**API WALKTHROUGH**

**Mega Connect technologies PVT LTD.**

Backend Developer API Walkthrough Document: -

Document Version: 1.0

Date: 13/06/2023

Table of Contents:

1. Introduction

2. API Details

3. Authentication and Authorization

4. Use cases

5. Error Handling

6. Conclusion

1. Introduction

- **Purpose:**

Importance for backend developers:

* On boarding and Familiarization: The walkthrough document helps onboard new backend developers by providing them with a clear understanding of how the signup and signin APIs work. It familiarizes them with the API endpoints, request methods, authentication requirements, and response formats.
* Consistency and Best Practices: The document ensures that backend developers adhere to consistent practices and standards when implementing the signup and signin APIs. It provides guidelines on request parameters, headers, and error handling, promoting code consistency across the development team.
* Time and Effort Savings: By providing a comprehensive guide, the walkthrough document reduces the time and effort required for backend developers to understand the intricacies of the signup and signin APIs. It acts as a ready reference, allowing developers to quickly access the necessary information and troubleshoot issues efficiently.
* Collaboration and Communication: The document serves as a communication tool between backend developers, project managers, and other stakeholders. It helps foster collaboration by providing a shared understanding of the signup and signin APIs, facilitating effective communication and teamwork.
* Maintenance and Updates: As the signup and signin APIs evolve over time, the walkthrough document can be updated to reflect any changes. This ensures that all backend developers are aware of the modifications and can make the necessary adjustments in their code accordingly.

In summary, the API walkthrough document for the signup and signin APIs plays a crucial role in facilitating efficient development, promoting consistency, and ensuring effective collaboration among backend developers.

**- API Overview:**

**A. MongoDB Database:**

* MongoDB is used as the database to store user-related information such as usernames, passwords, and email addresses.
* It provides a flexible and scalable NoSQL database solution for storing and retrieving user data.

**B. User Signup:**

* Users can sign up by providing their desired username, password, and email address.
* The API validates the input, checks if the username or email is already registered, and stores the user information securely in the MongoDB database.
* Additionally, the API may generate and send an email verification link to the provided email address.

**C. User Sign in:**

* Users can sign in by providing their username or email and corresponding password.
* The API verifies the credentials against the stored user data in the MongoDB database.
* Upon successful authentication, the API generates a JWT (JSON Web Token) that includes the user's identity and authorization details.
* The JWT can be used for subsequent authenticated API requests.

**D. Forgot Password:**

* If a user forgets their password, they can initiate a password reset request.
* The API may generate a unique password reset token and send it to the user's registered email address.
* The user can then use the password reset token to verify their identity and set a new password.

**E. JWT Token and Email Verification:**

* The API utilizes JWT tokens for authentication and authorization purposes.
* When a user signs up or successfully verifies their email address, the API generates a JWT token containing the user's identity and authorization claims.
* The JWT token is typically included in the Authorization header of subsequent API requests to authenticate and authorize the user.

**F. Email Verification:**

* The API may incorporate an email verification process to ensure the provided email address is valid and belongs to the user.
* After signup, an email verification link or token is sent to the user's email address.
* The user clicks the verification link or provides the token to confirm the ownership of the email address.
* Upon successful verification, the API can enable the user's account and grant access to restricted features.

**- Audience:**

Specify the target audience for this document, which is backend developers and frontend developers as well.

**2. API Details**

**Api-host={{localhost:8000}} //local endpoint**

**Signup API:**

* URL: **http://{{api-host}}/api/signup**
* Method: **POST**
* **Request Body:**

{

"name": "User Name",

"email": “user\_mail\_id@gmail.com”,

"password": "password123"

}

* **Response Body (Success):**

{

* {"message":: "Signup success. Please signin."  
  }
* **Response Body (Error):**

{

* {  
   "error":: "Email is taken"  
  }

**Account Activation API. (( FROM MAIL ID))**

* + URL: **http://{{api-host}}/api/account-activation**
  + Method: **POST**
  + Request Body:

{

* + {  
     "token": "activation\_token"
  + }
  + Response Body (Success):

