Assignment3

Application API Specification

The Application API allows users to manage job applications in the system. This includes creating, retrieving, and deleting job applications. It also supports filtering by status and sorting by creation date.

Base URL

```
/applications
```

Endpoints

1. Get All Applications

GET `/`

Retrieve all applications for the authenticated user, with optional filtering by status and sorting by date.

Query Parameters

Parameter	Type	Description
`status`	String	Filter applications by status (e.g., PENDING, APPROVED, REJECTED).
`sortBy`	String	Field to sort by (default: `createdAt`).
`order`	String	Sort order, either `ASC` or `DESC` (default: `DESC`).

Response

- **200 OK**: Returns an array of applications.
- **404 Not Found**: No applications found for the user.

Example Response

```
[
    {
```

```
        "id": 1,
        "jobId": 101,
        "userId": 5,
        "status": "PENDING",
        "createdAt": "2024-12-01T10:00:00Z",
        "relatedJob": {
            "title": "Software Engineer"
        }
    }
]
```

2. Get Application by ID

GET `/:id`

Retrieve a specific application by ID for the authenticated user.

Path Parameters

Parameter	Type	Description
`id`	Integer	ID of the application to retrieve.

Response

- **200 OK**: Returns the application details.
- **404 Not Found**: Application not found or access denied.

Example Response

```
{
    "id": 1,
    "jobId": 101,
    "userId": 5,
    "status": "PENDING",
    "createdAt": "2024-12-01T10:00:00Z",
    "relatedJob": {
        "title": "Software Engineer"
    }
}
```

3. Create Application

POST `/`

Create a new job application for the authenticated user.

Request Body

Field	Type	Description
`jobId`	Integer	ID of the job to apply for (required).
`resumeUrl`	String	URL of the resume (optional).

Response

- **201 Created**: Application created successfully.
- **400 Bad Request**: Missing required fields.
- **409 Conflict**: Application already exists for the specified job.

Example Response

```
{
    "message": "Application created successfully.",
    "application": {
        "id": 2,
        "jobId": 101,
        "userId": 5,
        "status": "PENDING",
        "createdAt": "2024-12-01T10:05:00Z"
    }
}
```

4. Delete Application

DELETE `/:id`

Delete an existing application by ID. Only applications with a `PENDING` status can be deleted.

Path Parameters

Parameter	Type	Description

`id`	Integer	ID of the application to delete.

Response

- **200 OK**: Application deleted successfully.
- **404 Not Found**: Application not found or access denied.
- **400 Bad Request**: Only `PENDING` applications can be deleted.

Example Response

```
{
    "message": "Application deleted successfully."
}
```

Bookmark API Specification

The Bookmark API allows users to manage their bookmarks for jobs. Users can create, retrieve, and delete bookmarks. Below are the details for each endpoint.

Base URL

```
/bookmarks
```

Endpoints

1. Get Bookmarks

GET `/`

Retrieve the list of bookmarks for the authenticated user with pagination.

Response

- **200 OK**: Bookmarks retrieved successfully.

Example Response

```
{
    "total": 25,
```

```
    "page": 1,
    "totalPages": 3,
    "bookmarks": [
        {
            "id": 1,
            "jobId": 101,
            "userId": 1,
            "createdAt": "2024-12-01T10:00:00Z",
            "relatedJob": {
                "id": 101,
                "title": "Software Engineer",
                "company": "TechCorp"
            }
        },
        {
            "id": 2,
            "jobId": 102,
            "userId": 1,
            "createdAt": "2024-12-02T10:00:00Z",
            "relatedJob": {
                "id": 102,
                "title": "Data Scientist",
                "company": "DataWorks"
            }
        }
    ]
}
```


2. Toggle Bookmark

POST `/`

Add or remove a bookmark for a specific job.

Request Body

Field	Type	Description
`jobId`	Integer	ID of the job (required).

Response

- **201 Created**: Bookmark added successfully.
- **200 OK**: Bookmark removed successfully.
- **404 Not Found**: Job not found.

Example Responses

Bookmark Added

```
{
    "message": "Bookmark added",
    "bookmark": {
        "id": 3,
        "jobId": 103,
        "userId": 1,
        "createdAt": "2024-12-03T10:00:00Z"
    }
}
```

Bookmark Removed

```
{
    "message": "Bookmark removed"
}
```

---

3. Get Bookmark by ID

GET `/:id`

Retrieve details of a specific bookmark by its ID.

