

DRF PoC Web API

- <https://django-poc-session-maheshbodas.herokuapp.com/auth/login/>
- Use above link to access browsable Web API.
- Username :- admin (lower case) – Admin
- Password :- poctest#1 (lower case)
- Username :- editor (lower case) – Non Admin
- Password :- test#123 (lower case)

Unit testing.

- Added test cases related for Post and Get action for RiskType and Risks.
- Tested response for Admin and Non-admin users as API support role base authorization.
- Made use of Design patterns to avoid duplication of code while writing test cases.

Salient features

- Solution that allows insurers to define their own custom data model for their risks. There are no database tables called automobiles, houses, or prizes. Instead, insurers will be able to create their own risk types and attach as many different fields as they would like.
- Fields are bits of data like first name, age, zip code, model, serial number, Coverage A limit, or prize dollar amount. Basically any data the carrier would want to collect about the risk. Fields can also be of different types, like text, date, number, currency, and so forth.

Salient features (continued)

- As risk field types are user defined and not committed at time of table creation. Actual field/column value stored in table is string type. But field type sanctity is enforced using model validators. i.e User can not supply Date value where Currency is expected.
- Uniqueness among RiskType names is maintained by enforcing unique constrain. But enforcing unique risk field name in similar fashion is somewhat restrictive and naive. I have tried to maintain unique field name within Risk Types and not across all Risk Type using mode validators.

Salient features (continued)

- During risk and risk field creation system checks that proper referential integrity is maintained among Risk Type and Risks. Risk Fields ensures that proper referential integrity is maintained with Risk Type and Risk Type fields.
- Risk API allows users to create RiskTypes and associated RiskTypeFields in one go with use of nested serializer.
- Risk API allows users to create Risk and associated RiskFields in one go with use of nested serializer.
- Various model validation errors within RiskType and RiskTypeFields, Risk and RiskFields are returned to client of RiskAPI in JSON format.

Salient features (continued)

- Risk and RiskFields carry extra metadata related to RiskType and RiskTypeFields.
- On authentication front, only authenticated users can access RiskAPI. Role based authorization further enable only admin users to create/delete RiskType. Non admin users only have list and details option for RiskTypes. However they can create Risk instances based on RiskTypes.
- Session authentication is used for browsable API that runs within same session that of Web API
- Token authentication is used for seperate website that access Web API for retrieving and creating RiskTypes and Risk.

Salient features (continued)

- Used Default router to list all available links and Pastebin docs and schema information also available for API.
- One touch Heroku deploy button.

Screenshot Heroku deployment

The screenshot shows the Heroku dashboard for an application named "django-poc-session-maheshbodas". The browser address bar shows the URL "https://dashboard.heroku.com/apps/django-poc-session-maheshbodas". The Heroku logo is in the top left, and a search bar is in the top right. The application name is displayed in the top navigation bar, along with a star icon, an "Open app" button, and a "More" dropdown menu. Below the navigation bar, there are tabs for "Overview", "Resources", "Deploy", "Metrics", "Activity", "Access", and "Settings". The "Overview" tab is selected. The main content area is divided into three sections: "Installed add-ons", "Dyno formation", and "Collaborator activity".

Installed add-ons \$0.00/month [Configure Add-ons](#)

- Heroku Postgres**
Hobby Dev

Dyno formation \$0.00/month [Configure Dynos](#)

