IWB-TODO-JAVA

Çok kullanıcı için geliştirilmiş her bir kullanıcının
kendisine özel todo lar oluşturabildiği güvenlikli bir ***Todo Application Java Backend***

Proje yapısı ***çok katmanlı mimari*** ile tasarlandı bu katmanlar;

- Entities
- Data Access
- Business
- Core
- API

ENTITIES

- Entity data classlarımızdır bunlar database de ki table lara karşılık gelen classlardır bir diğer adıyla ***domain object*** ler, örneğin "todos" table ı için "Todo" class ı Java Spring Boot ("**@Table**" ***annotation*** ile) ile table larla class ları bağlar.

- **@Data**, **@AllArgsConstructor**, **@NoArgsConstructor** annotationları **lombok** kütüphanesinden gelir bunlar bizim için gerekli constructor ları ve getter setter ları oluşturur temiz bir görünüm sağlar.

- ***Payload*** verileri taşımak ve yapılandırmak için kullanılan kısımlardır, örneğin **loginRequest** için kullanıcından username ve şifre almamız gerekli bunu bu class ı kullanarak alabiliyoruz ya da kullanıcıya kendi oluşturmuş olduğumuz bir veriyi göndermek istiyoruz **userResponse** gibi yani ***database ile bir bağlantısı olmayan*** veriyi taşırken kullanırız.

- **@CreationTimestamp**, **@UpdateTimestamp** ***Hibernate*** tarafından create edildiği ve değiştirildiği zamanları database e otomatik olarak işler.

- **@NotBlank, @Size(max = 20), @Email** annotationları **javax.validationdan** kütüphanesinden geliyor gönderilen değerin istenilen şekilde olup olmadığını ilk bu şekilde kontrol ediyoruz, örneğin email yerine 124 gibi bir değer gönderilemez.

DATA ACCESS

- **Data Access Object (DAO)** lerimizi tutuyoruz bunlarla birlikte database e sorgu göndererek verilerimizi isteğimiz doğrultusunda çekebiliyoruz, burada **JpaRepository** kullanıldı default işlemler hali hazırda ekli örneğin ***save()*** create ve update işlemleri için, ***findAll()*** tüm veriyi getirmek için gibi. FindAll içerisinde paging ve sort işlemleride yapılabiliyor bunlar ***direkt olarak database e gerekli sorguyu gönderiyor gelen veri üzerinde işlem*** yapılmıyor yani bize ek bir yük binmiyor işlemler hızlı yapılıyor.

- Bir çok işlem yalnızca method isimlendirmeleriyle halledilebiliyor **(Automatic Custom Queries)** örneğin,
  ```
  List<Todo> getAllByUserId(Long userId);
  ```
  Bu işlemle birlikte JpaRepository bizim için gerekli ***query*** yi hazırlıyor ve istediğimiz veriyi istedğimiz şekilde dönüyor.

- Eğer query yazılması gereken komplike durumlar olursa ***(Manual Custom Queries)*** ***@Query("")*** annotation ını kullanabiliriz.

BUSINESS

- Bu katmanda tüm işlemleri (kurallar, logicler) yerine getiriyoruz örneğin veriyi kaydetmek için gerekli kontrolleri yapmak çekmek istediğimiz veriyi istediğimiz hale getirmek kontrolleri yaptıktan sonra result ı dönmek gibi, örneğin password kontrolü, silinecek user var mı yok mu, silinecek todo silmeye çalışan user a ait mi değil mi, Register da ***email*** ve ***username*** (bunlar database imizde **unique**) daha önce kullanılmış mı kontrolleri yapılır.

- Buradaki ***Service* Interface** lerini kullanılma amacı **SOLID** design prensiplerine uygun bir şekilde yol almak bu şekilde refactoring, debuging işlemleri kolaylaşıyor ve kod okunurluğu artıyor.

- **SOLID**
  S - Single-responsiblity Principle (her bir class kendi işini yapmalı mesela roleManager class ında user ları databaseden çekmemeliyiz)

  O - Open-closed Principle (var olan özellikler silinmemeli üzerine gerekli özellikler eklenmeli bu şekilde yeni ve eski desteği korunmuş olur)

  L - Liskov Substitution Principle (buna örnek core->utilities->results burada ki Result class ı tüm resultların base i Result ta bulunan message ve success child classlara da aktarılıyor ve tekrar tekrar yazmayı engellemiş oluyoruz)

  I - Interface Segregation Principle (oluşturmuş olduğumuz service interfaceleri ile beraber AuthManager class ında roleService i inject ederek roleManager da bulunan methodlara erişebiliriz)

  D - Dependency Inversion Principle (örneğin jwtUtils impelemts tokenUtils ve eğer biz jwt yerine başka bir yöntemi kullansaydık bu yöntemde tokenUtils i implement etseydi hem test hem de başka bir teknolojiye geçişimiz kolaylaşacaktı bağımlılığımız azalacaktı)

API

- Bu katman gelen **HTTP** isteklerini karşılar bu istekler action metotları ile örtüşür ***GET, POST, PUT, DELETE*** gibi ve bu metotlara göre istekler gönderilen veriyle beraber business katmanına yönlendirilir.

- Dönen result un success durumuna göre **ResponseEntity** olarak **HttpStatus** ile beraber dönülür.

- Gelen istekteki değerlerin validationdan geçmesi için @**Valid** annotationını kullanarak validationdan geçiriyoruz ve **validationExceptionHandler** methodunu kullanarak gelen hataları ayrıştırıp kendi result methodumuza uygun şekilde dönüyoruz.

CORE

- Bu katmanda ***ortak olan işlemler*** tutuluyor mesela user birçok proje için ortak ve user ile ilgili işlemleri burada tutuyorum role ataması ortak ve utilities ler proje için ortak.