Path Parameters

Parameter	Type	Description
`id`	Integer	ID of the bookmark (required).

Response

- **200 OK**: Bookmark details retrieved successfully.

- **404 Not Found**: Bookmark not found.

Example Response

```
{
    "id": 1,
    "jobId": 101,
    "userId": 1,
    "createdAt": "2024-12-01T10:00:00Z",
    "relatedJob": {
        "id": 101,
        "title": "Software Engineer",
        "company": "TechCorp"
    }
}
```

---

4. Delete Bookmark

DELETE `/:id`

Delete a specific bookmark by its ID.

Path Parameters

Parameter	Type	Description
`id`	Integer	ID of the bookmark (required).

Response

- **200 OK**: Bookmark deleted successfully.
- **404 Not Found**: Bookmark not found.

Example Response

```
{
    "message": "Bookmark deleted successfully"
}
```

Error Handling

All endpoints return standard error responses with appropriate HTTP status codes and error messages.

Example Error Response

```
{
    "error": "Bookmark not found"
}
```

Company API Specification

The Company API allows users to manage company information such as creating, retrieving, and deleting company records. Below are the details for each endpoint.

Base URL

```
/companies
```

Endpoints

1. Get All Companies

GET `/`

Retrieve the list of all companies.

Response

- **200 OK**: Companies retrieved successfully.

Example Response

```
[
    {
        "id": 1,
        "name": "TechCorp",
        "location": "New York, USA",
        "website": "<https://www.techcorp.com>",
```

```
        "ceo": "Jane Doe",
        "industry": "Technology",
        "createdAt": "2024-12-01T10:00:00Z"
    },
    {
        "id": 2,
        "name": "DataWorks",
        "location": "San Francisco, USA",
        "website": "<https://www.dataworks.com>",
        "ceo": "John Smith",
        "industry": "Data Analytics",
        "createdAt": "2024-12-02T10:00:00Z"
    }
]
```

2. Create Company

POST `/`

Add a new company to the database.

Request Body

Field	Type	Description
`name`	String	Name of the company (required).
`location`	String	Location of the company (required).
`website`	String	Website URL (optional).
`ceo`	String	Name of the CEO (optional).
`industry`	String	Industry type (optional).

Response

- **201 Created**: Company added successfully.
- **400 Bad Request**: Validation failed.

Example Response

```
{
    "id": 3,
```

```
    "name": "CloudNine",
    "location": "Seattle, USA",
    "website": "<https://www.cloudnine.com>",
    "ceo": "Alice Johnson",
    "industry": "Cloud Services",
    "createdAt": "2024-12-03T10:00:00Z"
}
```

3. Get Company by ID

GET `/:id`

Retrieve details of a specific company by its ID.

Path Parameters

Parameter	Type	Description
`id`	Integer	ID of the company (required).

Response

- **200 OK**: Company details retrieved successfully.
- **404 Not Found**: Company not found.

Example Response

```
{
    "id": 1,
    "name": "TechCorp",
    "location": "New York, USA",
    "website": "<https://www.techcorp.com>",
    "ceo": "Jane Doe",
    "industry": "Technology",
    "createdAt": "2024-12-01T10:00:00Z"
}
```

4. Delete Company

DELETE `/:id`

Delete a specific company by its ID.

Path Parameters

Parameter	Type	Description
`id`	Integer	ID of the company (required).

Response

- **200 OK**: Company deleted successfully.
- **404 Not Found**: Company not found.

Example Response

```
{
    "message": "Company deleted successfully"
}
```

Error Handling

All endpoints return standard error responses with appropriate HTTP status codes and error messages.

Example Error Response

```
{
    "error": "Company not found"
}
```

Interview API Specification

The Interview API allows users to manage interview schedules, including creating, retrieving, and deleting interview records. Below are the details for each endpoint.

Base URL

```
/interviews
```

Endpoints

1. Get All Interviews

GET `/`

Retrieve the list of all interviews for the authenticated user.

Query Parameters

Parameter	Type	Description
`page`	Integer	Page number (optional, default: 1).
`limit`	Integer	Number of items per page (optional, default: 10).

Response

- **200 OK**: Interviews retrieved successfully.

