Practical policy-as-code

With Open Policy Agent and Rönd

Ciao! I'm...

Graziano Casto - Developer Relations @ Mia-Platform

Table of Contents.

Table of Contents.

Table of Contents.

01

What I talk about when I talk about policy-as-code.

What exactly policies are?

A policy is a set of rules that govern the behaviour of a software service.

It simply describe invariants that must hold in a software system.

Policy !== Auth*

Authorization is a special kind of policy that often dictates which people or machine can run which action on which resources.

Authentication is how people or machines prove they are who they say they are.

Policy-as-code is an approach to policy management in which policies are defined, updated, and enforced using code.

Why PaC?

Visibility

When policies are defined in code, it's easy for all stakeholders to use the code to understand what is happening within a system.

Why PaC?

Visibility

When policies are defined in code, it's easy for all stakeholders to use the code to understand what is happening within a system.

Accuracy

When teams define and manage policies using code, they avoid the risk of making configuration mistakes when managing a system manually.

Why PaC?

Visibility

When policies are defined in code, it's easy for all stakeholders to use the code to understand what is happening within a system.

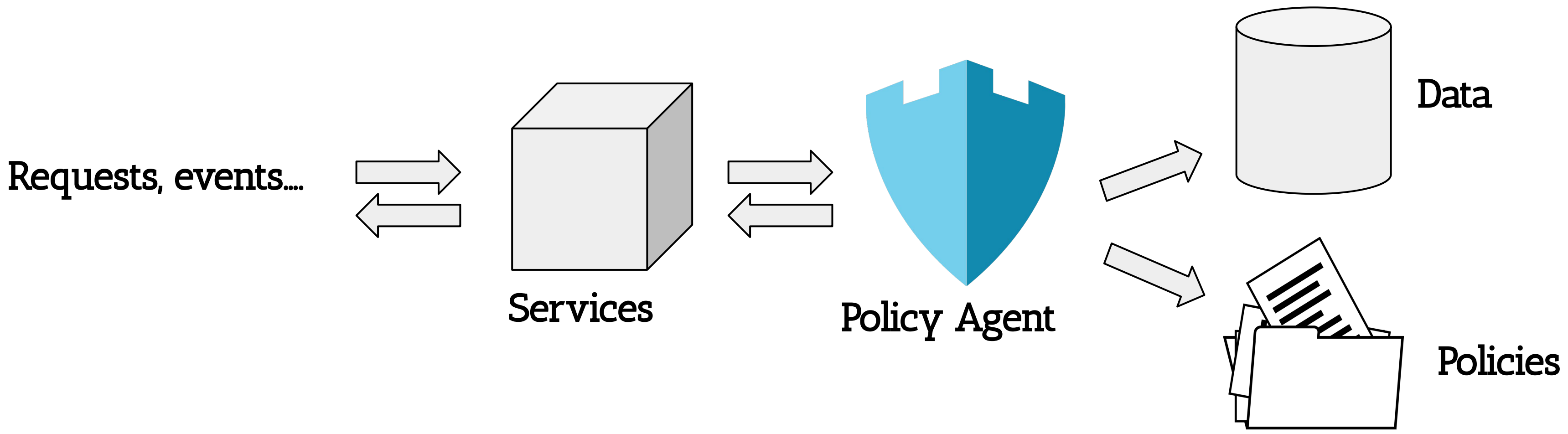
Accuracy

When teams define and manage policies using code, they avoid the risk of making configuration mistakes when managing a system manually.

Testing

When policies are written in code, it's easy to validate them using automated auditing tools.