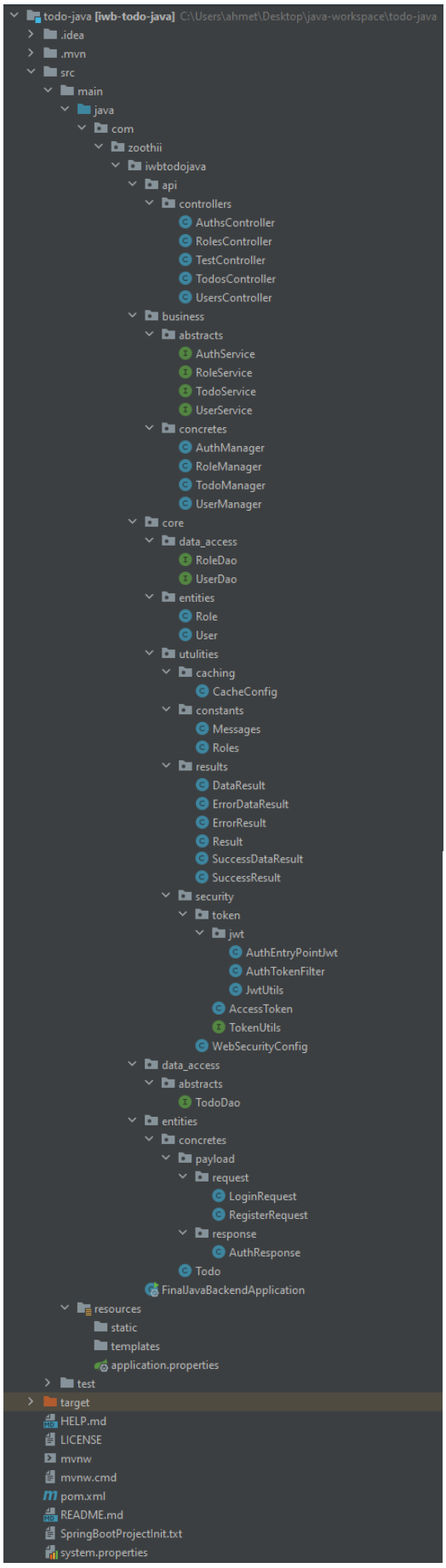
- Utilities->caching burada **cacheConfig EhCacheManager** ı kullanarak verilerin cache de ne kadar süreyle tutulacağını belirlemek için yazıldı.

- Utilities->constants burada **Message** lar ve **Default Roller** tutuluyor.

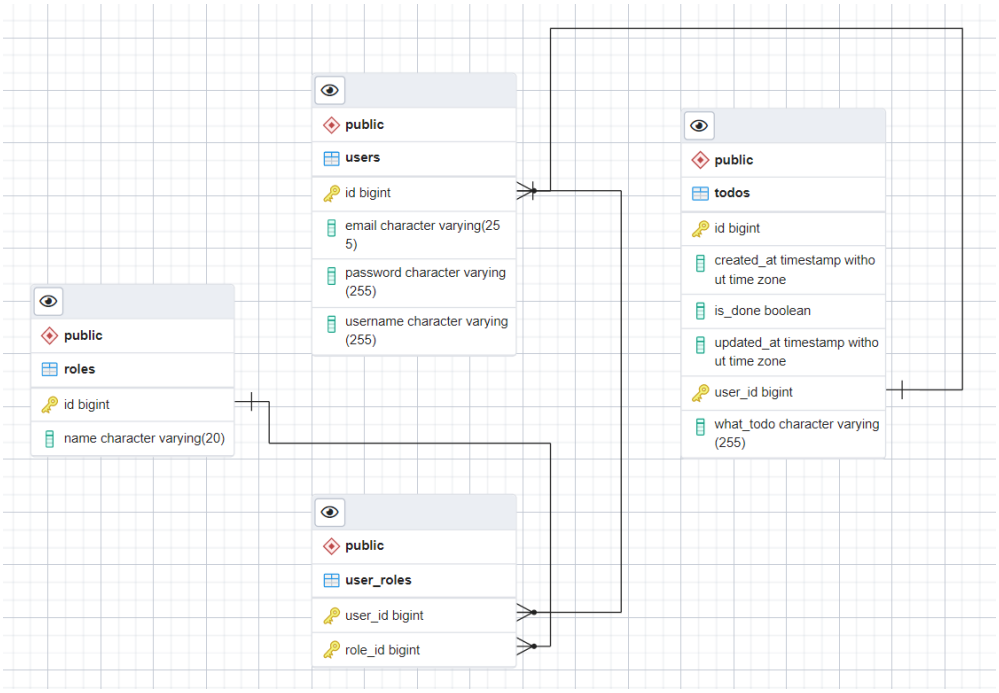
- Utilities->results burada result un durumuna göre dönüş yapabileceğimiz resultlar tutuluyor Data dönenler için **DataResult** ve dönmeyenler için **Result** sınıfı var bu sınıflar Result sınıfının alt sınıfları ve Result sınıfıda ***hateoas.RepresentationModel*** extend ediyor ve bu şekilde resultlarımıza linkleri ***(hypermedia-driven outputs)*** ekleyebiliyoruz.

- Utilities->security burada güvenlik işlemleri yapılıyor, **password encoding** (PBKDF2WithHmacSHA512 ve BCryptPasswordEncoder kullanılarak), **jwt token creation** (5 dakika kullanım süresine sahip, her girişte yenilenen), **http cors izinleri**, her request için **UsernamePasswordAuthentication** kontrolü yapılır

