API Documentation: Jolt Application

Version_goes-here - 9 Aug 2023

Contents

Getting Started With API

This document provides the following endpoints, methods and description of apis developed so far for mobile application called jolt.

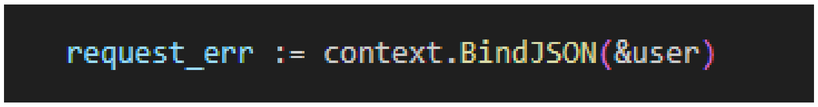
- **http://localhost:8081/Authenticate**
    - **POST METHOD**

- **http://localhost:8081/Register**
    - **POST METHOD**

API	Description
Register	This api allows stakeholders to provide the functionality of user sign up and add their details in a database from jolt mobile application.
Authenticate	Purpose of this api is to validate active users of jolt mobile application to grant them access to login the application and perform other features of it.

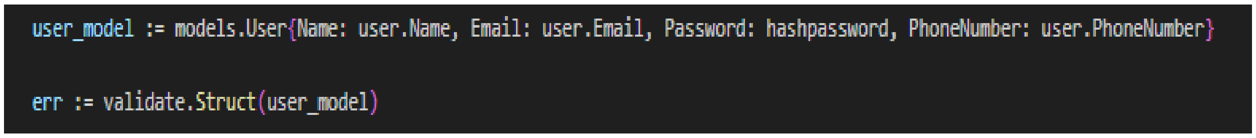
Technical Aspect of APIS

1) **Register:** Following api binds payload parameters to user object and this further goes to validation in order to check any missing values.



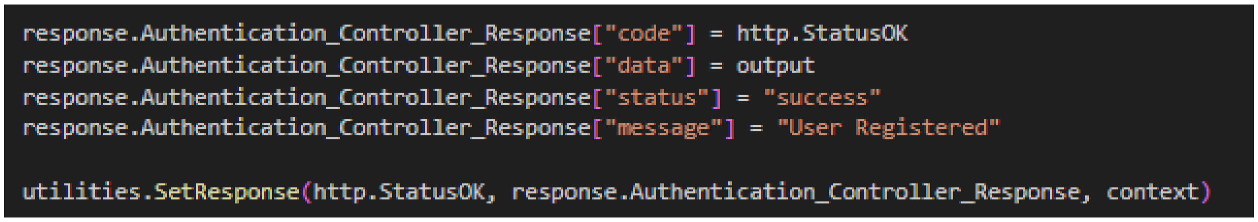
```
request_err := context.BindJSON(&user)
```

Validation applies using below statement:



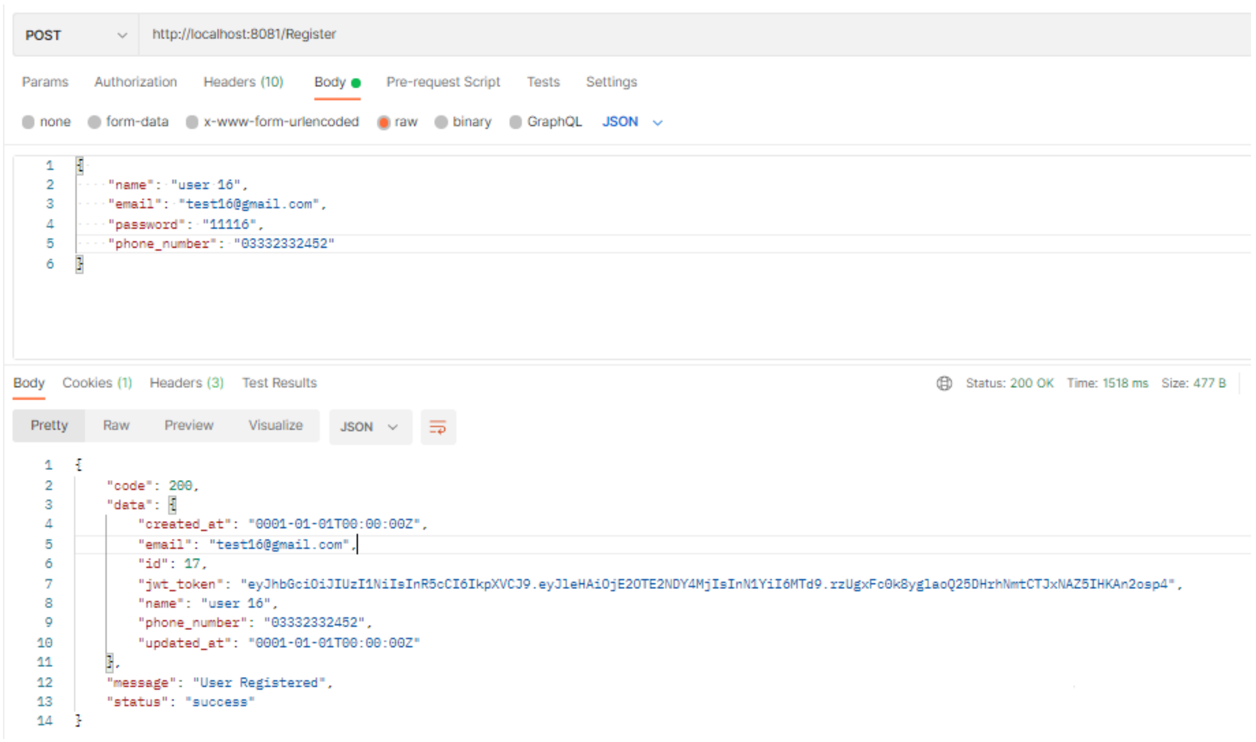
```
user_model := models.User{Name: user.Name, Email: user.Email, Password: hashpassword, PhoneNumber: user.PhoneNumber}

err := validate.Struct(user_model)
```

It then checks for various errors raised in response to functionalities in api. After successful insertion of record in database, a response is sent which includes user details alongwith token string using:



```
response.Authentication_Controller_Response["code"] = http.StatusOK
response.Authentication_Controller_Response["data"] = output
response.Authentication_Controller_Response["status"] = "success"
response.Authentication_Controller_Response["message"] = "User Registered"

utilities.SetResponse(http.StatusOK, response.Authentication_Controller_Response, context)
```

Example Postman Response (Registration)



POST http://localhost:8081/Register

Params · Authorization · Headers (10) · Body · Pre-request Script · Tests · Settings

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

```
{
    "name": "user 16",
    "email": "test16@gmail.com",
    "password": "11116",
    "phone_number": "03332332452"
}
```

Body · Cookies (1) · Headers (3) · Test Results — Status: 200 OK Time: 1518 ms Size: 477 B

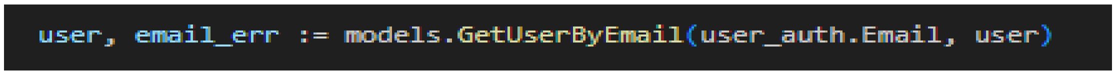
Pretty · Raw · Preview · Visualize · JSON

```
{
    "code": 200,
    "data": {
        "created_at": "0001-01-01T00:00:00Z",
        "email": "test16@gmail.com",
        "id": 17,
        "jwt_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE2OTE2NDY4MjIsInN1YiI6MTd9.rzUgxFc0k8yglaoQ25DHrhNmtCTJxNAZ5IHKAn2osp4",
        "name": "user 16",
        "phone_number": "03332332452",
        "updated_at": "0001-01-01T00:00:00Z"
    },
    "message": "User Registered",
    "status": "success"
}
```

2) Authenticate: Here binding of payload parameter happens in the same way as in register api but it only requires an email and password fields to authenticate the user. Following code snippet is for binding parameters to User_Authentication struct:

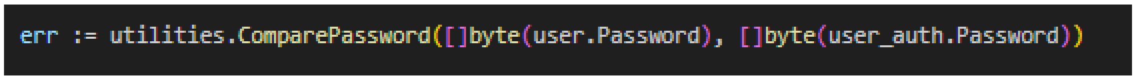
```
request_err := context.BindJSON(&user_auth)
```

A user is then fetched from database query using code snippet:



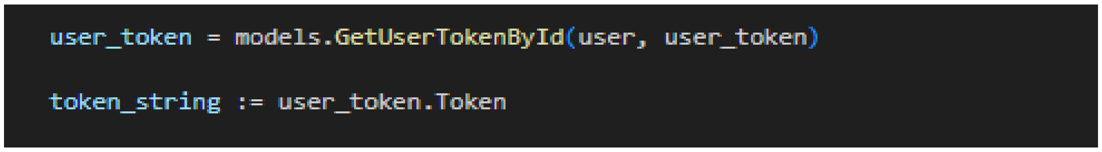
```
user, email_err := models.GetUserByEmail(user_auth.Email, user)
```

After then password is compared based on fetched user password which is in hashed format and using bcrypt package it validates user like:

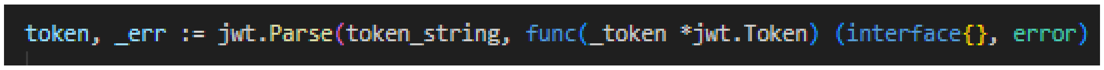


```
err := utilities.ComparePassword([]byte(user.Password), []byte(user_auth.Password))
```

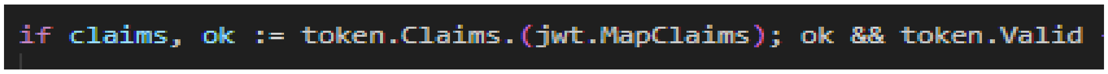
On successfully authenticating, a token is fetched from the database which was generated at the time of user registration to keep track of user session. This token is then processed using jwt package to parse it using the signing method and checked for its validity and expiry. Following snippets are sequence of preceding description:



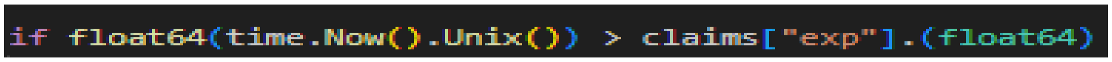
```
  user_token = models.GetUserTokenById(user, user_token)

  token_string := user_token.Token
```



```
token, _err := jwt.Parse(token_string, func(_token *jwt.Token) (interface{}, error)
```

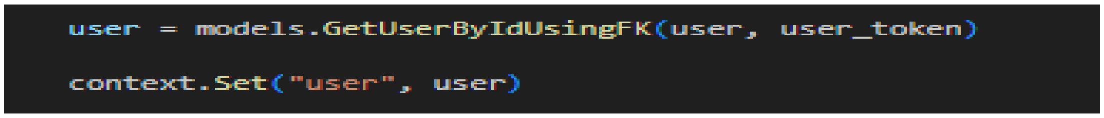


```
if claims, ok := token.Claims.(jwt.MapClaims); ok && token.Valid
```



```
if float64(time.Now().Unix()) > claims["exp"].(float64)
```