Policy Decoupling



02

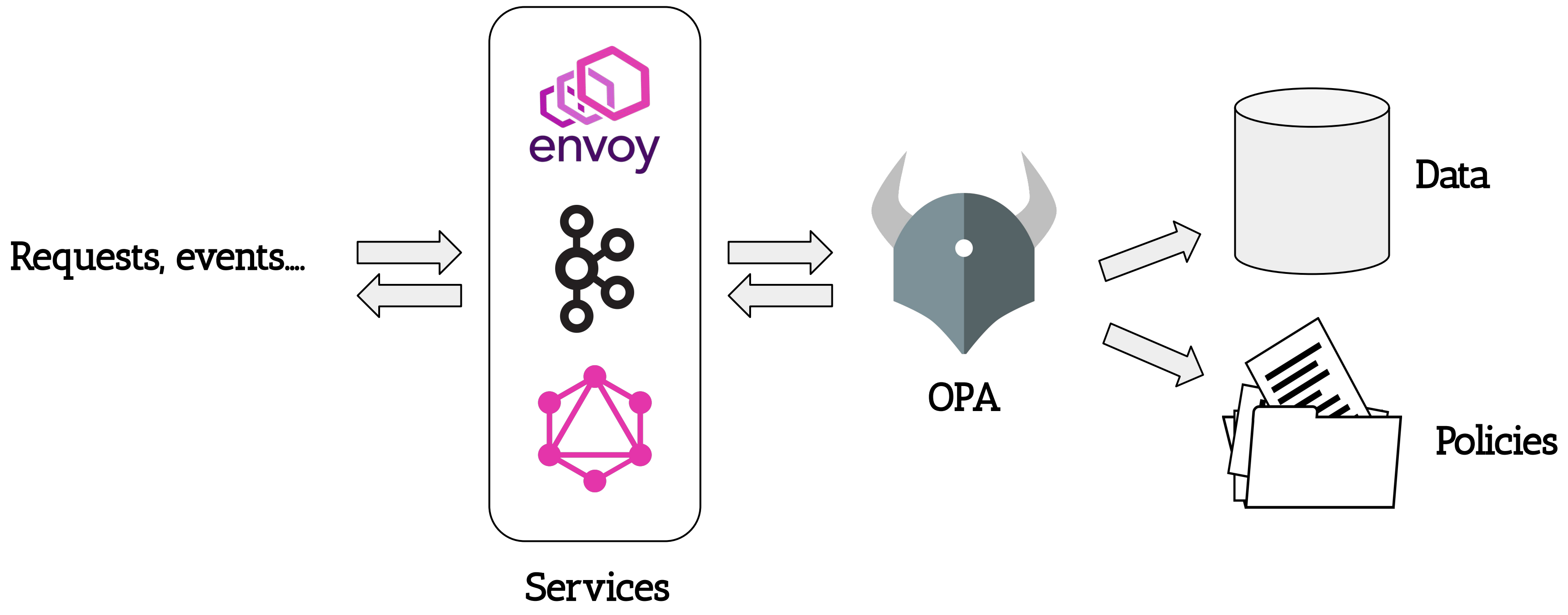
How to implement policy decoupling for API enforcement