Proje Yapısı



Database Yapısı



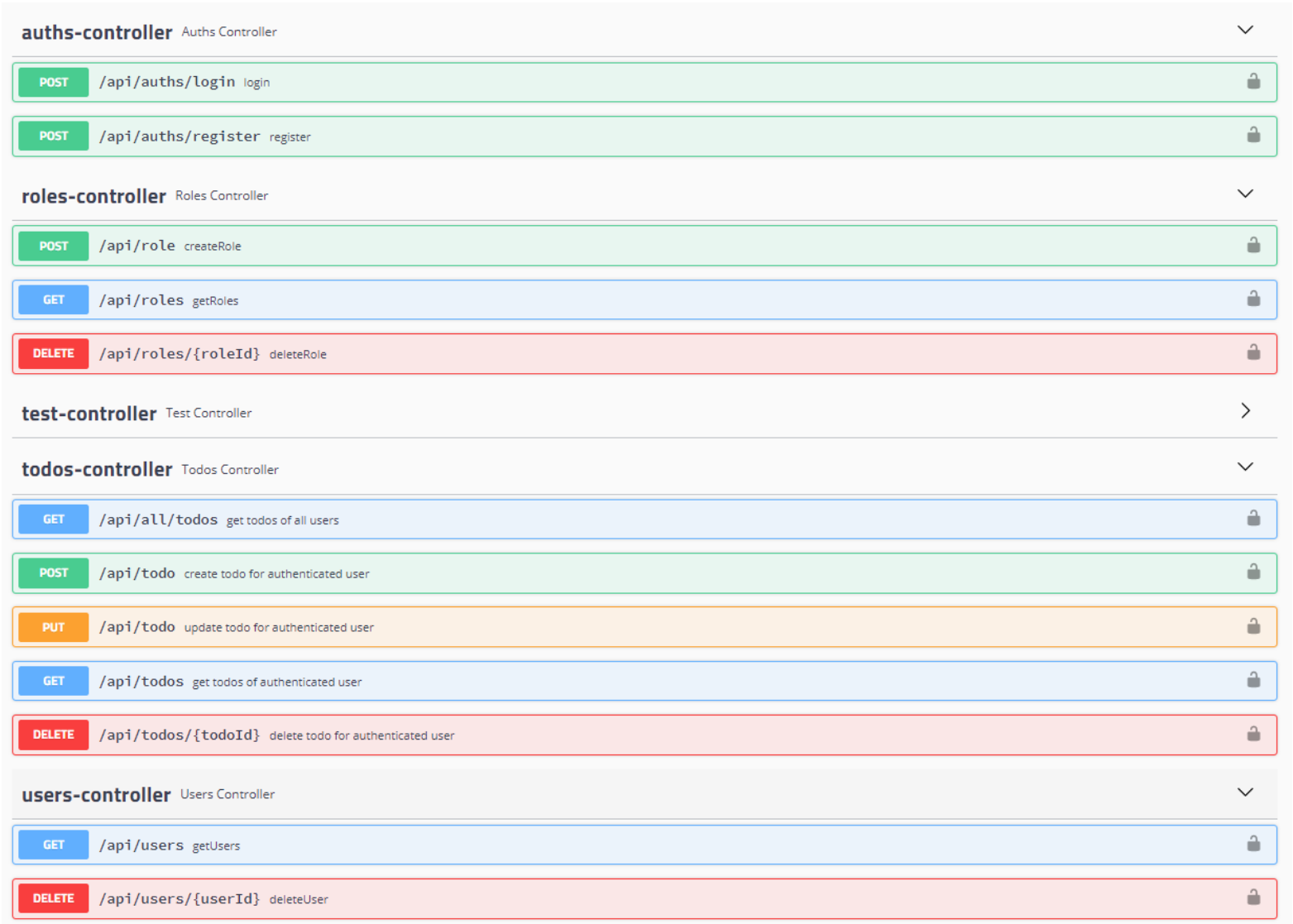
Controller İşlemleri

API documentasyonu için Swagger implementasyonu yapılmıştır.

Heroku üzerinden proje deploy edilmiştir.
https://iwb-todo-java.herokuapp.com/swagger-ui.html#/ üzerinden Swagger a ulaşılabilir ve test edilebilir.

LOCAL** de Tüm controller için* **Base URL** *: **localhost:8181/
HEROKU** de Tüm controller için* **Base URL** *: **iwb-todo-java.herokuapp.com/

Ben localde **POSTMAN** *üzerinden testlerimi gerçekleştirdim.*



AuthsController

Giriş ve Kayıt işlemleri için request body ve responselar aşağıdaki gibidir, responselar gönderilen request sonucuna göre message, success ve status code olarak değişiklik göstermektedir.

http://localhost:8181/api/auths/register

POST http://localhost:8181/api/auths/register

Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

Beautify

```
{
  "email": "string@gmail.com",
  "password": "string",
  "roles": [],
  "username": "string"
}
```

Body Cookies Headers (14) Test Results

Status: 200 OK Time: 4.72 s Size: 979 B Save Response

Pretty Raw Preview Visualize JSON

```
{
    "success": true,
    "message": "User registered successfully.",
    "data": {
        "accessToken": {
            "token": "eyJhbGciOiJIUzUxMiJ9.eyJqdGkiOiIxIiwic3ViIjoic3RyaW5nIiwiaWF0IjoxNjI5MDIxNTQyLCJleHAiOjE2MjkwMjE4NDJ9.49CVB8cnt2FCX_LKPjHJ9sxpKxHTooMiubnlP7Apl2zql-EuSyhKjO23Cbf6i59O0Ozp44YURiVYGzA5IpuzFw",
            "type": "Bearer",
            "expiration": "2021-08-15T10:04:02.353+00:00"
        },
        "id": 1,
        "username": "string",
        "email": "string@gmail.com",
        "roles": [
            "ROLE_USER"
        ]
    },
    "_links": {
        "self": {
            "href": "http://localhost:8181/api/auths/register"
        },
        "auths": {
            "href": "http://localhost:8181/api/auths"
        }
    }
}
```

Eğer username ve email daha önce varsa message olarak zaten var dönecek, eğer role yoksa role yok dönecek bu işlem default roller için geçerli değil (ROLE_USER, ROLE_ADMIN)

Herhangi bir role verilmezse default role (ROLE_USER) üretilir ve kullanıcıya atanır.

Admin role verilirse (ROLE_ADMIN) bu rol üretilir ve default role ile birlikte kullanıcıya atanır.

