7. Implementing and Testing the API

7.1 Introduction

This chapter discusses the implementation of the API. It justifies decisions taken and discusses tests undertaken to ensure the API works as required. Finally, the conclusion mentions steps taken to implement the API and evaluates the API.

7.1.1 Overview

When a coachee logs in, the login details are validated. If successful, an authentication token is sent, which will be used in subsequent requests for validation till the coachee signs out. This authentication token is reused if the coachee signs in with another device.

The functional and non-functional requirements are listed below and their implementations are examined in subsequent sections.

Functional Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Priority** | **API**  **Version** | **Description and Justification of Functional Requirement** |
| 15 | H | 1 | **API should encrypt authentication token**  To prevent a hacker or an unauthorised person from accessing the authentication token |
| 24 | H | 1 | **The API should authenticate the users**  This ensures only authorised Coachees can access data on the system |
| 25 | M | 1 | **The API should send the appropriate error message when it can't perform a function**  This informs the Coachee of the reason a function can't be performed. It also improves user experience |

Non-functional Requirements

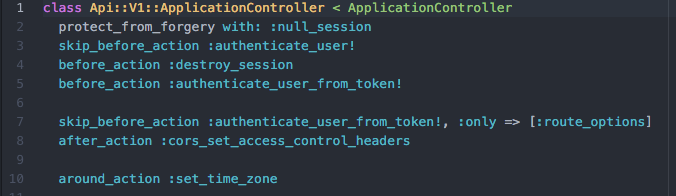
|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Priority** | **API**  **Version** | **Description and Justification of Non-Functional Requirement** |
| 22 | H | 1 | **The API should be well tested**  This confirms the API works as expected and the tests serve as regression tests |

7.2 Implementation of Functional Requirements

The API’s controllers extend the Api::V1::ApplicationController class located at /app/controllers/api/v1/application\_controller.rb. This, in turn, extends the ApplicationController class, located at /app/controllers/application\_controller.rb, as seen in Fig 7.1 on line 1 -- the Api::V1::ApplicationController extends ApplicationController so as to inherit ApplicationController’s public methods.

The meaning of the filters shown in Fig 7.1 are given below:

* before\_action: signifies an action or method that’s called before the controller is accessed
* skip\_before\_action: indicates an action or method should be skipped e.g. line 3 signifies that the “authenticate\_user! “ method should be skipped. The “authenticate\_user!” method is inherited from ApplicationController and is used to authenticate users signing in on CiaB’s website, therefore it should not be used to authenticate API requests. The skip\_action could have conditions specifying when an action should be skipped e.g. line 7 signifies that when the params object has a route\_options key, the “authenticate\_user\_from\_token!” method should be skipped.
* around\_action: signifies an action or method that’s to be called before and after a controller is accessed
* protect\_from\_forgery: sets the session to NULL thereby requiring the authentication of every request

Fig 7.1: API’s application\_controller filters

All the API’s controllers except the SessionsController (Api::V1::SessionsController) authenticate every request by the authentication token by calling the “authenticate\_user\_from\_token” method.

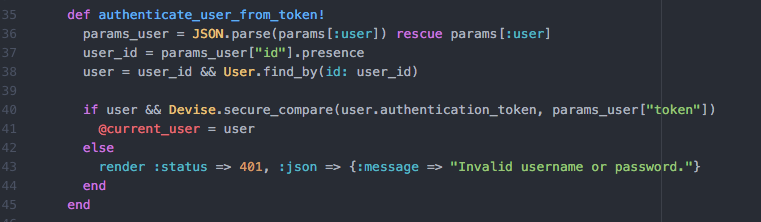


Fig 7.2: authenticate\_user\_from\_token!

The “authenticate\_user\_from\_token!” method expects the incoming request to have an authentication token and attempts to validate the coachee’s identity by the token using a plugin called Devise.

Line 36 of Fig 7.2 obtains the user JSON object in the request, line 37 checks for the presence of the user ID and line 38 searches for the user in the database by the user ID.

In code block from lines 40 to 45, using the devise plugin the authentication token sent in the request is compared with that saved in the database. If both matches then the request proceeds to the designated controller but if it fails an “Invalid username or password” message is returned.