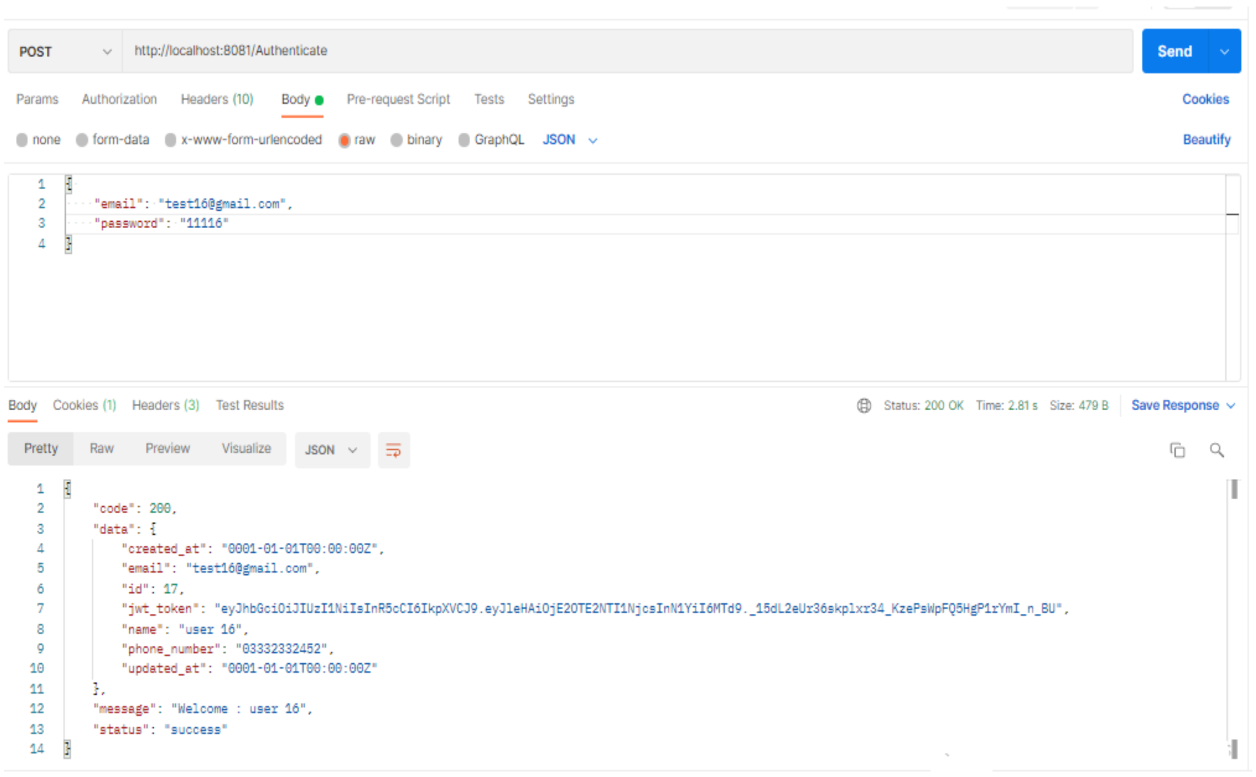
If everything works fine and token is valid and within expiry time then user is fetched from users table based on join condition using token id and as a response, user details with token information is passed as response with a welcome message.



```
  user = models.GetUserByIdUsingFK(user, user_token)

  context.Set("user", user)
```

```
response.Authentication_Controller_Response["code"] = http.StatusOK
response.Authentication_Controller_Response["data"] = output
response.Authentication_Controller_Response["status"] = "success"
response.Authentication_Controller_Response["message"] = "Welcome :" + " " + user.Name

utilities.SetResponse(http.StatusOK, response.Authentication_Controller_Response, context)
```

Example Postman Response (Authentication)



http://localhost:8081/Authenticate

```
{
    "email": "test16@gmail.com",
    "password": "11116"
}
```

Status: 200 OK   Time: 2.81 s   Size: 479 B

```
{
    "code": 200,
    "data": {
        "created_at": "0001-01-01T00:00:00Z",
        "email": "test16@gmail.com",
        "id": 17,
        "jwt_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE2OTE2NTI1NjcsInN1YiI6MTd9._15dL2eUr36skplxr34_KzePsWpFQ5HgP1rYmI_n_BU",
        "name": "user 16",
        "phone_number": "03332332452",
        "updated_at": "0001-01-01T00:00:00Z"
    },
    "message": "Welcome : user 16",
    "status": "success"
}
```