Request body payloadlardan RegisterRequest sınıfına denk gelmektedir.
username min 3 max 20 karakter olmalıdır.
email max 50 karakter olmalıdır, email standartlarında olmalıdır.
password bir sayı, bir küçük harf, bir büyük harf “@#$%” karakterlerden birini içermelidir.
password 20 karakterden çok 8 karakterden az olamaz.

```
@Pattern(regexp = "^(?=.*\\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[@#$%]).{8,20}$")
```

http://localhost:8181/api/auths/login

POST http://localhost:8181/api/auths/login

Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

Beautify

```
{
  "password": "string",
  "username": "string"
}
```

Body Cookies Headers (14) Test Results

Status: 200 OK Time: 2.91 s Size: 975 B Save Response

Pretty Raw Preview Visualize JSON

```
{
    "success": true,
    "message": "User logged in successfully.",
    "data": {
        "accessToken": {
            "token": "eyJhbGciOiJIUzUxMiJ9.eyJqdGkiOiIxIiwic3ViIjoic3RyaW5nIiwiaWF0IjoxNjI5MDIxNzU2LCJleHAiOjE2MjkwMjIwNTZ9.cA5D0nYRUHb-DbAe_CLVrKpKaJnhNXWYGWyr0CDlqdNNEtUQe1-XqT5A-XLVn3tWfj804JY_dCavwF-hIQJ9uA",
            "type": "Bearer",
            "expiration": "2021-08-15T10:07:36.636+00:00"
        },
        "id": 1,
        "username": "string",
        "email": "string@gmail.com",
        "roles": [
            "ROLE_USER"
        ]
    },
    "_links": {
        "self": {
            "href": "http://localhost:8181/api/auths/login"
        },
        "auths": {
            "href": "http://localhost:8181/api/auths"
        }
    }
}
```

Request body payloadlardan LoginRequest sınıfına denk gelmektedir.

password yanlış girilirse message olarak şifre yanlış success false döner.

username yanlış girilirse böyle bir username yok döner.

RolesController

role create işlemi için

```
{
  "name": "ROLE_TEST"
}
```

Request body ROLE_ ile başlamalıdır.

Role delete işleminde var olmayan bir role silinmeye çalışılırsa role yok message ı döndürülür.

UsersController

Get users var olan tüm kullanıcıları döner password u dönmez password için **@JsonIgnore** annotation ı kullanılır.

```
{
  "success": true,
  "message": null,
  "data": [
    {
      "id": 1,
      "username": "string",
      "email": "string@gmail.com",
      "roles": [
        {
          "id": 1,
          "name": "ROLE_USER"
        }
      ]
    },
    {
      "id": 2,
      "username": "string1",
      "email": "string1@gmail.com",
      "roles": [
        {
          "id": 2,
          "name": "ROLE_ADMIN"
        },
        {
          "id": 1,
          "name": "ROLE_USER"
        }
      ]
    }
  ]
}
```

User delete işleminde var olmayan bir user silinmeye çalışılırsa user yok message ı döndürülür.
Bu işlemlere ek kullanıcıya yeni roller atama işlemleride bulunmakta fakat aktif etmedim.
Kullanıcı silme işleminide sadece kendisi için yapma kontrolü getirilebilir veya admin kontrolü eklenebilir.

TestController

Hangi role sahip kullanıcıların hangi içeriğe erişebileceğini test edebiliriz bu işlemlerin kontrolü

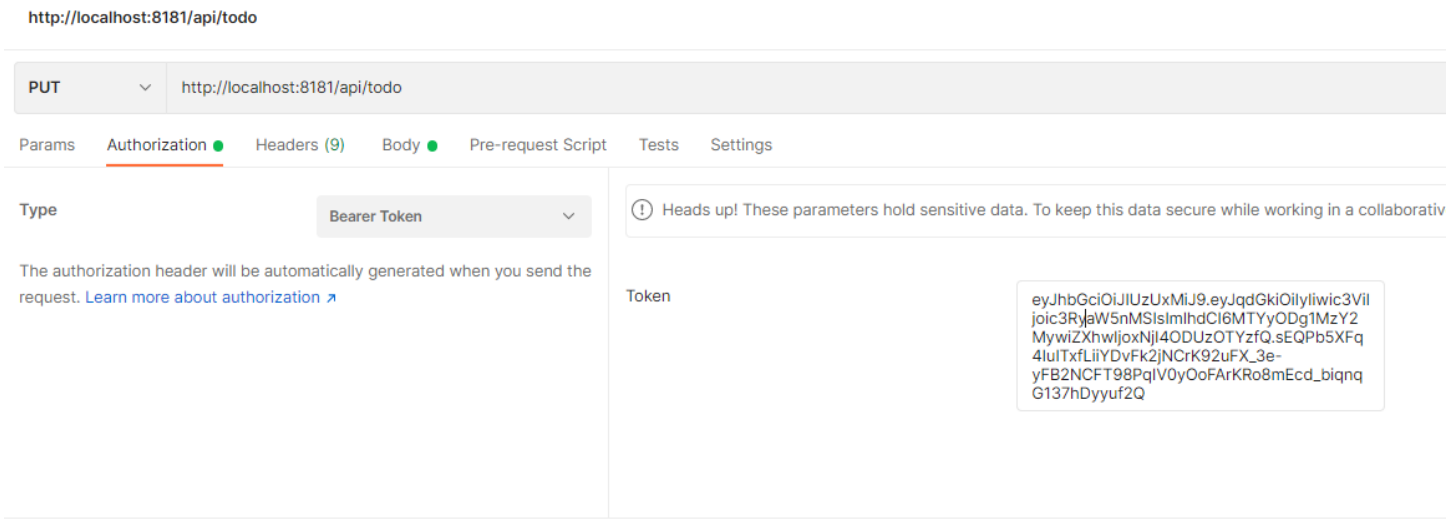
```
@PreAuthorize("hasAnyRole('USER')")
```

Şeklinde kontrol edilebilir.
Yetkisiz işlemlerde **"message": "Unauthorized"** döndürülür