Example Response


```
[
    {
        "id": 1,
        "applicationId": 10,
        "userId": 1,
        "date": "2024-12-15T14:00:00Z",
        "location": "Online - Zoom",
        "status": "Scheduled",
        "createdAt": "2024-12-01T10:00:00Z",
        "application": {
            "id": 10,
            "jobTitle": "Software Engineer",
            "company": "TechCorp"
        }
    },
    {
        "id": 2,
```

```
        "applicationId": 11,
        "userId": 1,
        "date": "2024-12-16T10:00:00Z",
        "location": "Company HQ",
        "status": "Scheduled",
        "createdAt": "2024-12-02T10:00:00Z",
        "application": {
            "id": 11,
            "jobTitle": "Data Scientist",
            "company": "DataWorks"
        }
    }
]
```


2. Create Interview

POST `/`

Create a new interview schedule for the authenticated user.

Request Body

Field	Type	Description
`applicationId`	Integer	ID of the related job application (required).
`date`	String	Scheduled date and time in ISO format (required).
`location`	String	Interview location (required).
`status`	String	Status of the interview (default: "Scheduled").

Response

- **201 Created**: Interview created successfully.
- **400 Bad Request**: Validation failed.

Example Response


```
{
    "id": 3,
    "applicationId": 12,
    "userId": 1,
```

```
    "date": "2024-12-17T11:00:00Z",
    "location": "Online - Google Meet",
    "status": "Scheduled",
    "createdAt": "2024-12-03T10:00:00Z"
}
```

---

3. Delete Interview

DELETE `/:id`

Delete a specific interview by its ID.

Path Parameters

Parameter	Type	Description
`id`	Integer	ID of the interview (required).

Response

- **200 OK**: Interview deleted successfully.
- **404 Not Found**: Interview not found.

Example Response

```
{
    "message": "Interview deleted successfully"
}
```

Error Handling

All endpoints return standard error responses with appropriate HTTP status codes and error messages.

Example Error Response

```
{
    "error": "Interview not found"
}
```

Job API Specification

The Job API provides endpoints to manage job postings, including listing, searching, filtering, sorting, and retrieving detailed information. Below are the details for each endpoint.

Base URL

```
/jobs
```

Endpoints

1. Get All Jobs

GET `/`

Retrieve a paginated list of all job postings with limited fields.

Query Parameters

Parameter	Type	Description
`page`	Integer	Page number (optional, default: 1).
`limit`	Integer	Number of items per page (optional, default: 20).

Response

- **200 OK**: Jobs retrieved successfully.

Example Response

```
{
    "page": 1,
    "totalPages": 10,
    "totalJobs": 200,
    "jobs": [
        {
            "id": 1,
            "title": "Software Engineer",
            "views": 150,
            "postedDate": "2024-12-01T10:00:00Z"
```

```
        },
        {
            "id": 2,
            "title": "Data Scientist",
            "views": 200,
            "postedDate": "2024-12-02T10:00:00Z"
        }
    ]
}
```

2. Get Job by ID

GET `/:id`

Retrieve detailed information about a specific job posting.

Path Parameters

Parameter	Type	Description
`id`	Integer	ID of the job posting (required).

Response

- **200 OK**: Job retrieved successfully.
- **404 Not Found**: Job not found.

Example Response

```
{
    "job": {
        "id": 1,
        "title": "Software Engineer",
        "company": "TechCorp",
        "location": "San Francisco, CA",
        "link": "<https://example.com/job/1>",
        "experience": "2+ years",
        "education": "Bachelor's degree",
        "employmentType": "Full-time",
        "deadline": "2024-12-31",
```

```
        "sector": "Technology",
        "postedDate": "2024-12-01T10:00:00Z",
        "views": 150
    },
    "relatedJobs": [
        {
            "id": 2,
            "title": "Data Scientist",
            "location": "San Francisco, CA",
            "company": "DataWorks"
        },
        {
            "id": 3,
            "title": "Backend Developer",
            "location": "San Francisco, CA",
            "company": "WebSolutions"
        }
    ]
}
```

---

3. Search Jobs

GET `/search`

Search for job postings by title.

Query Parameters

Parameter	Type	Description
`query`	String	Search keyword for job titles (required).

Response

- **200 OK**: Jobs retrieved successfully.
- **400 Bad Request**: Search query is required.

