Clapou Alexandru
248-1

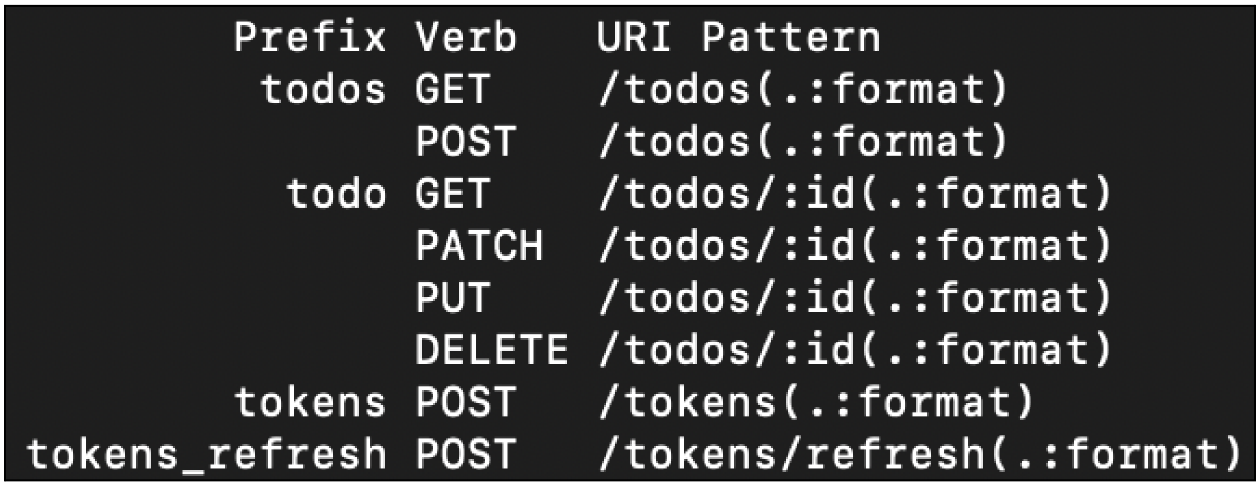
Securing a REST API

Securing a REST API is crucial to protect sensitive data and ensure the integrity of your application. In this tutorial, I will present a robust security approach for a REST API implemented in Ruby on Rails using Refresh Token Rotation. This method enhances security by regularly updating access and refresh tokens (after the access token expired), minimizing the risk of unauthorized access.

1. Prerequisites

- Basic token authentication knowledge
- REST APIs
- Basic RoR knowledge

To present the implementation of Refresh Token Rotation (RTR) and how it works I prepared a simple REST API with the following routes presented below.



```
         Prefix Verb   URI Pattern
          todos GET    /todos(.:format)
                POST   /todos(.:format)
           todo GET    /todos/:id(.:format)
                PATCH  /todos/:id(.:format)
                PUT    /todos/:id(.:format)
                DELETE /todos/:id(.:format)
         tokens POST   /tokens(.:format)
 tokens_refresh POST   /tokens/refresh(.:format)
```

Fig 1.1. Available routes in our API

Our database diagram is presented below.

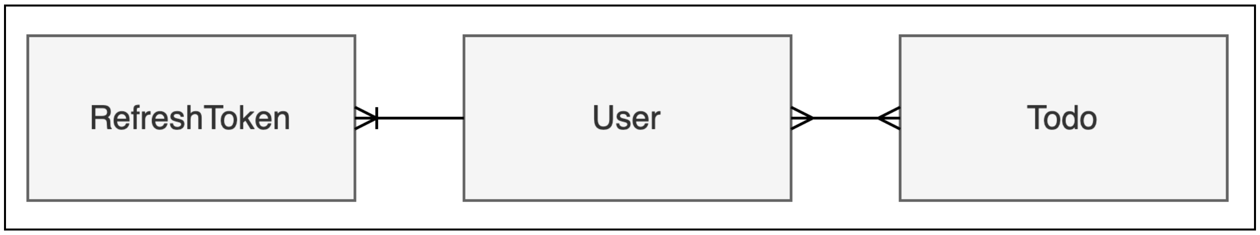


Fig 1.2. Database diagram

2. Refresh Token Implementation

After implementing the basic REST API (step which we skipped in this tutorial) we need to implement the Refresh Tokens.