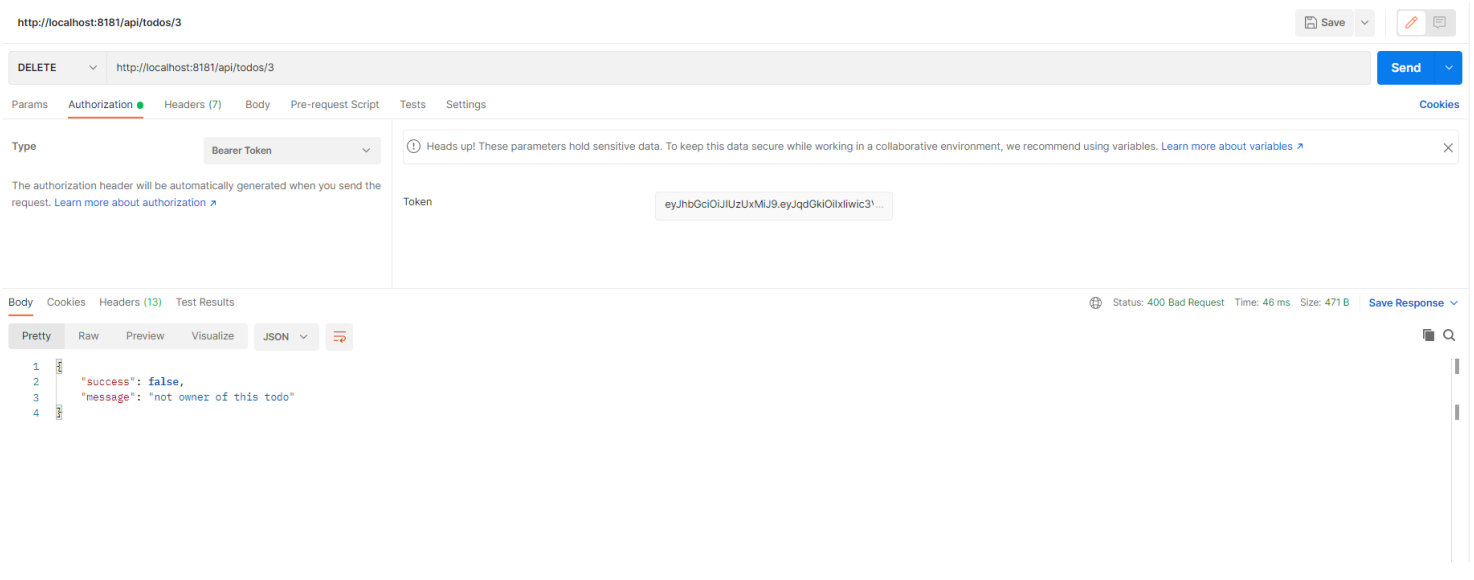
Swagger üzerinden test etmek için login yada register yapıldığında dönen token ile kilit butonuna basarak ilgili bölüme “Bearer {token}” yazarak test edebilirsiniz.

TodosController

Todo oluşturabilmek, todoları görmek, update etmek ve silmek için giriş yapılması gerekmekte ve gelen token Bearer tipi ile gönderilmeli.



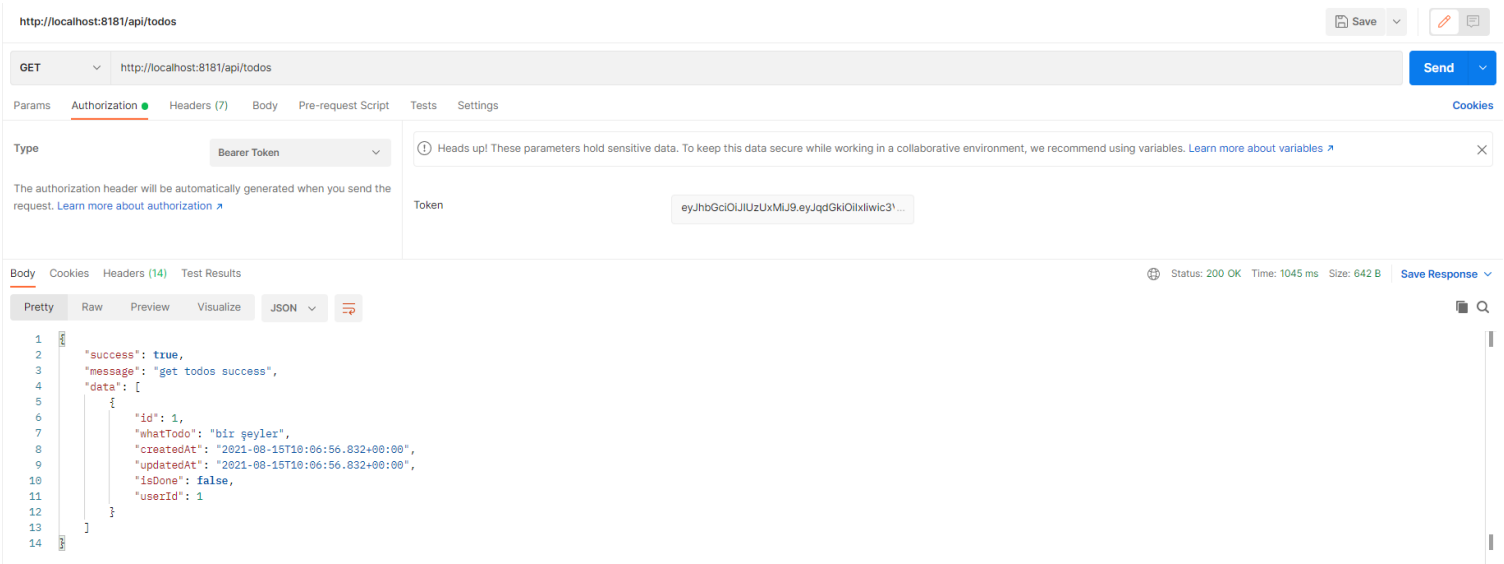
Giriş yapılmış fakat kullanıcı kendisine ait olmayan bir todo ya erişmek isterse todo sana ait değil message ı döndürülür. Örneğin, giriş yapan 1 id li user id si 2 olan user ın 3 id li todosunu silmeye çalışırsa