This app is using free dynos

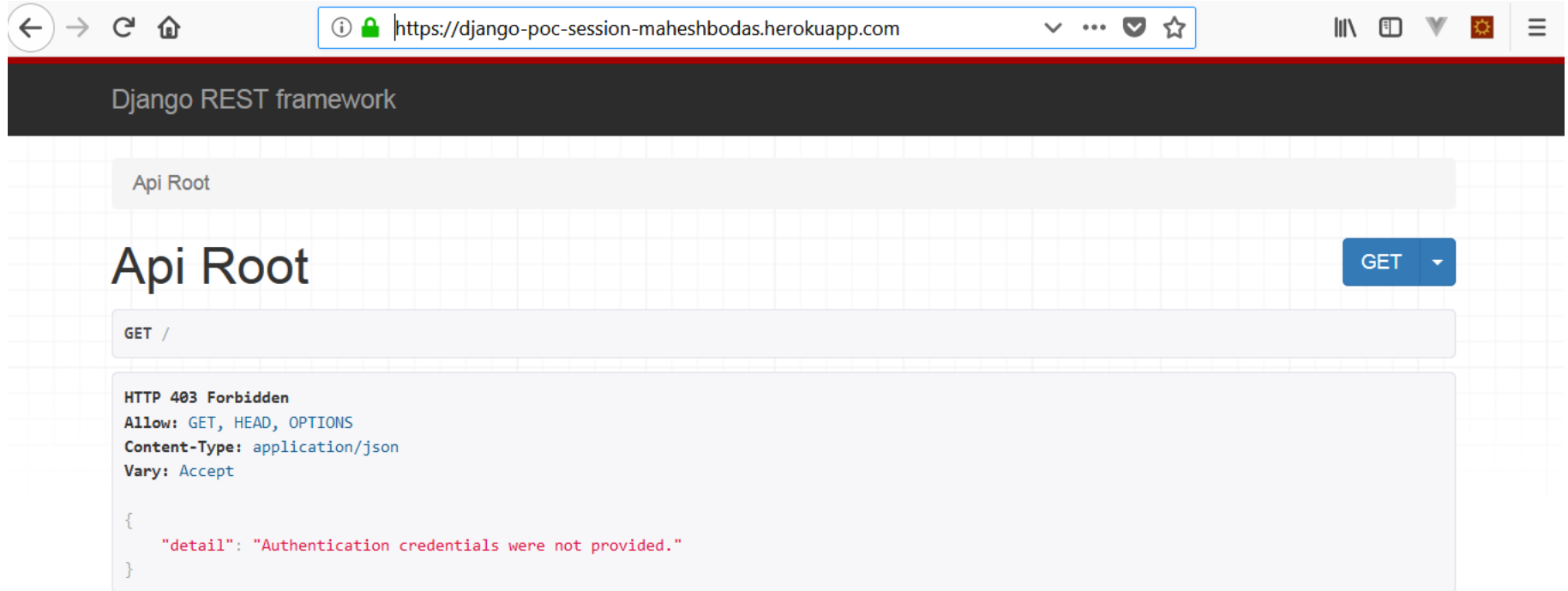
Process Type	Command	Status
web	gunicorn pocserver.wsgi --log-file -	ON

Collaborator activity [Manage Access](#)

Latest activity [All Activity](#)

- bodasmaresh@yahoo.com:** Deployed
Jul 16 at 6:36 PM · v5
- bodasmaresh@yahoo.com:** Build succeeded
Jul 16 at 6:35 PM · [View build log](#)
- bodasmaresh@yahoo.com:** Attach DATABASE (@ref:postgresql-flexible-43498)
Jul 16 at 6:35 PM · v4
- bodasmaresh@yahoo.com:** Set `ENVIRONMENT`, `SECRET_KEY` config vars

API response when not Logged In



The screenshot shows a web browser window with the address bar displaying `https://django-poc-session-maheshbodas.herokuapp.com`. The page title is "Django REST framework". The main content area shows "Api Root" with a "GET" button. Below this, the response details are displayed:

GET /

HTTP 403 Forbidden
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

```
{
  "detail": "Authentication credentials were not provided."
}
```

Enter credentials

← → ↻ 🏠 🔒 https://django-poc-session-maheshbodas.herokuapp.com/auth/login/ ⋮ 📄 ⚙️ ☰

Django REST framework

HTTP 405 Method Not Allowed
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

```
{  
  "detail": "Method \"GET\" not allowed."  
}
```

Raw data HTML form

Username

Email

Password

POST

Successful Login

The screenshot shows a web browser window with the address bar displaying `https://django-poc-session-maheshbodas.herokuapp.com/auth/login/`. The page header identifies the application as "Django REST framework" and the user as "mahesh.bodas". The breadcrumb trail shows "Api Root / Login". The main heading is "Login", with an "OPTIONS" button to its right. The description states: "Check the credentials and return the REST Token if the credentials are valid and authenticated. Calls Django Auth login method to register User ID in Django session framework". It also notes: "Accept the following POST parameters: username, password Return the REST Framework Token Object's key." The request details show a POST to `/auth/login/`. The response is an HTTP 200 OK with headers: `Allow: POST, OPTIONS`, `Content-Type: application/json`, and `Vary: Accept`. The JSON response body is `{ "key": "7fc2cd897a9628d89f77016e2c9ae2479114fbff" }`. At the bottom right, there are buttons for "Raw data" and "HTML form".

