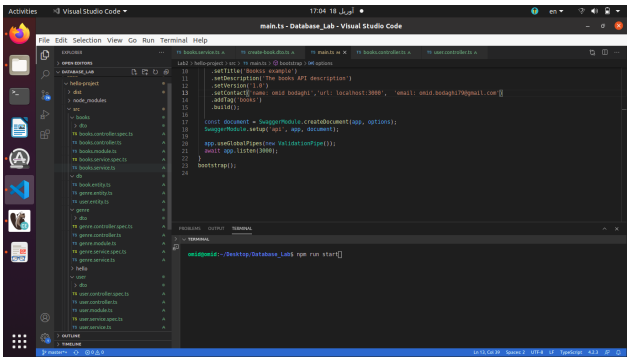
آدرس مخزن گیت: https://github.com/omigo2000/Database_Lab.git

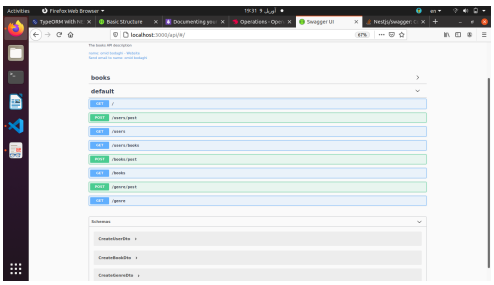
متن آخرین کامیت ثبت شده برای قسمت ۱ دستور کار: Initialize nest.js with hello-project(Sec. 1)

شناسه آخرین کامیت ثبت شده برای قسمت ۱ دستور کار: 94b9d53e42304d7a58bbb56e7fea742e1ffbd395

- **قسمت سوم**

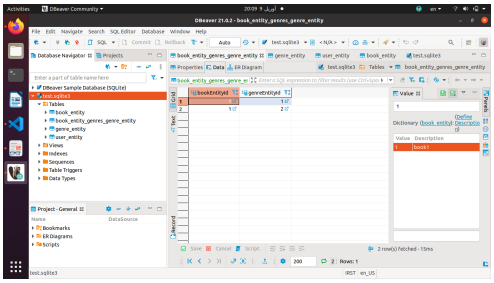
در این قسمت، با استفاده از typeorm با دیتابیس SQLite کار خواهیم‌کرد. ابتدا سه ماژول books، genre و user را می‌سازیم. تصویر زیر از محیط vscode گرفته شده‌است که نشان‌دهنده‌ی ساخته‌شدن این فایل‌ها می‌باشد. به دلیل مشابهت با لینک قرار داده‌شده، کدهای هر قسمت را قرار نمی‌دهم.

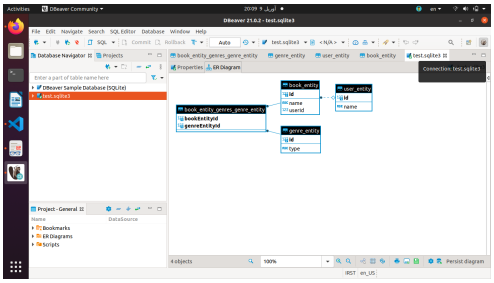




حال پس از اضافه‌کردن مستندسازی‌ها نمای سواگر به صورت شکل مقابل است. برای زیاد نشدن تعداد عکس‌ها، بازشده‌ی هر قسمت در گزارش ثبت نشده‌است. همانطور که مشاهده می‌شود، برای تمام اندپوینت‌های API مستندسازی انجام شده‌است. در انتها از طریق سواگر اقدام به وارد کردن کتاب، یوزر و ژانر می‌کنم. دو تصویر از پایگاه‌داده برای نمایش در اینجا آورده شده‌است.

در تصویر سمت چپ، جدولی که برای ارتباط book و genre می‌باشد آورده شده‌است و مشاهده می‌شود یک ردیف به آن اضافه شده‌است و کتاب با شناسه ۱، ژانری با شناسه ۱ دارد. در تصویر سمت راست، ER Diagram تولیدشده توسط DBeaver مشاهده می‌شود.