7.2.1 The API should authenticate the users

The config/routes.rb determines which controller handles a particular HTTP request.

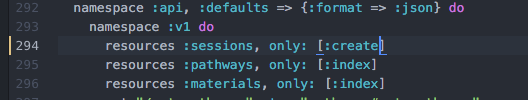


Fig 7.3: Screenshot of config/routes.rb

Line 294 above indicates that all POST requests to /sessions should be handled by the “create” method of the Api::V1::SessionsController.



Fig 7.4: SessionsController class (/app/controllers/sessions\_controller.rb)

Line 2 above asks that the “authenticate\_user\_from\_token!” method be not called before the controller is accessed. This makes sense, as at the time the coachee is logging in, the coachee doesn’t have an authentication token.

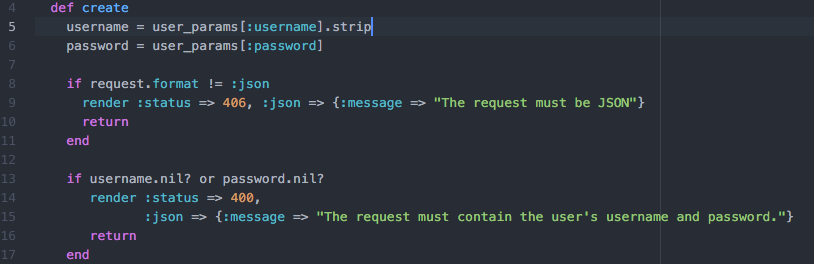


Fig 7.5 create method of SessionsController

In Fig 7.5, line 5 strips leading and trailing spaces in the username from the user JSON object whilst line 6 obtains the password from the user JSON object. Stripping of leading and trailing spaces in the username enables coachees to login with leading and trailing spaces in their username – a requirement specified in the Requirements Specification. If the format of the request is not in JSON then the error message “The request must be JSON” is returned (lines 8 to 11). If the user JSON object doesn’t contain the username and/or password, then the error message “The request must contain the user's username and password” is returned (lines 13 to 17).

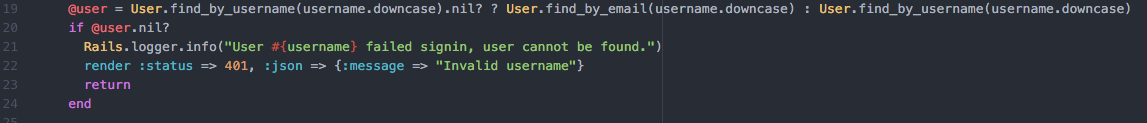


Fig 7.6: Lines 19 to 24 of “create” method

Line 19 of the “create” method uses the ternary operator to check for a coachee by the coachee’s username or email. This check enables a coachee to be able to sign in with either the coachee’s username or email address – a requirement specified in the Requirements Specification. If a null object is received then the “Invalid username” message is returned.

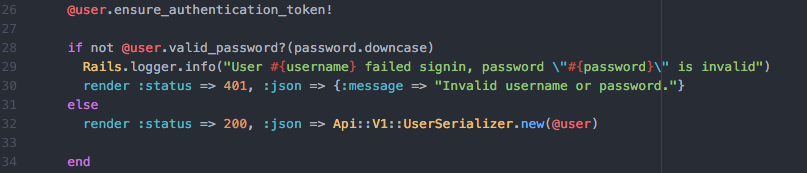


Fig 7.7: Lines 26 to 34 of create method

Line 26 (fig 7.7) checks for the presence of an authentication token in the user database object. If there’s none, one is created and persisted (see ‘app/models/user.rb’). Lines 28 to 34 check if the submitted password is valid, if it isn’t valid the error message “Invalid username or password” is returned. The user database object is serialised by the Api::V1::UserSerializer (located at /app/serializers/sessions\_controller.rb) if the submitted password is valid.

