EasyWay API

The EasyWay's API is a JSON-based API that allows developers to access and interact with the web application's data and functionalities. All requests are made to endpoints beginning with "http://localhost:3000/api" and "http://localhost:5000", and can be made using HTTP methods.

By using the EasyWay's API, developers can build custom integrations or third-party applications that can communicate with the web application's backend. This API documentation provides detailed information about the available endpoints, their input and output parameters, and the necessary authentication requirements to access them.

Whether you are building a mobile application, integrating with a third-party service, or simply exploring the possibilities of the EasyWay's API, this documentation will guide you through the process and help you get started quickly and efficiently.

User

The User contains the API documentation for a web application. It includes descriptions of the available endpoints, their inputs and outputs, and any relevant information about authentication, errors, and usage guidelines.

Within this, there are several endpoint descriptions for user-related actions.

The "**POST** Add User" endpoint allows a user to be created with the specified details.

The "**GET** Get User Details" endpoint retrieves the details of a specific user, based on their unique identifier.

The "**POST** Forgot Password" endpoint initiates the process for a user to reset their forgotten password.

The "**POST** Forgot Username" endpoint initiates the process for a user to retrieve their forgotten username.

Each endpoint should have its own separate documentation file, outlining the specific parameters, responses, and any additional information required for usage.

POST Add user

```
http://localhost:3000/api/user
```

The "Add user" API POST endpoint allows clients to create a new user in the system. The endpoint URL is "http://localhost:3000/api/user". The request must contain a JSON object in the request body with the following attributes:

- "name": The name of the user (string)
- "email": The email of the user (string)

- "gender": The gender of the user, either "M" (male) or "F" (female) (string)
- "username": The username of the user (string)
- "password": The password of the user (string)

If the request is successful, the server will return a 201 CREATED status code along with a JSON object containing the newly created user's attributes, including an "id" attribute generated by the server.

Body raw (json)

```
json
{
    "name": "alex",
    "email": "alex@pace.edu",
    "gender": "F",
    "username": "alex",
    "password": "alex1"
}
```

GET Get user details

http://localhost:3000/api/user/dummy

The "Get user details" API GET endpoint allows clients to retrieve the details of a user from the system. The endpoint URL includes the user's unique identifier, for example "http://localhost:3000/api/user/1" to retrieve the details of user with id 1.

In the example provided, the endpoint URL is "http://localhost:3000/api/user/dummy" which suggests that there might be a user in the system with the username "dummy".

If the request is successful, the server will return a 200 OK status code along with a JSON object containing the requested user's attributes.

POST Forgot Password

http://localhost:3000/api/forgotPassword

This is a RESTful API for resetting a forgotten password in a web application. The API endpoint is located at http://localhost:3000/api/forgotPassword and accepts a POST request. The request body is expected to contain a JSON object with the email address of the user who forgot their password. The API response is a JSON object containing a message indicating that a temporary password has been sent to the email address provided.

To use this API, clients can make a POST request to the endpoint with the email address of the user who forgot their password included in the request body.

The API will then generate a temporary password and send it to the email address provided. The client will receive a JSON response indicating that the temporary password has been sent to the email address.

This API can be used by web developers who want to provide their users with a way to reset their passwords if they forget them. It simplifies the process of resetting a password for the user and can help reduce the number of support requests related to password resets.

If the request is successful, the server will return a 200 OK status code along with a JSON object containing the requested user's attributes.

Body raw

```
{
  "email": "dummy@pace.edu"
}
```

POST Forgot Username

http://localhost:3000/api/forgotUsername

This is a RESTful API for resetting a forgotten username in a web application. The API endpoint is located at http://localhost:3000/api/forgotUsername and accepts a POST request. The request body is expected to contain a JSON object with the email address of the user who forgot their username. The API response is a JSON object containing a message indicating that a temporary username has been sent to the email address provided.

To use this API, clients can make a POST request to the endpoint with the email address of the user who forgot their username included in the request body. The API will then generate a temporary username and send it to the email address provided. The client will receive a JSON response indicating that the temporary username has been sent to the email address.

This API can be used by web developers who want to provide their users with a way to reset their usernames if they forget them. It simplifies the process of resetting a username for the user and can help reduce the number of support requests related to username resets.

If the request is successful, the server will return a 200 OK status code along with a JSON object containing the requested user's attributes.

Body raw

```
{
  "email": "dummy@pace.edu"
}
```

Service

The Service contains the API documentation for a web application. It includes descriptions of the available endpoints, their inputs and outputs, and any relevant information about authentication, errors, and usage guidelines.

Within this, there are several endpoint descriptions for service-related actions.

The "**GET** Get all services" endpoint retrieves a list of all available services within the system.

The "**GET** Search Services" endpoint allows users to search for services based on certain criteria, such as keywords or specific parameters.

Each endpoint should have its own separate documentation file, outlining the specific parameters, responses, and any additional information required for usage. The documentation should also include any relevant examples or code snippets to assist developers in using the API.