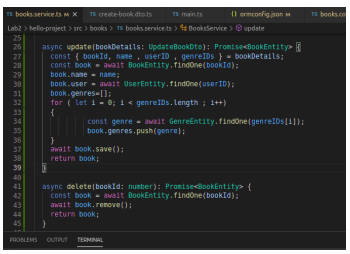
متن آخرین کامیت ثبت شده برای قسمت ۳ دستور کار: Implement bookstore using sqlite3

شناسه آخرین کامیت ثبت شده برای قسمت ۳ دستور کار: 246efc426263b47bc8e1c11f286f44334c48e5d6

• قسمت چهارم

در این قسمت بجای کار با SQLite از پستگرس استفاده می‌کنم. ابتدا فایل ormconfig را به صورت زیر تغییر می‌دهم. username و password ای که ساخته شده‌است todoapp1 می‌باشد. نام پایگاه‌داده نیز postgress می‌باشد.

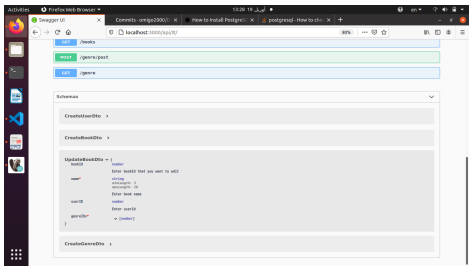
حال توابع put و delete را پیاده‌سازی می‌کنم. در تصویر مقابل، کنترلر book را تغییر داده‌ام و دو اندپوینت برای delete و put را اضافه می‌کنم. برای پیاده‌سازی delete از param و برای put، از update-book.dto استفاده شده‌است که همانند create-book.dto می‌باشد و صرفا یک متغیر bookId بیشتر دارد. وضعیت‌های مختلف(کد ۲۰۰ و ۴۰۰) در حالات مختلف request پیاده‌سازی شده‌اند.

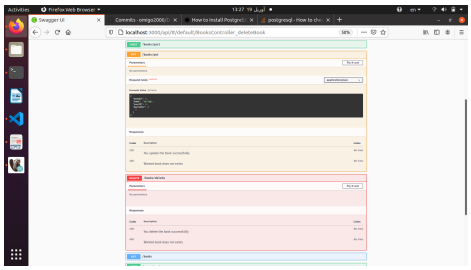


در تصویر مقابل، service را برای دوتابع update و delete پیاده‌سازی می‌کنیم. update بسیار شبیه insert می‌باشد صرفا بجای new BookEntity() با findone() کتاب مورد نیاز برای

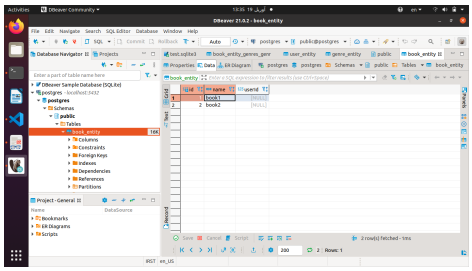
تغییر را پیدا می‌کنم. در delete نیز مجددا با findone کتاب را پیدا می‌کنم و با استفاده از ()remove آن را حذف می‌کنم.

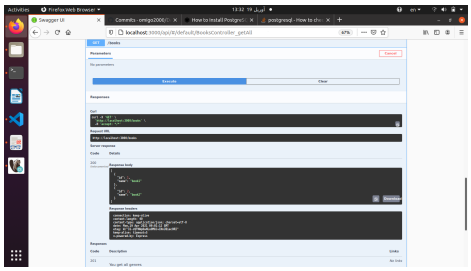
قسمت‌های تغییر یافته در سواگر به صورت زیر می‌باشد و مشاهده می‌شود قسمت update و delete اضافه شده‌است:



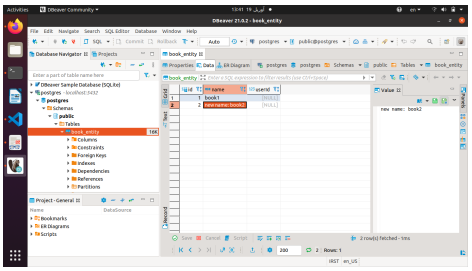


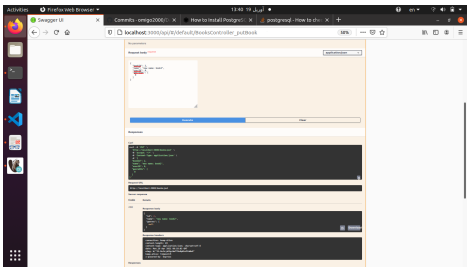
برای اطمینان از درستی توابع پیاده‌سازی شده، ابتدا دو کتاب را اضافه می‌کنیم و نتایج آن در سواگر و پستگرس به صورت زیر است:



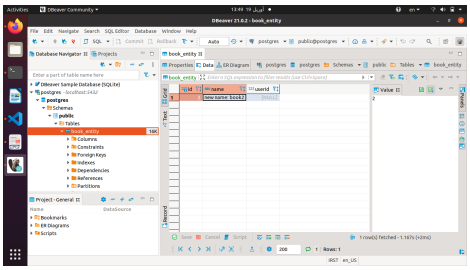


با استفاده از سواگر و به صورت مقابل، نام کتاب با شناسه ۲ را به صورت زیر تغییر می‌دهیم و مشاهده می‌شود در پایگاه‌داده نیز نام کتاب دوم تغییر کرده‌است.





در انتها کتاب با شناسه ۱(bookId=1) را توسط endpoint با نام delete حذف می‌کنیم و همانطور که مشاهده می‌شود، در پایگاه داده این کتاب حذف شده‌است.



متن آخرین کامیت ثبت شده برای قسمت ۴ دستور کار: Implement connection to postgreSql

شناسه آخرین کامیت ثبت شده برای قسمت ۴ دستور کار: 17ecb59ab54db7f429157d9f1291acf91abbdcd3

- **قسمت پنجم**

ابتدا کتابخانه‌های لازم را نصب می‌کنیم. سپس ماژول و سرویس auth را در پوشه‌ی src ایجاد می‌کنیم. AuthService وظیفه‌ی دریافت کاربر و چک کردن رمز عبور او را برعهده دارد. در این فایل، تابع validateUser را که وظیفه بررسی کردن وجود نام کاربری داده‌شده و درست بودن رمز را برعهده دارد، ایجاد کرده‌ام.

حال برای ایجاد استراتژی فایل local.strategy.ts ایجاد می‌شود و در تابع validate آن، تابعی که در بالا توضیح دادم، صدا زده می‌شود. در صورتی که کاربر وجود نداشت یا رمز عبور نادرستی داشت، خطای کاربر احراز نشده(()UnaouthorizedException) فعال می‌شود. در صورت درست بودن این فرآیند، اطلاعات کاربر برگردانده می‌شود.

Gaurds امکاناتی را فراهم می‌کند تا در صورتی که کاربری احراز هویت نشده‌بود، امکان استفاده از سرویس‌های خاصی را نداشته‌باشد و در صورت احراز هویت شدن، با فراهم کردن jwt، دسترسی وی در درخواست‌های بعدی را نیز فراهم کند. همچنین در صورتی که کاربر احراز هویت شده‌باشد، می‌تواند به منابع تعریف شده برای او دسترسی داشته‌باشد. در فایل local-auth.guard.ts یک گارد از نوع local ایجاد کرده‌ام.

حال در فایل auth-service.ts، تابع login را نیز اضافه می‌کنم که وظیفه ساخت jwt را دارد. به عنوان ورودی username و name را در payload قرار می‌دهد و آن را sign می‌کند. حال رمز دلخواه و بسیار مهم خودم را در constants.ts اضافه می‌کنم.

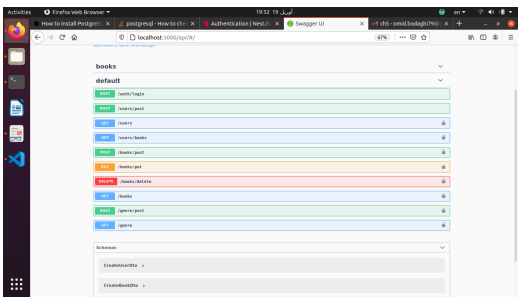
در این مرحله، در auth.module نیز JWT را اضافه می‌کنم و زمان منقضی شدن آن را ۶۰ ثانیه قرار می‌دهم.