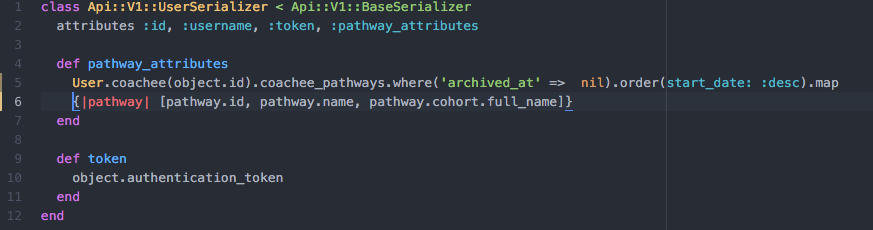


Fig 7.8: Api::V1::UserSerializer

Line 2 dictates the attributes of the user database object that will be returned back in the JSON response. As can be seen in fig 7.9, the user database object doesn’t have the token and pathway\_atrributes specified on line 2 of fig 7.8. The attributes not present in the user database object (token and pathway\_atrributes) are defined in the Api::V1::UserSerializer class where “object” stands for an instance of the User class (@user) that was passed on line 32 of fig 7.7.



Fig 7.9: User database object

7.2.1.1 Tests and Results

Testing was done using the Mac OS X command “curl”. “curl” is a command line tool used to transfer data to and from a server using a supported protocol. It works without any user interaction. The API was initially tested on the author’s local server, “local.ciabos.dev” before it was deployed to “ci-ciabos-pr-276.herokuapp.com”.

1. Requirement 24

*“The API should authenticate the users”*

Test:

*curl -v -H "Accept: application/json" -H "Content-type: application/json" -X POST -d '{"user":{"username":"fred coachee ","password":"66cooking"}}'* [*http://local.ciabos.dev/api/v1/sessions*](http://local.ciabos.dev/api/v1/sessions)

Result:

{

"user": {

"id": 24372,

"username": "fred coachee",

"token": "c5qdL-t12C11CGMdxUbc",

"pathway\_attributes": [

[

28599,

"New Programme for mypage",

"Dummy Test Cohort One"

],

[

28594,

"Sample programme for mypage",

"Dummy Test Cohort One"

],

[

28593,

"sort self learning programme",

"Dummy Test Cohort One"

]

]

}

}

Evaluation of result

The coachee is authenticated and details of the coachee’s pathways are sent in the response.

2. Requirement 25

*“The API should send the appropriate error message when it can't perform a function”*

Test:

*curl -v -H "Accept: application/json" -H "Content-type: application/json" -X POST -d '{"user":{"username":"fred coachee ","password":"password"}}'* [*http://local.ciabos.dev/api/v1/sessions*](http://local.ciabos.dev/api/v1/sessions)

Result:

{

"message": "Invalid username or password."

}

Evaluation of result

Appropriate error message is sent

7.3 Conclusion

The author performed an extensive research on how to create a Rails API. After some mock trials to ensure the API was working, the author embarked on building the API. The author created a Git branch in the “coach\_in\_a\_box” directory and worked on this branch whilst implementing the API.

Routes are defined in config/routes.rb and matched to corresponding controllers, which process the request’s parameters and control the flow of logic. The controllers can optionally format the returned JSON object by calling the serializers.

Each request was tested using “curl” from the Mac OS X terminal and the returned JSON object was formatted using online JSON editors like JSON Editor (<http://www.jsoneditoronline.org/>) and JSLint (<http://www.jslint.com/>).

Debugging was done by observing the server logs for errors and using a Rails plugin called “pry-remote” (<https://github.com/Mon-Ouie/pry-remote>). “pry-remote” pauses execution of the API and enables one to inspect the value of variables and test defined methods from the terminal. The Rails console, gotten by the ‘rails c’ terminal command, was also useful for debugging. The Rails console enables interactions with the API from the terminal.

The authentication used in the API is a simple token-based authentication -- though a more robust one may be used in production. Also, at the time of writing this report, the API has not been well tested using unit tests (Requirement 22) and the authentication token is not encrypted (Requirement 15), raising security concerns.

A token was chosen because the author didn’t want to reveal the password in subsequent communications between the UI and the API. Ideally, the password should be encrypted at the client-side before it’s sent to the API but that has not been implemented. Encrypting the token will give it the same level of security implemented on the password.

This implementation uses an object-oriented approach, where methods common to all controllers are placed in the Api::V1::ApplicationController class whilst methods specific to a controller are placed in the controller or the controller overrides an inherited method.