{

* + {  
     "message": "Signup success. Please signin."  
    }
  + Response Body (Error):
  + {
  + “error": "Expired link. SignUP again"  
    }
* **Sign in API:**
  + URL: **http://{{api-host}}/api/signin**
  + Method: **POST**
  + **Request Body:**
  + {{"email": “user\_mail\_id@gmail.com”,

"password": "password123"}

* + **Response Body (Success):**

{

"token": "access\_token",

"user": {

"\_id": "user\_id",

"name": "John Doe",

"email": “user\_mail\_id@gmail.com”,

"role": "user"

}}

**Response Body (Error):**

* + **{**
  + {  
     "error": "User with that email does not exist. Please signup"  
    }
  + or  
    or {
  + error": "Email and password do not match"  
    }
* **Update Password API:**
  + URL: **http://{{api-host}}/api/user/update**
  + Method: **PUT**
  + Authorization:
  + Bearer Token {eyJhbGciOiJIUz....}
  + **Request Params:** None
  + **Request Body:**
    - **name** (string, required) - The updated name of the user
    - **password** (string, optional) - The updated password of the user (minimum 6 characters)
  + **Response Body:** The updated user object (excluding **hashed\_password** and **salt** fields)
  + **Action Method:** **update**

{

"name": "Name",

"password": "newpassword"

}

* + **Get user details:**

**URL : http://{{api-host}}/api/user/:id**

* + **Method:** GET
  + **Request Params:** **id** (The ID of the user to retrieve)
  + **Request Body:** None
  + **Response Body:** The user object (excluding **hashed\_password** and **salt** fields)
  + **Action Method:** **read**
  + The response will contain the user object if found, or an error message if the user is not found.

* + **Forgot Password API:**
  + URL: **http://{{api-host}}/api/forgot-password**
  + Method: **POST**
  + Request Body:
  + {  
     "email": "user\_mail\_id@gmail.com"  
    }
  + Response Body (Success):
  + {  
     "message": "Email sent with password reset instructions."  
    }
  + Response Body (Error):
  + {  
     "error": "User with that email does not exist"  
    }
* **Reset Password API:**
  + URL: **http://{{api-host}}/api/reset-password**
  + Method: **PUT**
  + **Request Body:**

{

* + {  
     "resetPasswordLink": "reset\_password\_token",  
     "new Password": "new\_password123"  
    }
  + **Response Body (Success):**

**{**

* + {  
     "message": "Great! Now you can login with your new password"  
    }
  + Response Body (Error):

{

* + {  
     "error": "Expired link. Try again"  
    }

Or

{

* + {  
     "error": "Something went wrong. Try Later"  
    }

**dependency to login with google in front-end:**

1.tokenId generated when user click on mail id of google pop-up to login through gmail.

**library : "react-google-login": "^5.2.2"**

2. Add Google\_client\_id= 1014560470948-neud0e44fqc1pbtjff1uvbnfgajn4n6o.apps.googleusercontent.com in project directory of react in .env

3. API FOR LOGIN WITH GOOGLE

request body for googleLogin api

{  
 "idToken": "<replace-with-your-google-id-token-generated in front-end>"  
 }

response will be this

{  
 "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",  
 "user": {  
 "\_id": "123456789",  
 "email": "[example@example.com](mailto:example@example.com)",  
 "name": "USER NAME",  
 "role": "user"  
 }  
 }

url is

<http://techmega.cloud:8000/api/google-login>

ex:

method: 'POST',

url: `[http://techmega.cloud:8000/api/google-login`](http://techmega.cloud:8000/api/google-login%60),

data: { idToken: response.tokenId }

**FACEBOOK LOGIN API**

* Method: POST
* URL: [http://techmega.cloud:8000/api/facebook-login](http://localhost:3000/api/facebook-login)
* Headers:
  + Content-Type : application/json
* Request Body (raw JSON):

{ "userID": "1026494898361323",  
 "accessToken": "<your-facebook-accessToken>"  
}

In the above example, replace <your-facebook-userID> : REACT\_APP\_FACEBOOK\_APP\_ID=1026494898361323 and <your-facebook-accessToken> which is generated when you click on login with facebook button.