در مرحله آخر، با استفاده از JWT passprt، برای اندپوینت‌ها شرط وجود یک jwt معتبر را قرار می‌دهم. با jwtFromRequest، jwt از request بدست می‌آید. ignoreException را نادرست قرار می‌دهیم تا به انقضا اهمیت داده‌شود. همچنین secret key را نیز تعیین می‌کنیم.

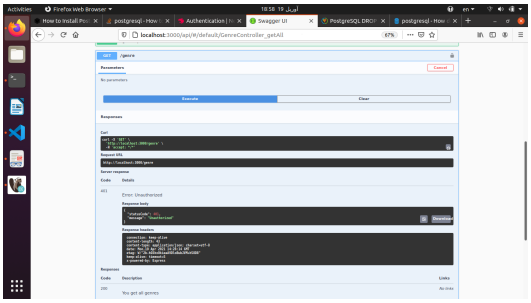
برای استفاده از jwt در اندپوینت‌ها، در قسمت providers، علاوه بر سرویس متناظر هر ماژول، عبارت مقابل را نیز اضافه کرده‌ام:

{provide: APP_GUARD, useClass: JwtAuthGuard,}

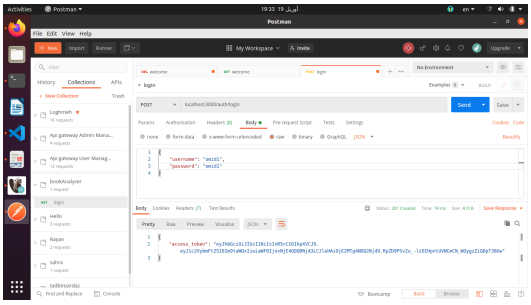
همچنین برخی اندپوینت‌های سیستم مانند ساخت کاربر جدید نیاز به احراز هویت ندارند. برای این موارد، می‌توان از @Public() استفاده‌کرد. تعریف این تابع در custom-decorator.ts قابل مشاهده‌است. حال در jwt-auth-guard.ts بررسی می‌کنیم و اگر Public بود، مقدار true را باز میگردانم و jwt بررسی نمی‌شود.



در نهایت همچیز آماده می‌شود و در زیر به توضیح تست و استفاده از سواگر خواهم پرداخت. همانطور که در تصویر مقابل مشاهده می‌شود، بجز تابع login و ساخت کاربر، در کنار تمام توابع تصویر قفل وجود دارد که نشان‌دهنده‌ی نیاز آنها به jwt می‌باشد.



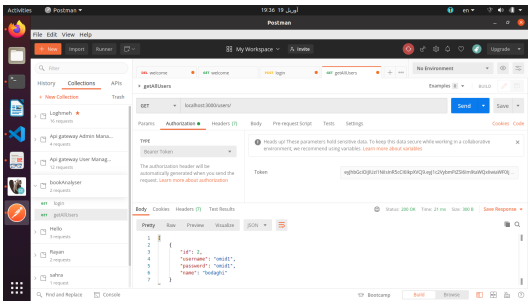
حال یک کاربر ایجاد می‌کنم. مطابق تصویر مقابل، در صورتی که بخواهیم لیست ژانرها را بازگردانیم، کد ۴۰۱ دریافت می‌کنم و نیاز به احراز هویت است.



بنابراین، از طریق postman، ابتدا کاربر را login می‌کنیم.

مطابق تصویر مقابل، در خروجی درخواست، token برای استفاده‌های بعدی کاربر تولید شده‌است.

حال این توکن را در درخواست‌های مورد نظر، در قسمت Authentication (گزینه‌ی Bearer) اضافه می‌کنیم و درخواست‌های خود را انجام می‌دهیم.



همانطور که مشاهده می‌شود، با افزودن jwt امکان مشاهده لیست کاربران را خواهیم داشت.

بنابراین فرآیند احراز هویت نیز با موفقیت انجام شد.