Example Response

```
[
    {
        "id": 91,
        "title": "[카메라 시스템] 카메라 광학 시스템 제작 및 응용 S
W개발",
        "company": "하이퍼그램 주식회사",
        "location": "대전  유성구",
        "link": "https://www.saramin.co.kr/zf_user/jobs/rel
ay/view?view_type=search&rec_idx=49573802&location=ts&searc
hword=%EA%B0%9C%EB%B0%9C%EC%9E%90&searchType=search&paid_fl
=n&search_uuid=73fa7d1a-5cff-4d44-ad3d-690333d74cae",
        "experience": "신입",
        "education": "대졸↑",
        "employmentType": "정규직",
        "deadline": "~ 12/31(화)",
        "sector": "영상처리, 이미지프로세싱, 컴퓨터비전, AI(인공지
능), C++ 외                    수정일 24/12/11",
        "postedDate": "2024-12-11T00:00:00.000Z",
        "views": 0,
        "createdAt": "2024-12-12T01:03:47.000Z",
        "updatedAt": "2024-12-12T01:03:47.000Z"
    }
]
```


4. Filter Jobs

GET `/filter`

Filter job postings based on various criteria.

Query Parameters

Parameter	Type	Description
`location`	String	Filter by location (optional).
`sector`	String	Filter by job sector (optional).
`employmentType`	String	Filter by employment type (optional).

Example: /jobs/filter?location=대전&employmentType=정규직&sector=영상처리

Response

- **200 OK**: Jobs retrieved successfully.

Example Response

```
[
    {
        "id": 1,
        "title": "Software Engineer",
        "company": "TechCorp",
        "location": "San Francisco, CA",
        "postedDate": "2024-12-01T10:00:00Z"
    },
    {
        "id": 3,
        "title": "Backend Developer",
        "company": "WebSolutions",
        "location": "San Francisco, CA",
        "postedDate": "2024-12-03T10:00:00Z"
    }
]
```

---

5. Sort Jobs

GET `/sort`

Sort job postings based on specified fields.

Query Parameters

Parameter	Type	Description
`sortBy`	String	Field to sort by (e.g., `title`, `company`, `postedDate`, `views`) (required).
`order`	String	Sort order (`ASC` or `DESC`, default: `ASC`).

Response

- **200 OK**: Jobs sorted successfully.
- **400 Bad Request**: Invalid sort field.

Example Response


```
[
    {
        "id": 2,
        "title": "Data Scientist",
        "company": "DataWorks",
        "location": "San Francisco, CA",
        "postedDate": "2024-12-02T10:00:00Z",
        "views": 200
    },
    {
        "id": 1,
        "title": "Software Engineer",
        "company": "TechCorp",
        "location": "San Francisco, CA",
        "postedDate": "2024-12-01T10:00:00Z",
        "views": 150
    }
]
```

Error Handling

All endpoints return standard error responses with appropriate HTTP status codes and error messages.

Example Error Response


```
{
    "error": "Job not found"
}
```


Notification API Specification

The **Notification API** allows users to retrieve and create notifications. Below are the detailed specifications for each endpoint.

Base URL

```
/notifications
```

Endpoints

1. Get All Notifications

GET `/`

Retrieve all notifications for the authenticated user.

Headers

Key	Value	Description
Authorization	Bearer `<token>`	JWT token for authenticated user

Response

- **200 OK**: Notifications retrieved successfully.

Example Response

```
[
    {
        "id": 1,
        "userId": 1,
        "title": "Interview Scheduled",
        "message": "Your interview for Software Engineer is
scheduled on 2024-12-15.",
        "type": "Interview",
        "isRead": false,
        "createdAt": "2024-12-01T10:00:00Z"
    },
    {
        "id": 2,
        "userId": 1,
```

```
        "title": "Job Alert",
        "message": "A new job posting for Data Scientist is
available.",
        "type": "JobAlert",
        "isRead": true,
        "createdAt": "2024-12-02T10:00:00Z"
    }
]
```

Error Responses

- **401 Unauthorized**: User is not authenticated.

```
{
    "error": "Unauthorized"
}
```

---

2. Create Notification

POST `/`

Create a new notification for the authenticated user.

Headers

Key	Value	Description
Authorization	Bearer `<token>`	JWT token for authenticated user

Request Body

Field	Type	Description
`title`	String	Title of the notification (required).
`message`	String	Notification message content (required).
`type`	String	Type of notification. One of: `System`, `JobAlert`, `Interview`. Default: `System`.