GET Get all services

```
http://localhost:3000/api/services
```

The "Get all services" API GET endpoint allows clients to retrieve a list of all services from the system. The endpoint URL is "http://localhost:3000/api/services".

If the request is successful, the server will return a 200 OK status code along with a JSON object containing an array of service objects. Each service object in the array will contain the attributes of the service, such as service id, name, description, and cost.

GET Search Services

```
http://localhost:3000/api/services/search?name=plum
```

This API allows users to search for services based on a query parameter 'name'. The endpoint URL is http://localhost:3000/api/services/search?name=plum, and the HTTP method is GET. The query parameter 'name' is required and should be specified in the URL as `name=` . A request body is optional, and can be sent as JSON in the raw text format.

An example request is provided in cURL format, with the query parameter 'name' set to 'plum' and a request body specifying a search query for 'Plumber'. The response is returned as a JSON array of services matching the search query, each service represented as an object with the following properties: 'id', 'name', 'description', 'category', 'image_name', and 'price'.

PARAMS

name	plum

Body raw (json)

```
json
{
    "body" : "Plumber"
}
```

Booking

The Booking contains the API documentation for a web application. It includes descriptions of the available endpoints, their inputs and outputs, and any relevant information about authentication, errors, and usage guidelines.

Within this, there are several endpoint descriptions for booking-related actions.

The "**GET** Get Bookings from a given user" endpoint retrieves a list of all bookings made by a specific user, based on their unique identifier.

The "**POST** Book a Service" endpoint allows users to create a new booking for a particular service, providing the necessary details and parameters.

The "**PUT** Update Booking" endpoint allows users to update an existing booking, based on its unique identifier. This may include changes to the date, time, or service details.

Each endpoint should have its own separate documentation file, outlining the specific parameters, responses, and any additional information required for usage. The documentation should also include any relevant examples or code snippets to assist developers in using the API.

GET Get bookings from a given user

```
http://localhost:3000/api/bookings/3
```

The "Get bookings from a given user" API GET endpoint allows clients to retrieve a list of all bookings associated with a particular user. The endpoint URL includes the user's unique identifier, for example "http://localhost:3000/api/bookings/3 retrieve the bookings of user with id 3.

If the request is successful, the server will return a 200 OK status code along with a JSON object containing an array of booking objects. Each booking object in the array will contain the attributes of the booking, such as booking id, user id, service id, date, start time, and end time.

POST Book a service

```
http://localhost:3000/api/bookService
```

The "Book a service" API POST endpoint allows clients to book a service by creating a new booking. Clients can send a cURL POST request to the endpoint "http://localhost:3000/api/bookService" to create a new booking.

"http://localhost:3000/api/bookService" to create a new booking.

The request body must include a JSON object that contains the following attributes:

- id: the unique identifier for the booking user_id: the unique identifier for the
- user who is making the booking service_id: the unique identifier for the service
- being booked date: the date of the booking, in YYYY-MM-DD format start_time:
- the start time of the booking, in HH:MM format end_time: the end time of the
- booking, in HH:MM format
-

If the request is successful, the server will return a 201 CREATED status code along with a JSON object containing the attributes of the new booking.

If there is an error in creating the new booking, the server will return an appropriate error response.

Body raw (json)

```json
{
    "id": 5,
    "user_id": 3,
    "service_id": 4,
    "date": "2022-02-15",
    "start_time": "12:30",
    "end_time": "13:30"
}
```

PUT Update Booking

```
http://localhost:3000/updateBooking?id=1
```

This API is a PUT endpoint that updates an existing booking based on its ID. The endpoint is located at http://localhost:3000/updateBooking and expects a query parameter "id" that represents the ID of the booking to be updated.

The request body contains a JSON object that represents the updated booking details. The example shows an updated booking with an ID of 1, userId of 1, serviceId of 1, and a new date, start time, and end time for the booking. Additionally, there is a boolean flag "isCancelled" to indicate if the booking has been cancelled.

The API will update the booking details based on the ID provided in the query parameter and the new booking details provided in the request body. The response of the API will indicate the success or failure of the update operation.

This API is useful in various applications that require updating existing bookings, such as booking systems for hotels, restaurants, or other service-based businesses.

PARAMS
PARAMS

id	1

Body raw

```
{
    "id": 1,
    "userId": 1,
    "serviceId": 1,
    "date": "2022-05-20",
    "startTime": "14:00",
    "endTime": "15:00",
    "isCancelled": false
}
```


Blog

The Blog contains the API documentation for a web application. It includes descriptions of the available endpoints, their inputs and outputs, and any relevant information about authentication, errors, and usage guidelines.

Within this, there is a single endpoint description for retrieving all blogs.

The "**GET** Get All Blogs" endpoint retrieves a list of all blog posts available within the system. This endpoint may include parameters to filter the results based on certain criteria, such as author or category.

The endpoint should have its own separate documentation file, outlining the specific parameters, responses, and any additional information required for usage. The documentation should also include any relevant examples or code snippets to assist developers in using the API.