In token-based authentication, a refresh token is a credential used to obtain a new access token after the original access token has expired. Access tokens are short-lived for security reasons, while refresh tokens typically have a longer lifespan.

2.1. What is Refresh Token Rotation?

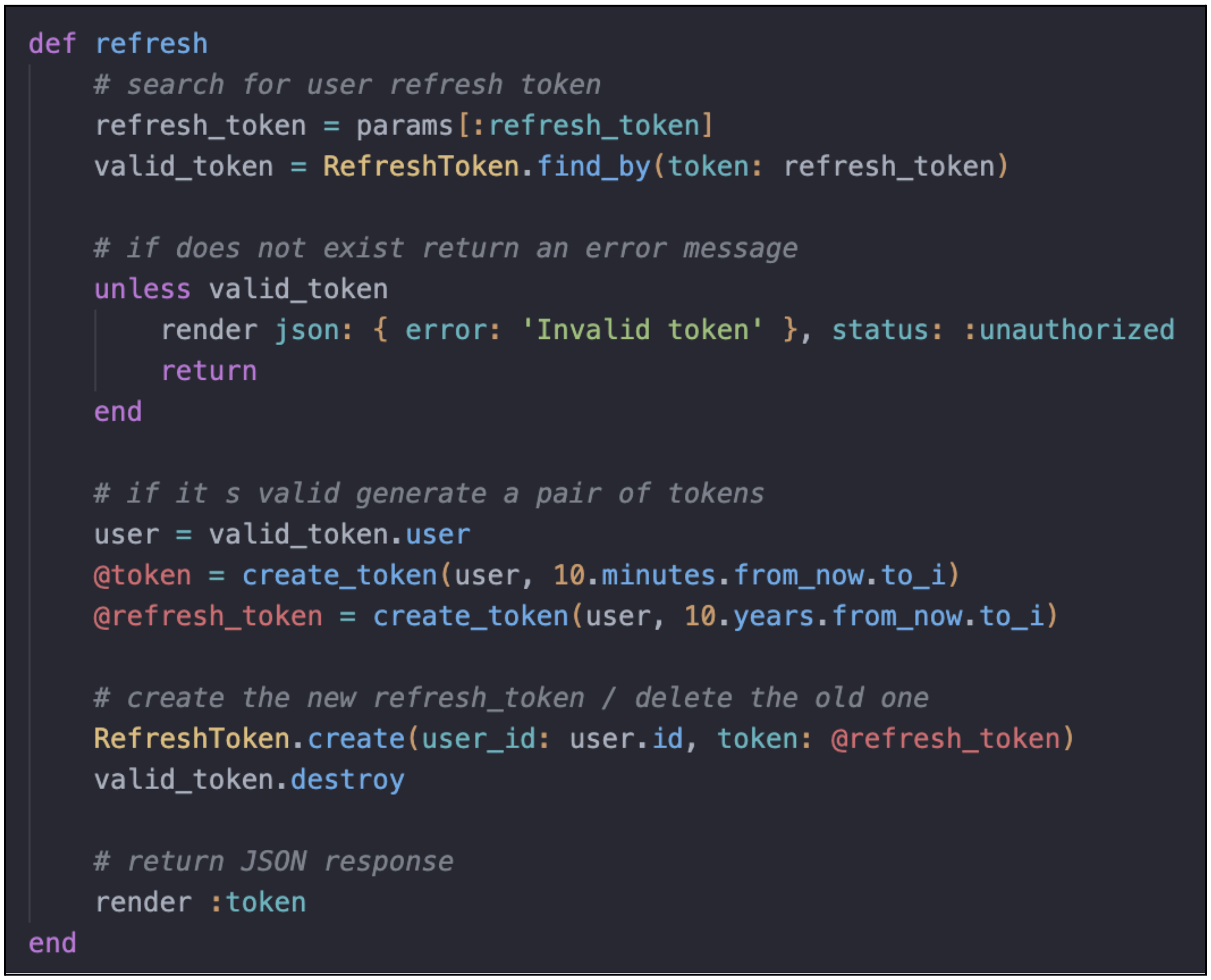
Refresh token rotation involves regularly issuing a new refresh token and invalidating the previous one. This practice helps mitigate the risk associated with long-lived refresh tokens. If a refresh token is compromised or if there are concerns about its security, rotating the refresh token ensures that even if an attacker gains access to an old token, it will soon become invalid.