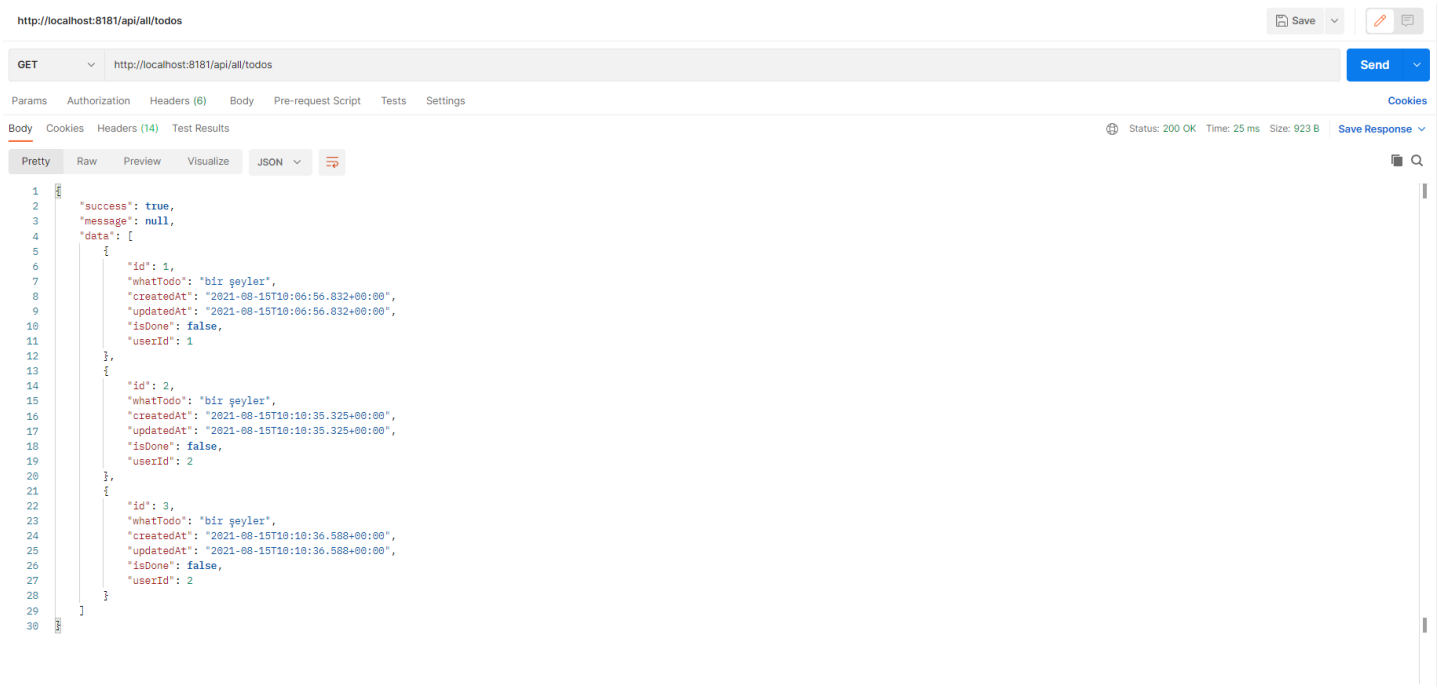
,

getAllTodos işlemi tüm todoları getirir user a göre ayrım yapmaz (test amacıyla)

*giriş yapmış olan 1 id li user todoları dönülüyor **api/todos***



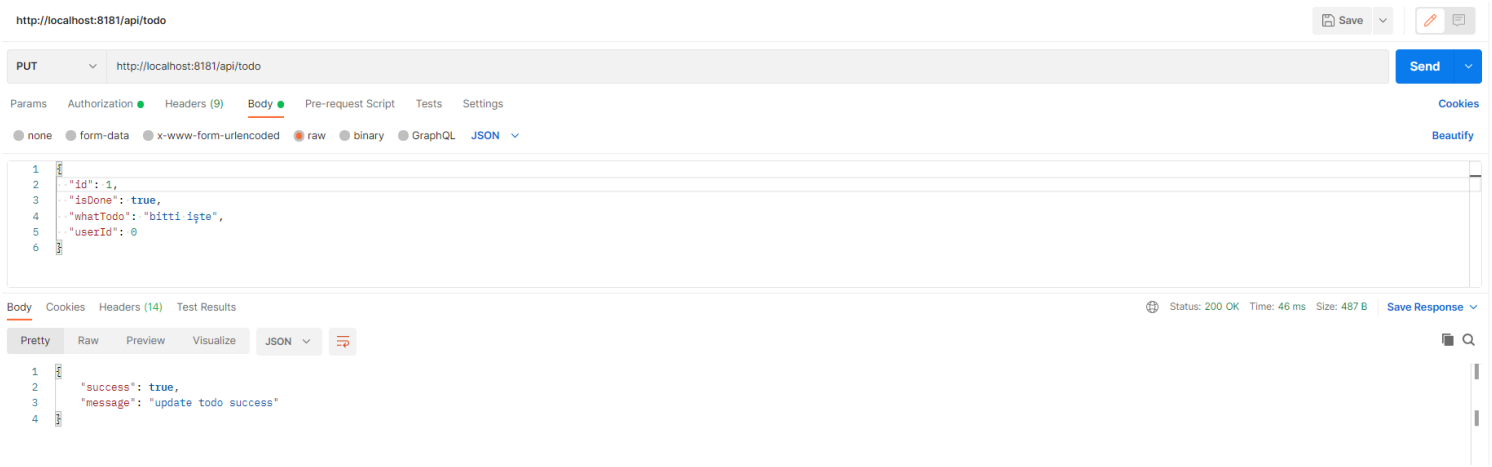
*tüm userların todoları dönülüyor **api/all/todos***