← → ↻ 🏠

🔒 `https://django-poc-session-maheshbodas.herokuapp.com/auth/login/` ⋮ 🛡️ ☆

📄 📄 ▼ ⚙️ ☰

Django REST framework mahesh.bodas

Api Root / Login

Login OPTIONS

Check the credentials and return the REST Token if the credentials are valid and authenticated. Calls Django Auth login method to register User ID in Django session framework

Accept the following POST parameters: username, password Return the REST Framework Token Object's key.

POST `/auth/login/`

HTTP 200 OK
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

```
{  
  "key": "7fc2cd897a9628d89f77016e2c9ae2479114fbff"  
}
```

Raw data HTML form

API Root shows Default Router

The screenshot shows a web browser window with the address bar displaying `https://django-poc-session-maheshbodas.herokuapp.com`. The page header is dark grey with "Django REST framework" on the left and "mahesh.bodas" on the right. The main content area has a light grey background with a grid pattern. At the top, there's a "Api Root" label. Below it, the title "Api Root" is displayed in a large font, followed by the subtitle "The default basic root view for DefaultRouter". To the right of the title are two buttons: "OPTIONS" and "GET". Below the subtitle, there's a section for the GET method, showing the response details: "HTTP 200 OK", "Allow: GET, HEAD, OPTIONS", "Content-Type: application/json", and "Vary: Accept". The response body is a JSON object listing the API endpoints.

Api Root

Api Root

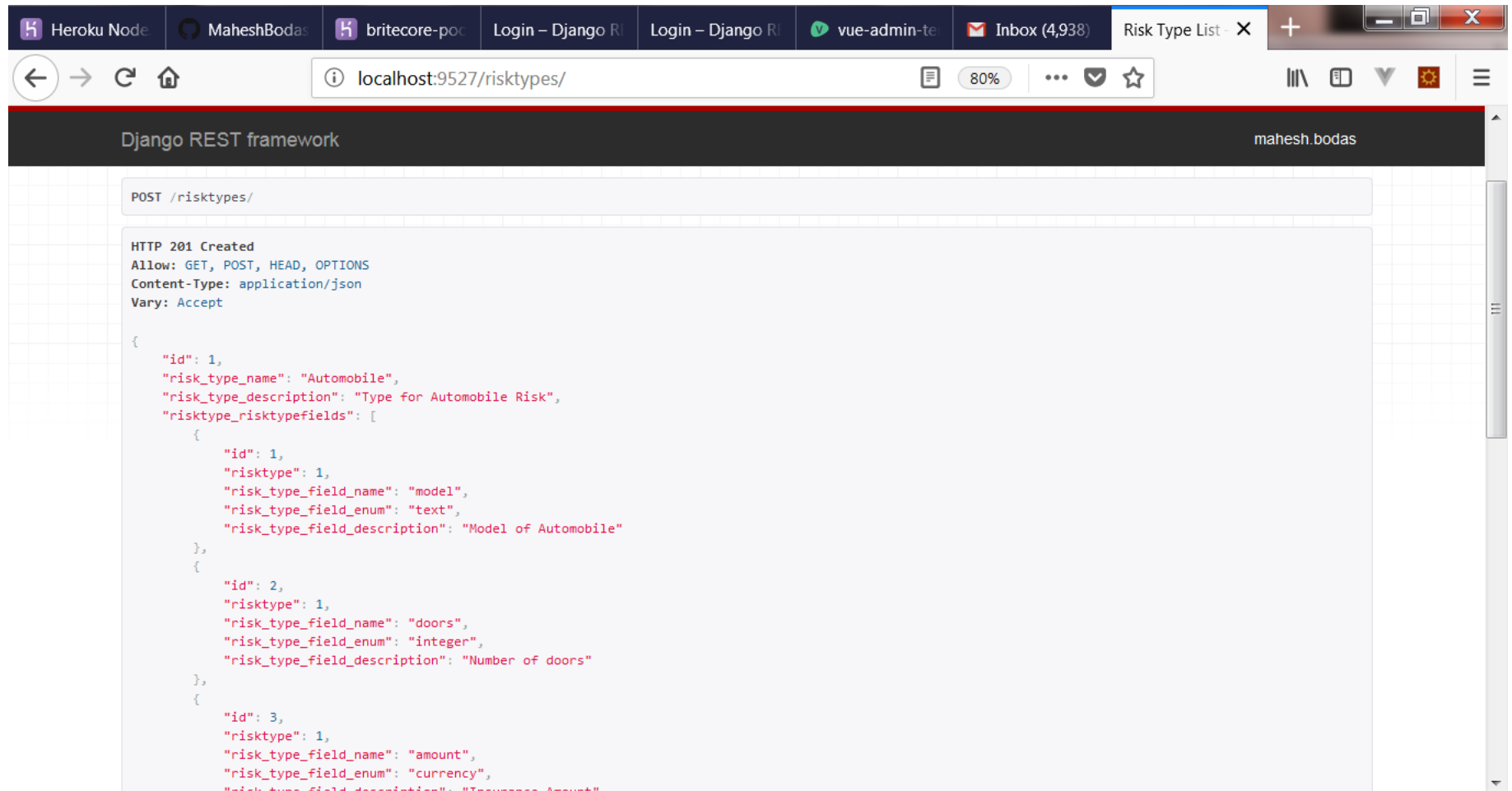
The default basic root view for DefaultRouter