Our RefreshToken table contains 2 attributes (beside id, timestamps): **user_id** and **token**. When we create a new RefreshToken we save it to the database, and delete the old one (having an additional attribute **invalid:boolean** could be helpful to track possible 'attackers' who have the old token).

The 'internal flow' of RTR is the following:

1. The user send the request to authenticate to the app
2. The API successfully return two tokens (**acces_token** and **refresh_token**)
3. The user send access_token until is expired.
4. When acces_token is expired the user request for a refresh.
5. On refresh, the app check the old refresh_token, and if it s valid it creates a new one and it returns a pair of <access_token, refresh_token>
6. User continues to send requests.

2.2. RTR implementation in Ruby



```ruby
def refresh
    # search for user refresh token
    refresh_token = params[:refresh_token]
    valid_token = RefreshToken.find_by(token: refresh_token)

    # if does not exist return an error message
    unless valid_token
        render json: { error: 'Invalid token' }, status: :unauthorized
        return
    end

    # if it s valid generate a pair of tokens
    user = valid_token.user
    @token = create_token(user, 10.minutes.from_now.to_i)
    @refresh_token = create_token(user, 10.years.from_now.to_i)

    # create the new refresh_token / delete the old one
    RefreshToken.create(user_id: user.id, token: @refresh_token)
    valid_token.destroy

    # return JSON response
    render :token
end
```

Fig 2.2.1. refresh action

2.3. Testing RTR

Now, in order to make sure it works as expected we will have some test cases:

2.2.1. Invalid Credentials

```
Request:
curl -X POST -H "Content-Type: application/json" -d '{"name":"wrong-name"}'
http://localhost:3000/tokens/
Response
"error":"Invalid credentials"}
```

2.2.1. Expired Token

```
Request
curl -X GET -H "Authorization: Bearer
eyJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoxLCJleHAiOjE3MDY3MzkxNDl9.Nhaj9TPHMqL1oQPQUD5s-pJz
bp0YTYMqPf_QBH4De5M" http://localhost:3000/todos/4
Response
{"error":"Expired Token"}
```

2.2.1. Invalid Token

```
Request
curl -X GET -H "Authorization: Bearer
eyJhbGciOiJIUzI1rNiJ9.eyJ1c2VyX2lkIjoxLCJleHAiOjIwMjIzNDk3NDN9.tpmbIgPdKzoezDvkHht-8t1
zKT4OBXUcC-91DRXnOCg" http://localhost:3000/tokens/refresh
Response
{"error":"Invalid Token"}
```

2.2.1. Unauthorized

```
Request
curl -X GET http://localhost:3000/todos/4
Response
{"error":"Unauthorized"}
```

2.2.1. Success Request

```
Request
curl -X GET -H "Authorization: Bearer
eyJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoxLCJleHAiOjE3MDY3Mzk0Nzh9.m7l_XeXTfkXnvA1308umnpwA
eWs5DBkJyu6lyULNFBM" http://localhost:3000/todos/4
Response
{
  "account": {
    "id": 4,
    "user_id": 1,
    "name": "Todo-User-Main-3",
    "created_at": "2024-01-31T19:03:19.270Z",
    "updated_at": "2024-01-31T19:03:19.270Z"
  }
}
```