Example Request

```
{
    "title": "New Interview Scheduled",
    "message": "Your interview for Backend Developer is sch
eduled on 2024-12-20.",
    "type": "Interview"
}
```

Response

- **201 Created**: Notification created successfully.

Example Response

```
{
    "id": 3,
    "userId": 1,
    "title": "New Interview Scheduled",
    "message": "Your interview for Backend Developer is sch
eduled on 2024-12-20.",
    "type": "Interview",
    "isRead": false,
    "createdAt": "2024-12-03T10:00:00Z"
}
```

Error Responses

- **400 Bad Request**: Validation error or missing required fields.

```
{
    "error": "Validation failed: 'title' and 'message' are
required."
```

```
}
```

- **401 Unauthorized**: User is not authenticated.

```
{
    "error": "Unauthorized"
}
```

Error Handling

All endpoints return standard error responses with appropriate HTTP status codes and error messages.

Example Error Response

```
{
    "error": "Something went wrong"
}
```

Task API

1. Retrieve All Tasks

Endpoint: `GET /`

Middleware: `authMiddleware`

Description: Fetches all tasks belonging to the currently authenticated user.

Request:

- **Headers**
  - `Authorization`: Bearer Token (JWT) for user authentication.

Response:

- **200 OK**Returns a list of tasks in JSON format.**Example Response:**

```
[
  {
    "id": 1,
    "userId": 5,
    "title": "Complete project documentation",
    "isCompleted": false,
    "dueDate": "2024-06-30T00:00:00.000Z",
    "createdAt": "2024-06-14T10:30:00.000Z",
    "updatedAt": "2024-06-15T14:00:00.000Z"
  }
]
```

- **500 Internal Server Error**Returns an error message.**Example:**

```
{ "error": "Server error occurred." }
```

---

2. Retrieve a Specific Task

Endpoint: `GET /:id`

Middleware: `authMiddleware`

Description: Retrieves a specific task based on its ID. Only tasks belonging to the authenticated user can be accessed.

Request:

- **Headers**
  - `Authorization`: Bearer Token (JWT).
- **Parameters**
  - `id`: Task ID (integer).

Response:

- **200 OK**Returns the requested task.**Example Response:**

```
{
  "id": 1,
  "userId": 5,
  "title": "Complete project documentation",
  "isCompleted": false,
  "dueDate": "2024-06-30T00:00:00.000Z",
  "createdAt": "2024-06-14T10:30:00.000Z",
  "updatedAt": "2024-06-15T14:00:00.000Z"
}
```

- **404 Not Found**If the task does not exist or belongs to another user.**Example:**

```
{ "error": "Task not found or access denied" }
```

- **500 Internal Server Error**Returns an error message.

---

3. Create a New Task

Endpoint: `POST /`

Middleware: `authMiddleware`

Description: Creates a new task for the currently authenticated user.

Request:

- **Headers**
  - `Authorization`: Bearer Token (JWT).
- **Body (JSON)**
  - `title` (string, required): Title of the task.
  - `dueDate` (string, optional): Task due date (ISO format).

  Example Request Body:

```
{
  "title": "Write API documentation",
  "dueDate": "2024-06-25"
}
```

Response:

- **201 Created**Returns the newly created task.**Example Response:**

```
{
  "id": 2,
  "userId": 5,
  "title": "Write API documentation",
  "isCompleted": false,
  "dueDate": "2024-06-25T00:00:00.000Z",
  "createdAt": "2024-06-14T11:00:00.000Z",
  "updatedAt": "2024-06-14T11:00:00.000Z"
}
```

- **500 Internal Server Error**Returns an error message.

---

4. Update a Task

Endpoint: `PUT /:id`

Middleware: `authMiddleware`

Description: Updates a specific task belonging to the authenticated user.

Request:

- **Headers**
  - `Authorization`: Bearer Token (JWT).
- **Parameters**
  - `id`: Task ID (integer).

- **Body (JSON)**
  - `title` (string, optional): Updated task title.
  - `isCompleted` (boolean, optional): Completion status of the task.
  - `dueDate` (string, optional): Updated due date (ISO format).

  Example Request Body:

```
{
  "title": "Update API documentation",
  "isCompleted": true,
  "dueDate": "2024-06-26"
}
```

Response:

- **200 OK**Returns the updated task.**Example Response:**

```
{
  "id": 2,
  "userId": 5,
  "title": "Update API documentation",
  "isCompleted": true,
  "dueDate": "2024-06-26T00:00:00.000Z",
  "createdAt": "2024-06-14T11:00:00.000Z",
  "updatedAt": "2024-06-15T12:00:00.000Z"
}
```

- **404 Not Found**If the task does not exist or belongs to another user.**Example:**

```
{ "error": "Task not found or access denied" }
```

- **500 Internal Server Error**Returns an error message.