GET /

HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

```
{
  "risktypekeys": "https://django-poc-session-maheshbodas.herokuapp.com/risktypekeys/",
  "riskkeys": "https://django-poc-session-maheshbodas.herokuapp.com/riskkeys/",
  "risktypes": "https://django-poc-session-maheshbodas.herokuapp.com/risktypes/",
  "risks": "https://django-poc-session-maheshbodas.herokuapp.com/risks/",
  "users": "https://django-poc-session-maheshbodas.herokuapp.com/users/"
}
```

Posting RiskType successfully



The screenshot shows a web browser window with the address bar displaying `localhost:9527/risktypes/`. The browser tabs include Heroku Node, MaheshBodas, britecore-poc, Login - Django R, Login - Django R, vue-admin-te, Inbox (4,938), and Risk Type List - X. The page title is "Django REST framework" and the user is logged in as "mahesh.bodas".

The main content area displays the response to a POST request to `/risktypes/`. The response is a 201 Created status with the following headers:

```
HTTP 201 Created
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept
```

The response body is a JSON object containing three risk types:

```
{
  "id": 1,
  "risk_type_name": "Automobile",
  "risk_type_description": "Type for Automobile Risk",
  "risktype_risktypefields": [
    {
      "id": 1,
      "risktype": 1,
      "risk_type_field_name": "model",
      "risk_type_field_enum": "text",
      "risk_type_field_description": "Model of Automobile"
    },
    {
      "id": 2,
      "risktype": 1,
      "risk_type_field_name": "doors",
      "risk_type_field_enum": "integer",
      "risk_type_field_description": "Number of doors"
    },
    {
      "id": 3,
      "risktype": 1,
      "risk_type_field_name": "amount",
      "risk_type_field_enum": "currency",
      "risk_type_field_description": "Insurance Amount"
    }
  ]
}
```

Posting Risk Instance

The screenshot shows a web browser window with the address bar displaying `localhost:9527/risks/`. The browser's taskbar at the top includes icons for Heroku Node, MaheshBodas, britecore-poc, two instances of Login - Django R, vue-admin-te, and an Inbox with 4,938 messages. The active tab is titled "Risk List - Djar".

The browser's address bar shows the URL `localhost:9527/risks/`. The page content is a Django REST framework response, with the header "Django REST framework" and the username "mahesh.bodas" in the top right corner. The response is a JSON object representing a risk instance, with the following structure:

```
Vary: Accept

{
  "id": 1,
  "risktype": 1,
  "risk_type_name": "Automobile",
  "risk_name": "Toyota 1",
  "risk_description": "Toyota 1 Risk policy",
  "risk_riskfields": [
    {
      "id": 1,
      "risktypefield": 1,
      "risk": 1,
      "risk_type_field_name": "model",
      "risk_type_field_enum": "text",
      "risk_field_value": "TYT1000"
    },
    {
      "id": 2,
      "risktypefield": 2,
      "risk": 1,
      "risk_type_field_name": "doors",
      "risk_type_field_enum": "integer",
      "risk_field_value": "4"
    }
  ]
}
```

Get RiskTypeKeys

The screenshot shows a web browser window with the URL `localhost:9527/risktypekeys/`. The browser's taskbar at the top includes icons for Heroku Node, MaheshBodas, britecore-poc, and several open applications. The Django REST framework header is visible, displaying 'Django REST framework' on the left and 'mahesh.bodas' on the right. The main content area is titled 'Risk Type Key List' and includes a breadcrumb 'Api Root / Risk Type Key List'. To the right of the title are 'OPTIONS' and 'GET' buttons. Below the title, the HTTP method 'GET' and the endpoint path '/risktypekeys/' are shown. The response details indicate a successful 'HTTP 200 OK' status with 'Allow: GET, HEAD, OPTIONS', 'Content-Type: application/json', and 'Vary: Accept'. The response body is a JSON array containing two objects, each representing a risk type key with an 'id' and a 'risk_type_name'.