Open Policy Agent

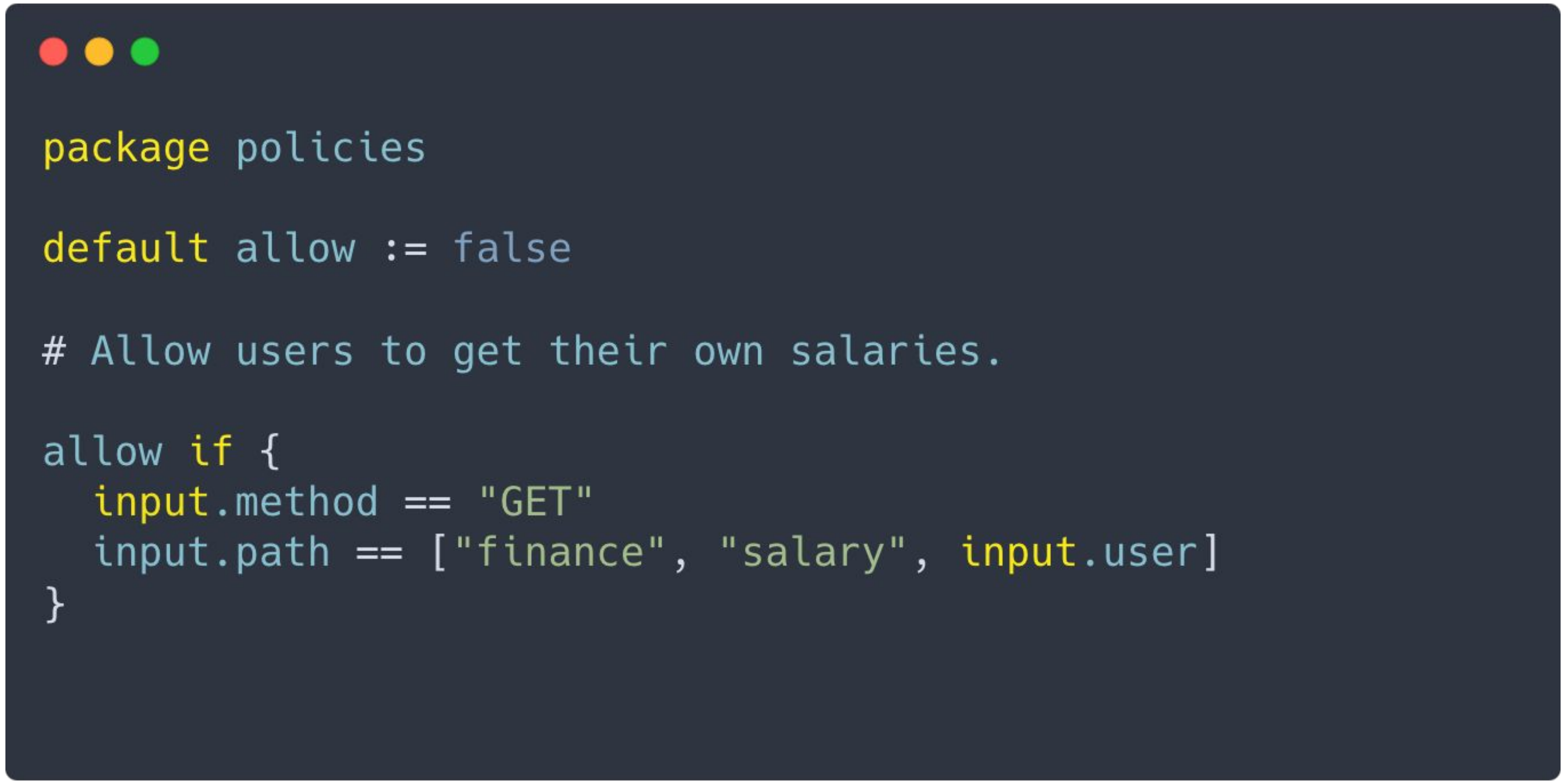
Open Policy Agent (OPA) is an open-source CNCF graduated project.