متن آخرین کامیت ثبت شده برای قسمت ۵ دستور کار: Implement Authentication

شناسه آخرین کامیت ثبت شده برای قسمت ۵ دستور کار: 35461a878244f35a6b5620e1664ca107b7f81dab

- **قسمت دوم**

در این قسمت، API ها را مطابق جدول زیر طراحی می‌کنیم. چهار قابلیت به شرح زیر می‌باشد:

۱- امکان ثبت‌نام، ویرایش اطلاعات، مشاهده و حذف کاربر(برای کارفرما و فریلنسر)

۲- امکان اضافه‌کردن شرکت، مشاهده اطلاعات، ویرایش و حذف اطلاعات شرکت(برای کارفرما)

۳- امکان اضافه‌کردن پروژه، مشاهده، ویرایش و حذف پروژه(برای کارفرما)

۴- امکان اضافه‌کردن درخواست، مشاهده، ویرایش و حذف درخواست(برای فریلنسر)

Resource	Method	Request	Response	Description
/users	POST	UserObj = {username, firstname, lastname, email, password}	[200, 400: Bad Request]	Create new user
/users/:username	Get		UserObj [200, 400: Bad Request]	Retrieve user details
/users/:username	PUT	UserObj	UserObj [200, 400: Bad Request]	Update user info
/users/:username	Delete	UserObj	UserObj [200, 400: Bad Request]	delete user
/users/:username/company	POST	CompanyObj= {id, name, location, website}	[201, 400: User not found]	Register company informations for the user {id}
/users/:username/company	GET		CompanyObj [200, 400: User not found]	Retrieve company informations for the user {id}
/users/:username/company	PUT	CompanyObj	CompanyObj [201, 400: User not found]	Update company informations for the user {id}
/users/:username/company	DELETE		CompanyObj [201, 400: User not found]	Remove company informations for the user {id}

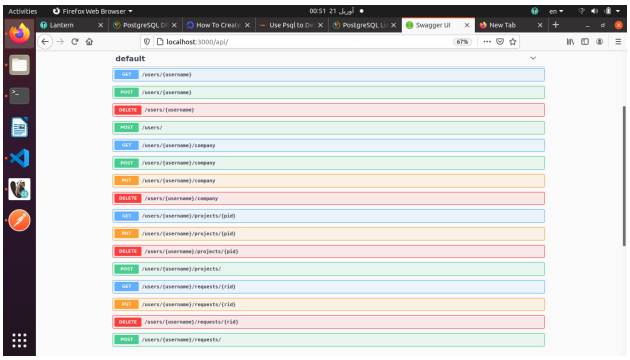
/users/:username/projects	POST	projectObj = {title, budget, deadline, filePath]	[200, 400: User not found]	Create new project
/users/:username/projects/:pid	GET		projectObj [201, 400: User or project not found]	Retrieve the details for project {pid}
/users/:username/projects/:pid	PUT		projectObj [201, 400: User or project not found]	Update the details of project {pid}
/users/:username/projects/:pid	DELETE		projectObj [201, 400: User or project not found]	Remove project {pid}
/users/:username/requests/	POST	reqObject= {projectId, budget, deadline}	[200, 400: user or project not found, repeated project]	Create new request
/users/:username/requests/:rid	GET		reqObject, [200, 400: user or project not found]	Retrieve the details for request {pid}
/users/:username/requests/:rid	PUT		reqObject, [200, 400: user or project not found]	Update the details of request for {pid}
/users/:username/requests/:rid	DELETE		[200, 400: user or project not found]	Remove request for{pid}

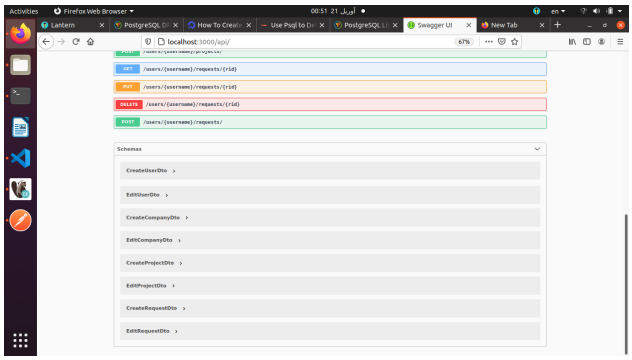
- **قسمت ششم**

در این قسمت، ماژول jobseekers را ایجاد کرده‌ام. در فایل Entity موجودیت‌ها را تعریف کرده‌ام. مطابق جدول قسمت دوم، هر کاربر صرفا می‌تواند اطلاعات یک شرکت را ثبت کند و هر شرکت می‌تواند برای دقیقا یک کاربر باشد. پس بنابراین، یک رابطه‌ی One to One بین user و company وجود دارد. همچنین بین user و درخواست‌های خود(requests) و پروژه‌های خود(projects) یک رابطه One to Many وجود دارد و هر کاربر، می‌تواند چند پروژه تعریف کرده و یا درخواست داده شده داشته‌باشد. برای هر کدام از ساخت کاربر، ویرایش کاربر، ساخت درخواست، ویرایش درخواست، ساخت شرکت، ویرایش شرکت، ساخت پروژه و ویرایش پروژه، یک dto ایجاد