Heroku Node MaheshBodas britecore-poc Login – Django Ri Login – Django Ri vue-admin-te Inbox (4,938) Risk Type Key X

localhost:9527/risktypekeys/

Django REST framework mahesh.bodas

Api Root / Risk Type Key List

Risk Type Key List

OPTIONS GET

GET /risktypekeys/

HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

```
[
  {
    "id": 1,
    "risk_type_name": "Automobile"
  },
  {
    "id": 2,
    "risk_type_name": "Home"
  }
]
```

Get Risk Key List

The screenshot shows a web browser window with multiple tabs. The active tab is titled "Risk Key List". The address bar shows the URL "localhost:9527/riskkeys/". The browser's developer tools are open, displaying the Django REST framework API interface. The interface shows the endpoint "Risk Key List" with a "GET" method selected. The response is displayed in a code block, showing a JSON array of two risk keys.

Django REST framework maresh.bodas

Api Root / Risk Key List

Risk Key List

OPTIONS GET

GET /riskkeys/

HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

```
[
  {
    "id": 1,
    "risk_name": "Toyota 1"
  },
  {
    "id": 2,
    "risk_name": "HillView"
  }
]
```


<https://django-poc-session-maheshbodas.herokuapp.com/docs/>

The screenshot shows a web browser displaying the API documentation for 'Pastebin API' at the URL <https://django-poc-session-maheshbodas.herokuapp.com/docs/>. The browser's address bar and navigation icons are visible at the top. On the left, a dark sidebar contains a menu with the following items: 'api-token-auth', 'auth', 'riskkeys', 'risks', 'risktypekeys', 'risktypes', 'users', 'Authentication session', and 'Source Code shell'. The main content area has a white background. At the top of this area is the title 'Pastebin API' in a large, bold font, followed by a description: 'A Web API for creating and viewing RiskTypes and RiskInstances based on that.' Below this, the 'api-token-auth' endpoint is highlighted. Underneath, the 'create' action is shown with a 'POST' method and the endpoint path '/api-token-auth/'. To the right of the 'create' action is a green 'INTERACT' button. Further right, a code block contains the following text: '# Load the schema document', '\$ coreapi get https://django-poc-session-maheshbodas', '# Interact with the API endpoint', and '\$ coreapi action api-token-auth create'. Below the 'create' section, the 'auth' endpoint is visible. At the bottom, the 'login > create' action is shown with a 'POST' method and the endpoint path '/auth/login/'. To the right of this action is another green 'INTERACT' button. A code block to the right contains the following text: '# Load the schema document' and '\$ coreapi get https://django-poc-session-maheshbodas'.

Pastebin API

A Web API for creating and viewing RiskTypes and RiskInstances based on that.

api-token-auth

create

POST /api-token-auth/

INTERACT

```
# Load the schema document
$ coreapi get https://django-poc-session-maheshbodas

# Interact with the API endpoint
$ coreapi action api-token-auth create
```

auth

login > create

POST /auth/login/

INTERACT

```
# Load the schema document
$ coreapi get https://django-poc-session-maheshbodas
```

<https://django-poc-session-maheshbodas.herokuapp.com/schema>

The screenshot shows a web browser window with the address bar displaying `https://django-poc-session-maheshbodas.herokuapp.com/schema`. The page title is "Django REST framework" and the user is logged in as "mahesh.bodas". The breadcrumb "Api Root / Schema" is visible. The main heading is "Schema", with "OPTIONS" and "GET" buttons to its right. Below the heading, the method "GET /schema/" is shown. The response details are listed: "HTTP 200 OK", "Allow: GET, HEAD, OPTIONS", "Content-Type: application/coreapi+json", and "Vary: Accept". The JSON response is displayed in a light blue box with syntax highlighting.

```
{
  "_type": "document",
  "_meta": {
    "url": "https://django-poc-session-maheshbodas.herokuapp.com/schema/",
    "title": "Pastebin API"
  },
  "api-token-auth": {
    "create": {
      "_type": "link",
      "url": "/api-token-auth/",
      "action": "post"
    }
  }
}
```