GET Get All Blogs

```
http://localhost:3000/api/getAllBlogs
```

This is a RESTful API that retrieves all the blogs in a web application. The API endpoint is located at http://localhost:3000/api/getAllBlogs and accepts a GET request. The request body is empty, and the API response is a JSON object containing an array of all the blogs in the web application.

To use this API, clients can make a GET request to the endpoint with no request body. The API will then retrieve all the blogs in the web application and return them in a JSON array. Each blog in the array contains an id, a title, content, created_at and updated_at timestamp, and an optional image_name attribute.

This API can be used by web developers who want to display all the blogs in their web application to users. It simplifies the process of retrieving all the blogs This API can be used by web developers who want to display all the blogs in their web application to users. It simplifies the process of retrieving all the blogs in the web application and makes it easier for clients to integrate the blogs into their website or application.

Body raw (json)

```
json

{

}
```

Comment

The Comment contains the API documentation for a web application. It includes descriptions of the available endpoints, their inputs and outputs, and any relevant information about authentication, errors, and usage guidelines.

Within this, there are several endpoint descriptions for blog and comment-related actions.

The "**GET** Get All Comments of a Blog" endpoint retrieves a list of all comments associated with a specific blog post, based on its unique identifier.

The "**POST** Add Comment" endpoint allows users to create a new comment for a particular blog post, providing the necessary details and parameters.

Each endpoint should have its own separate documentation file, outlining the specific parameters, responses, and any additional information required for usage. The documentation should also include any relevant examples or code snippets to assist developers in using the API.

GET Get All Comments of a Blog

```
http://localhost:3000/api/getAllComments/1
```

This is a RESTful API that retrieves comments associated with a particular blog post. The API endpoint is located at http://localhost:3000/api/getAllComments/1, it likely accepts a GET request with a blog_id parameter that identifies the blog post to retrieve comments for.

The API response is a JSON array containing one or more comment objects. Each comment object contains an id that uniquely identifies the comment, a blog_id that indicates the blog post the comment is associated with, a content field that contains the text of the comment, and created_at and updated_at fields that indicate when the comment was created and last updated, respectively.

This API can be used by web developers who want to display comments associated with a particular blog post to users. By providing an API endpoint to retrieve comments, developers can simplify the process of retrieving and displaying comments and can integrate the comments into their website or application.

POST Add Comment

```
http://localhost:3000/api/blogs/1/comments
```

This is a RESTful API that allows users to add a comment to a particular blog post. The API endpoint is provided as http://localhost:3000/api/blogs/1/comments, where 1 is the blog_id of the blog post that the comment should be associated with.

The API request requires a POST method, and the request body is expected to be a JSON object containing a blog_id field that indicates the ID of the blog post that the comment should be associated with, and a content field that contains the text of the comment.

The API response is not shown in the example, but it would typically include a status code indicating whether the comment was successfully added or not.

This API can be used by web developers who want to allow users to add comments to their blog posts. By providing an API endpoint to add comments, developers can simplify the process of adding comments and can integrate the comments into their website or application.

Body raw

```
{
    "blog_id": 0,
    "content": "This is a new comment"
}
```

Detection

The Detection contains the API documentation for a web application. It includes descriptions of the available endpoints, their inputs and outputs, and any relevant information about authentication, errors, and usage guidelines.

Within this, there are endpoint descriptions for language detection-related actions.

The "**POST** Detections" endpoint allows users to input text and detect the language it is written in. This endpoint will return the detected language code.

The "**GET** Get Detections" endpoint retrieves a list of all the language detection results made by the user for a specific session, identified by a unique identifier.

Each endpoint should have its own separate documentation file, outlining the specific parameters, responses, and any additional information required for usage. The documentation should also include any relevant examples or code snippets to assist developers in using the API.

POST Detections

```
http://localhost:3600/detections
```

This API is a POST endpoint that receives image files and returns a JSON response containing object detection results. The endpoint is located at http://localhost:3600/detections.

The request body format is form-data and contains a single field named "images" which represents the image file being uploaded. The example shows an image file named "dog.jpg" with a unique identifier of "1RILOqThE".

The expected response format is a JSON object with a "response" field that contains an array of objects. Each object in the array represents a single image and contains a "detections" field which is an array of detected objects with their respective confidence scores. Additionally, each object contains an "image" field which represents the name of the image file.

This API uses a deep learning-based object detection algorithm to detect objects in the uploaded image. It can detect multiple objects and their respective confidence scores. The API is useful in various applications that require object detection, such as security surveillance, image tagging, and autonomous vehicles.

Body formdata

images

GET Get Detections

```
http://localhost:3000/api/detection?folderName=detections
```

This API is a GET endpoint that retrieves a list of detections from a specified folder. The endpoint is located at http://localhost:3000/api/detection and expects a query parameter "folderName" that represents the name of the folder where the detections are stored.

The example shows a request being made to retrieve detections from a folder named "detections". The response from the API will be a list of detections in the specified folder.

This API is useful in various applications that require accessing object detection results, such as image analysis and object tracking systems. By retrieving the detection results, further analysis and processing can be performed on the detected objects.

PARAMS

folderName	detections