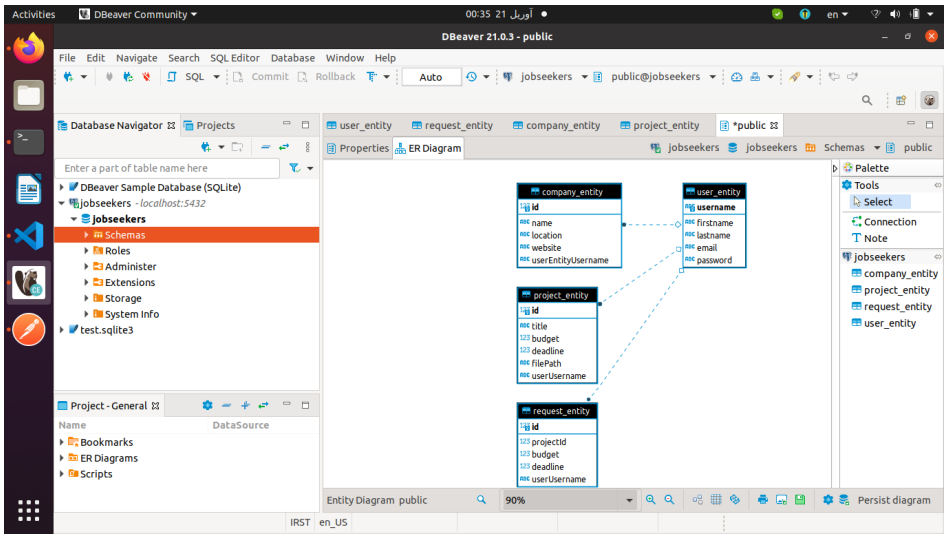
شده‌است. تمام اندپوینت‌ها با استفاده از جدول قسمت ۲، در jobseekers.controller.ts و سرویس‌های استفاده شده در jobseelers.service.ts وجود دارد. همچنین در dto و controller، با استفاده از swagger، داکیومنتیشنی ایجاد شده‌است.

در دو تصویر زیر، صفحه سواگر و اندپوینت‌های اضافه شده و dto ها قابل مشاهده می‌باشد:





همچنین ER Diagram بدست آمده توسط DBeaver به این صورت می‌باشد:



متن آخرین کامیت ثبت شده برای قسمت ۶ دستور کار: Complete jobseekers

شناسه آخرین کامیت ثبت شده برای قسمت ۶ دستور کار: 589210700fb0ebd2778c7cdb7ed9cc560bf61f66

مشکلات و توضیحات تکمیلی

حجم پروژه بسیار زیاد بود و زمان بسیار زیادی صرف آن شد. از نظر اینجانب، یادگیری سواگر و یا فرآیند احراز هویت قسمت‌هایی می‌باشند که بودنشان بار یادگیری زیادی ندارد.
همچنین منبع قسمت ۳(لینک قرار داده‌شده) اشتباهات پیاده‌سازی زیادی دارد.
احتمالا کار کردن با دستورات postgreSql یا شاخص‌ها و موارد مشابه یادگیری بیشتری را فراهم می‌کرد. البته یادگیری typeorm بسیار ارزشمند بود اما قسمت کمی از پروژه به آن اختصاص داشت.
همچنین حجم زیادی کد بسیار مشابه و صرفا وقت گیر برای قسمت ۶ نوشته شد.

آنچه آموختم

آشنایی با Swagger برای داکیومنت کردن و typeorm که بسیار ارشمند بود. همچنین با فریمورک nestjs آشنا شدم و توانستم یک نمونه نسبتا کامل شامل فرآیند احراز هویت، ایجاد API، دریافت request ها و ایجاد response ها و همچنین نگهداری داده‌ها در پایگاه‌داده را پیاده سازی کنم.