If everything goes well on server-side then below is what response would look like in front-end

**Response**

{ "token": "<jwt-token>",  
 "user": {  
 "\_id": "<user-id>",  
 "email": "<user-email>",  
 "name":"<user-name>,  
 "role":"<user-role>"  
 }  
}

## **LOGIN WITH OTP : Send-Otp**

**Endpoint:** **http:localhost:8000/api/send-otp**

**Method:** POST

**Description:** This API endpoint is used to send a One-Time Password (OTP) to a mobile number for account verification.

### **Request Body**

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| mobileNumber | string | The mobile number to send OTP |

### **Response Body**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| success | boolean | Indicates whether the OTP was sent successfully |
| message | string | A message indicating the status of the OTP sending |

### **Example Request**

POST /api/send-otp

Content-Type: application/json

{

"mobileNumber": "1234567890"

}

### **Example Response**

HTTP/1.1 200 OK

Content-Type: application/json

{

"success": true,

"message": "OTP sent successfully"

}

### **Error Responses**

In case of an error, the API will respond with a status code of 500 and the following response body:

HTTP/1.1 500 Internal Server Error

Content-Type: application/json

{

"success": false,

"message": "Failed to send OTP"

}

## **API Documentation: verifyOtp**

**Endpoint: http:techmega.cloud:8000/api/verify-otp**

**Method:** POST

**Description:** This API endpoint is used to verify the One-Time Password (OTP) provided by the user for account verification.

### **Request Body**

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| mobileNumber | string | The mobile number to verify |
| otp | string | The OTP provided by the user |

### **Response Body**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| success | boolean | Indicates whether the OTP verification was successful |
| message | string | A message indicating the status of the OTP verification |

### **Example Request**

POST /api/verify-otp

Content-Type: application/json

{

"mobileNumber": "1234567890",

"otp": "123456"

}

### **Example Response**

HTTP/1.1 200 OK

Content-Type: application/json

{

"success": true,

"message": "OTP verification successful"

}

### **Error Responses**

In case of an error or an invalid OTP, the API will respond with a status code of 500 and the following response body:

HTTP/1.1 500 Internal Server Error

Content-Type: application/json

{

"success": false,

"message": "Failed to verify OTP"

}

HTTP/1.1 200 OK

Content-Type: application/json

{

"success": false,

"message": "Invalid OTP"

}

Please ensure that the **Content-Type** header of the request is set to **application/json** when making a request to this API endpoint.

### **DELETE ACCOUNT API**

### URL: http://techmega.cloud:8000/api/user/delete-Account **Headers**

* **Authorization**: Bearer **{JWT\_TOKEN}**

### **Request Body**

No request body is required for this API.

### **Response Body**

#### Successful Response

* **Status Code**: **200 OK**

**{**

**"message": "User deleted successfully"**

**}**

#### Unsuccessful Response

* **User Not Found**
* **Status Code**: **404 Not Found**

**{**

**"message": "User not found"**

**}**

**Server Error**

* **Status Code**: **500 Internal Server Error**

**{**

**"error": "Error deleting user"**

**}**

### **Description**

Deletes a user from the database. This API requires JWT token authentication. Make sure to pass the JWT token in the **Authorization** header as a Bearer token.

**curl -X DELETE "https://example.com/api/v1/users" \**

**-H "Authorization: Bearer {JWT\_TOKEN}"**

**3.Authentication and Authorization**

| Authentication | [Explain the authentication mechanism used by the API] |

| Authorization | [Describe the authorization process and any required permissions or roles needed to access the API] |

**4. Use Cases:**

* User registration and authentication for web or mobile applications.
* Secure access to user-specific resources and personalized content.
* Enforcing role-based permissions and authorization.
* Password reset and account recovery functionality.

**5. Error Handling**

When using the API for user authentication and authorization, various error scenarios can occur. Handling and interpreting these error responses is crucial for providing a smooth user experience. Here are some possible error scenarios and guidance on how to handle them:

A. Validation Errors:

- Error Scenario: Users provide invalid or incomplete input during signup or password reset.