---

5. Delete a Task

Endpoint: `DELETE /:id`

Middleware: `authMiddleware`

Description: Deletes a specific task belonging to the authenticated user.

Request:

- **Headers**
  - `Authorization`: Bearer Token (JWT).
- **Parameters**
  - `id`: Task ID (integer).

Response:

- **204 No Content**Indicates successful deletion without returning a response body.
- **404 Not Found**If the task does not exist or belongs to another user.**Example:**

```
{ "error": "Task not found or access denied" }
```

- **500 Internal Server Error**Returns an error message.

Task Model

The `Task` model represents tasks in the database.

Field	Type	Constraints
`id`	INTEGER	Primary Key, Auto Increment
`userId`	INTEGER	Foreign Key (references `User`)
`title`	STRING	Not Null
`isCompleted`	BOOLEAN	Default: `false`

`dueDate`	DATE	Optional
`createdAt`	TIMESTAMP	Auto-generated
`updatedAt`	TIMESTAMP	Auto-updated

Auth API

Base URL

`/auth`

---

1. User Login

Endpoint

`POST /login`

Description

Logs in a user and generates a JWT access token.

Request Body

```
{
  "email": "string",    // User's email
  "password": "string"  // User's password
}
```

Response

- **Success (200)**

```
{
  "message": "Login successful",
  "token": "string",  // JWT Access Token
  "user": {
    "id": "integer",
    "email": "string"
  }
```

```
}
```

- **Error (401)**

```
{
  "message": "Invalid email or password"
}
```

---

2. User Registration

Endpoint

`POST /register`

Description

Registers a new user with an email and password.

Request Body

```
{
  "email": "string",    // User's email
  "password": "string"  // User's password
}
```

Response

- **Success (201)**

```
{
  "message": "User registered successfully",
  "user": {
    "id": "integer",
    "email": "string"
  }
}
```

- **Error (400)**

```
{
  "message": "Invalid email format"
}
```

- **Error (409)**

```
{
  "message": "Email already in use"
}
```

---

3. Refresh Access Token

Endpoint

`POST /refresh`

Description

Generates a new access token using a valid refresh token.

Request Body

```
{
  "refreshToken": "string"  // Valid Refresh Token
}
```

Response

- **Success (200)**

```
{
  "accessToken": "string"  // New JWT Access Token
}
```

- **Error (400)**

```
{
  "message": "Refresh token is required"
}
```

- **Error (404)**

```
{
  "message": "User not found"
}
```

---

4. Update User Profile

Endpoint

`PUT /update`

Description

Updates the user's email and/or password.

Headers

`Authorization: Bearer <JWT Access Token>`

Request Body

```
{
  "email": "string",      // New email (optional)
  "password": "string"    // New password (optional)
}
```

Response

- **Success (200)**

```
{
  "message": "Profile updated successfully",
  "user": {
    "id": "integer",
```

```
    "email": "string"
  }
}
```

- **Error (404)**

```
{
  "message": "User not found"
}
```

---

5. Get User Profile

Endpoint

`GET /profile`

Description

Retrieves the authenticated user's profile.

Headers

`Authorization: Bearer <JWT Access Token>`

Response

- **Success (200)**

```
{
  "message": "User profile retrieved successfully",
  "user": {
    "id": "integer",
    "email": "string",
    "createdAt": "string" // User creation date
  }
}
```

- **Error (404)**

```
{
  "message": "User not found"
}
```

---

6. Delete User Account

Endpoint

`DELETE /delete`

Description

Deletes the authenticated user's account.

Headers

`Authorization: Bearer <JWT Access Token>`

Response

- **Success (200)**

```
{
  "message": "Account deleted successfully"
}
```

- **Error (404)**

```
{
  "message": "User not found"
}
```

Common Error Responses

- **Unauthorized (401)**

```
{
  "message": "Invalid email or password"
```

```
}
```

- **Validation Error (400)**

```
{
  "message": "Bad Request"
}
```

- **Server Error (500)**

```
{
  "message": "Internal Server Error"
}
```