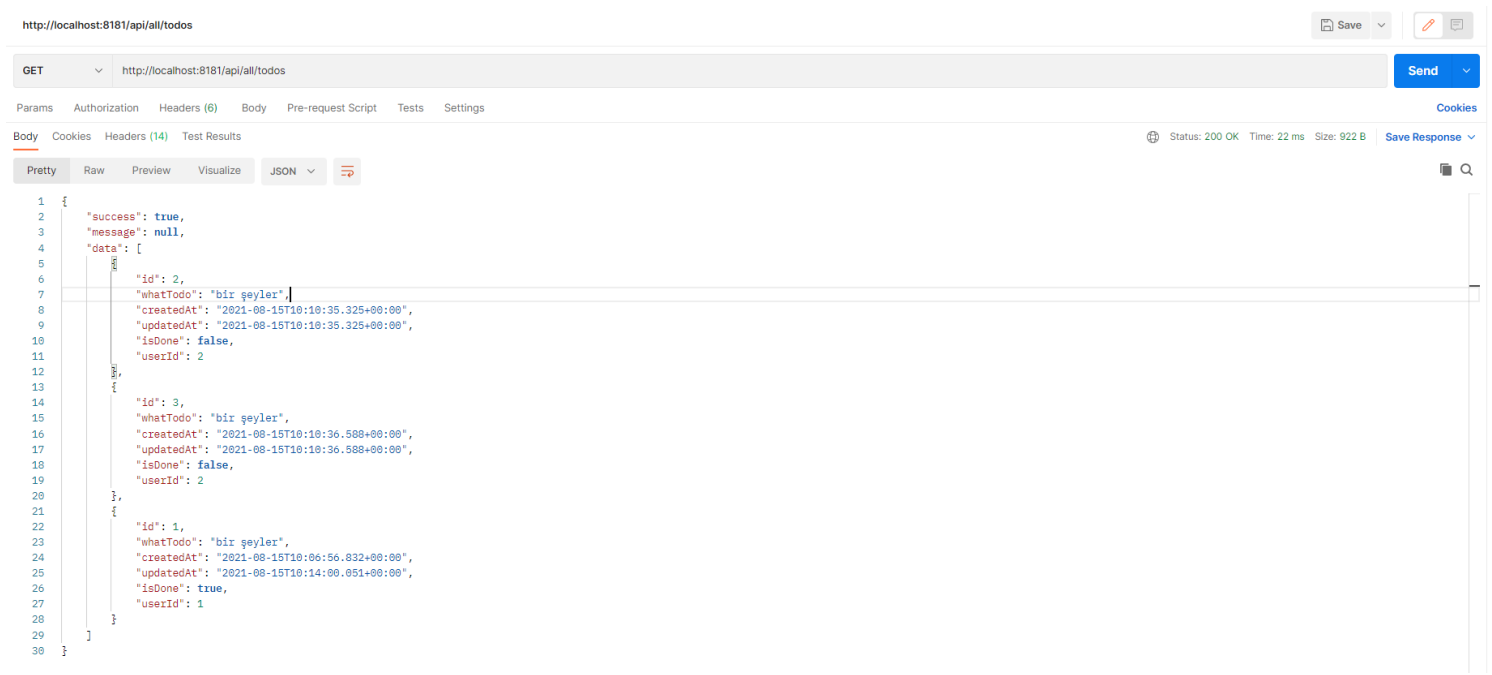


Giriş yapan kullanıcı id JWT ile belirlenir ve bu id için todolarla işlem yapılır.

Giriş yapılmamışsa yetki yok message ı döndürülür.

Update işleminde updatedAt saati değişir ve sadece isDone kısmı değiştirilebilir.





Diğer İşlemler

Bu işlemlere ek role create işlemi, role silme işlemi, rolleri getirme işlemi, userları getirme işlemi, user silme işlemi (silinen user a ait todolarda silinir)

UML Diagram


User
Todo
TodoService
TodoManager
Result
DataResult<T>
ErrorResult
SuccessResult
ErrorDataResult<T>
SuccessDataResult<T>
AuthResponse
AuthService
AuthManager
UserService
UserManager
RegisterRequest
RoleService
RoleManager
AccessToken
TokenUtils
JwtUtils
LoginRequest
Role
Messages
WebSecurityConfig
TodosController
MavenWrapperDownloader
FinalJavaBackendApplication
RolesController
AuthTokenFilter
TestController
UsersController
AuthsController
UserDao
AuthEntryPointJwt
CacheConfig
Roles
TodoDao
FinalJavaBackendApplicationTests
RoleDao

Local Üzerinde Çalıştırmak

Local de çalıştırmak için öncelikle database **PostgreSQL** kurulumu yapmamız gerekiyor. Ben bunu **Docker** üzerinden yaptım ve Docker üzerinden kurulumu anlatacağım.

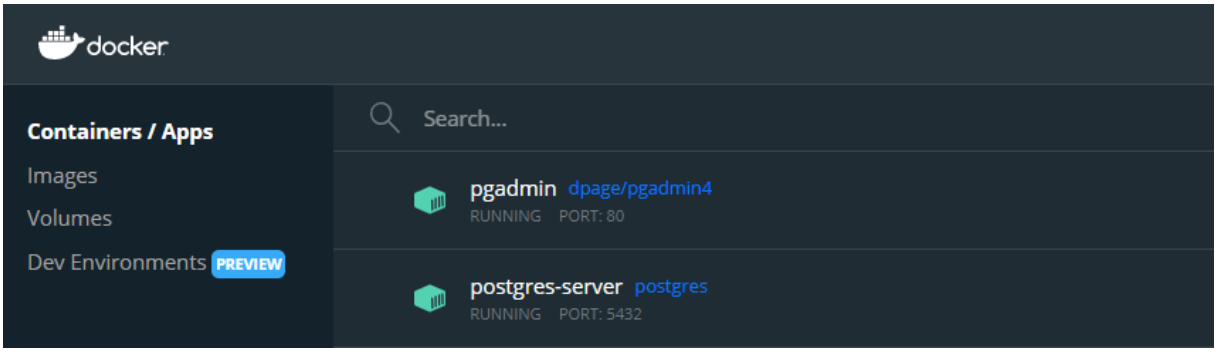
https://www.docker.com/products/docker-desktop Bu link üzerinden **Docker Desktop** indirilip kurulur.

Daha sonra **Windows Powershell** üzerinden

- ***docker run --name postgres-server -p 5432:5432 -e POSTGRES_USER=root -e POSTGRES_PASSWORD=1234 -d postgres***

- ***docker run -p 80:80 --name pgadmin -e PGADMIN_DEFAULT_EMAIL="ahmet.zoothii@gmail.com" -e PGADMIN_DEFAULT_PASSWORD="1234" dpage/pgadmin4***