- Handling: The API should validate user input and respond with specific error messages indicating the validation failures (e.g., missing required fields, invalid email format, weak password). The client-side application can display these error messages to the user for correction.

B. Duplicate User:

- Error Scenario: Users attempt to sign up with a username or email address that is already registered.

- Handling: The API should respond with an error message indicating that the username or email is already in use. The client-side application can display a message informing the user to choose a different username or initiate a password reset if they forgot their password.

C. Authentication Errors:

- Error Scenario: Users provide incorrect credentials during the sign in process.

- Handling: The API should respond with an error message indicating that the provided credentials are invalid. The client-side application can display an appropriate error message to the user, prompting them to re-enter their credentials or initiate a password reset if needed.

D. Email Verification Errors:

- Error Scenario: Users encounter issues with email verification, such as an expired or invalid verification token.

- Handling: The API should respond with an error message indicating the issue with email verification. The client-side application can inform the user about the error and provide options to resend the verification email or contact support for assistance.

E. Token Errors:

- Error Scenario: Users provide an expired or invalid JWT token for authentication.

- Handling: The API should respond with an error message indicating that the token is invalid or has expired. The client-side application can prompt the user to sign in again or initiate the token renewal process if applicable.

F. Server Errors:

- Error Scenario: Internal server errors occur, such as database connection issues or unexpected exceptions.

- Handling: The API should respond with an appropriate error message indicating the server error. The client-side application can display a generic error message and recommend trying again later or contacting support for assistance.

G. Rate Limiting:

- Error Scenario: Users exceed the allowed rate limit for API requests (e.g., too many failed login attempts).

- Handling: The API should respond with an error message indicating that the rate limit has been exceeded. The client-side application can inform the user about the rate limit and suggest waiting for a specified period before trying again.

In all error scenarios, it's important to provide clear and meaningful error messages to users, guiding them on how to resolve the issue. Error responses should include appropriate HTTP status codes (e.g., 400 for validation errors, 401 for authentication errors) to help distinguish between different types of errors. Additionally, logging and monitoring server-side errors can help identify and address any underlying issues promptly.

**6. Conclusion**

A. MongoDB Official Documentation:

- Official documentation for MongoDB, providing comprehensive guides, tutorials, and references on working with MongoDB as the database for the API.

- Website: [MongoDB Official Documentation] (https://docs.mongodb.com/)

B. JWT.io:

- A website dedicated to JSON Web Tokens (JWT), providing explanations, libraries, and tools for working with JWT tokens.

- Website: [JWT.io] (https://jwt.io/)

C. Express.js Documentation:

- Official documentation for Express.js, a popular Node.js web application framework commonly used for building APIs.

- Website: [Express.js Documentation] (https://expressjs.com/)

D. Nodemailer:

- Nodemailer is a module for Node.js applications that allows sending emails. It can be used for sending verification emails in the API.

- GitHub Repository: [Nodemailer](https://github.com/nodemailer/nodemailer)

E. Node.js Crypto Module:

- The Crypto module in Node.js provides cryptographic functionality, including hashing and encryption, which can be useful for securely storing passwords.

- Official Documentation: [Node.js Crypto Module](https://nodejs.org/api/crypto.html)

F. Sending HTTP Responses in Express.js:

- A guide on how to send HTTP responses using Express.js, including setting status codes, sending JSON responses, and handling errors.

- Article: [Sending HTTP Responses in Express.js](https://zellwk.com/blog/express-responses)

G. Error Handling in Express.js:

- Best practices for error handling in Express.js applications, including middleware for centralized error handling and graceful error responses.

- Article: [Error Handling in Express.js] (https://expressjs.com/en/guide/error-handling.html)

These resources should provide valuable insights, examples, and code snippets to assist backend developers in working with the described API.

- Contact Information: [Ajit [senapati - ajitsenapati02@techmega.cloud](mailto:senapati%20-%20ajitsenapati02@techmega.cloud),

Mahesh hiremath-maheshhiremath03@techmega.cloud]