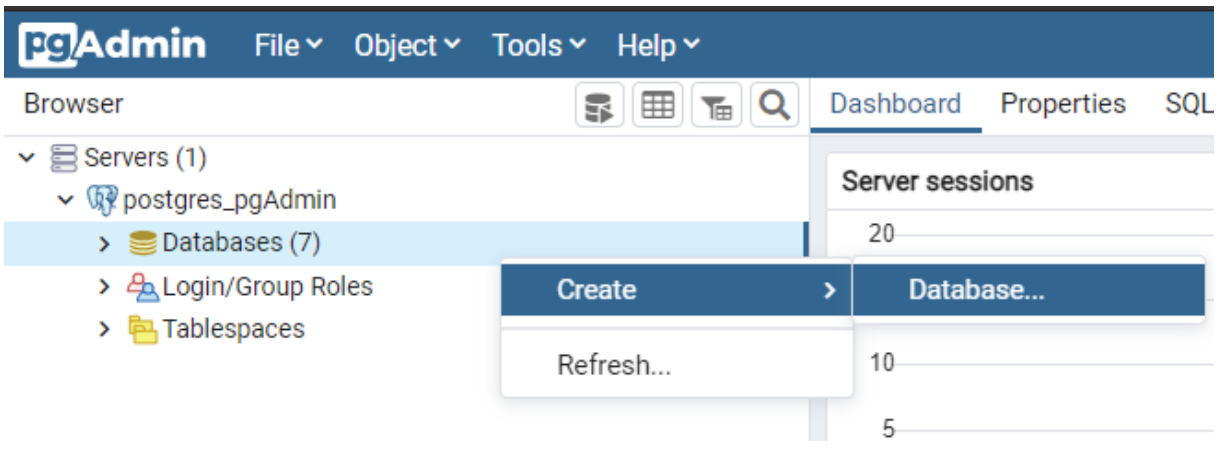
Yukarıdaki iki komutu çalıştırdığımızda Docker üzerinde 5432 portunda id si root şifresi 1234 olan bir server oluşturulacak daha sonra bu server ı ulaşmak dataları görebilmek için web browser üzerinde **pgAdmin** kurulumu başlatılacak port 80 üzerinden email ve şifreyi değiştirebilirsiniz fakat email bir email olmalı.

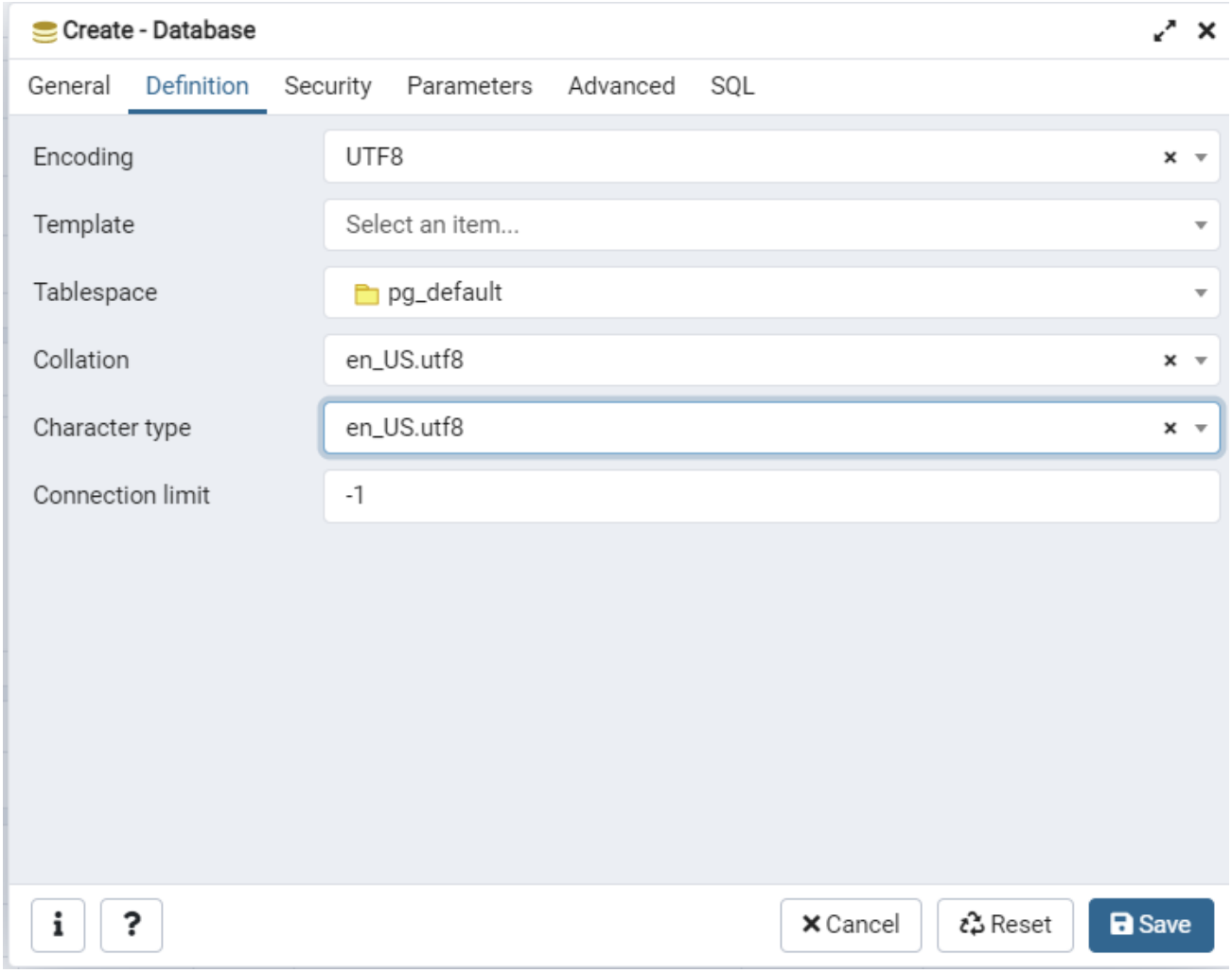


http://localhost:80/ daha sonra pgAdmin e bu URL üzerinden ulaşabiliriz. Email ve Password ile giriş yaptıktan sonra database server a bağlantı sağlamak için gerekli biligiler girilir.

Hostname olarak **host.docker.internal** girilmesi gerekmekte.

Daha sonra yeni bir database oluşturuyoruz ismini iwb_todo yapıyoruz ve definition kısmını ekran görüntüsündeki gibi giriyoruz.





Daha sonra github üzerinden çekmiş olduğumuz kodu **IntelliJ IDEA Ultimate** ya da **Eclipse** üzerinden çalıştırabiliriz.

Src altında resources dosyasının içinde application.properties dosyasında database için gerekli bilgilir tanımlı.

GITHUB PROJE

https://github.com/ZootHii/todo-java