It provides a high-level declarative language (Rego) that lets you specify policy as code and simple APIs to offload policy decision-making from your software.

Open Policy Agent Architecture



Securing APIs with OPA



```
package policies

default allow := false

# Allow users to get their own salaries.

allow if {
  input.method == "GET"
  input.path == ["finance", "salary", input.user]
}
```

Authentication approach

Role-based access control (RBAC) is a method of regulating access to a specific resource based on the roles of individual users within your organization.

Authentication approach

Role-based access control (RBAC) is a method of regulating access to a specific resource based on the roles of individual users within your organization.

Attribute-based access control (ABAC) is another authorization method built on top of RBAC that allow you to define your security model also on user or request attribute.

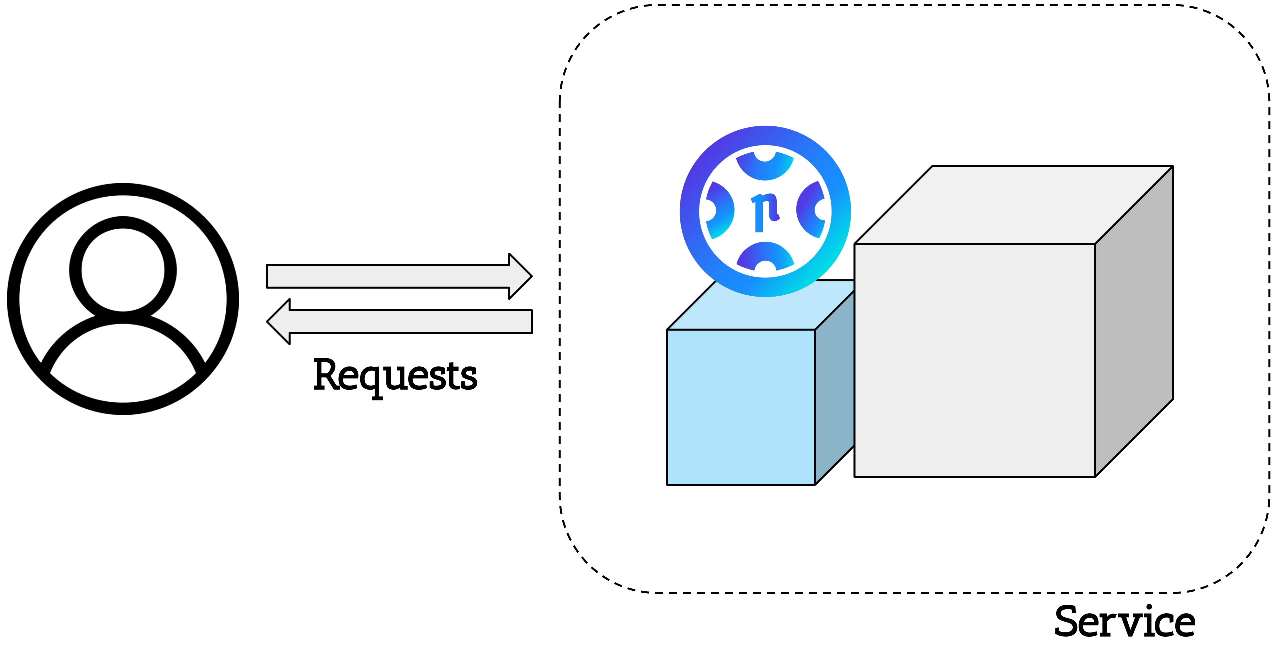
Rönd

It's an open-source lightweight Kubernetes sidecar container that helps you protect your APIs with simple security policies.

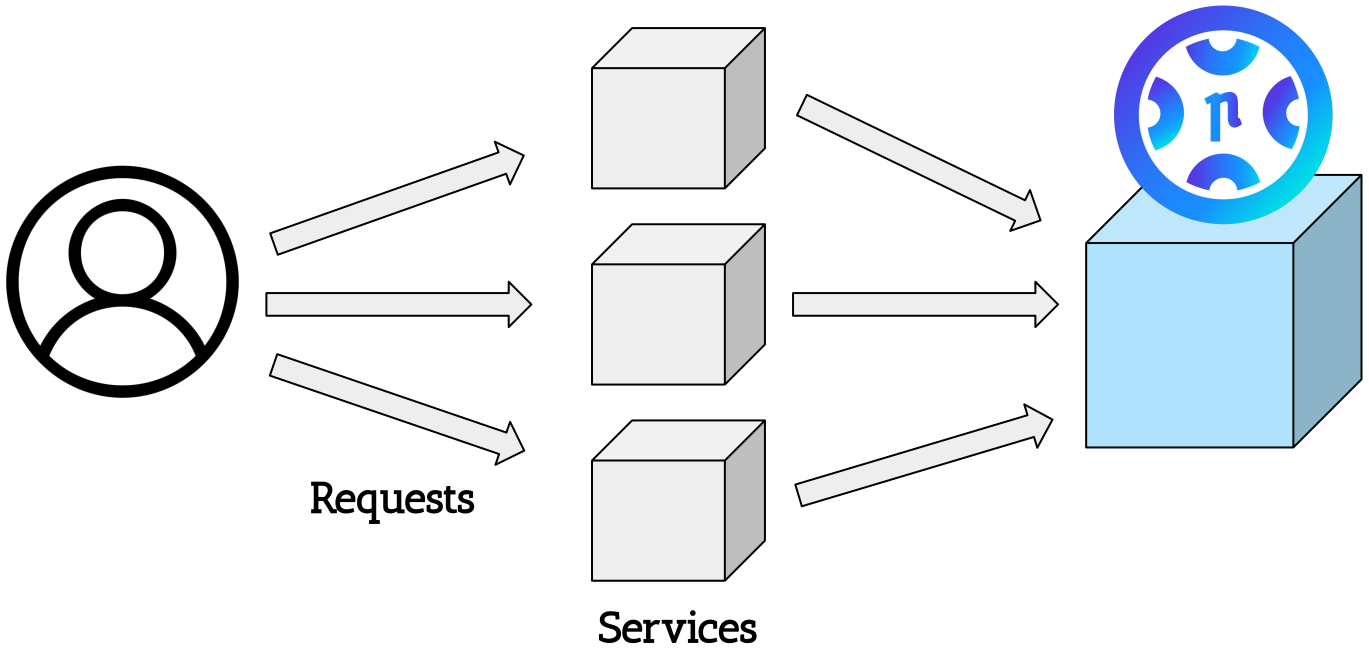
It uses OPA as security engine for validating authorization rules, and leverages Rego language for writing the security policies.

Rönd is an authorization mechanism, but it also natively allows you to build an RBAC solution by defining the concepts of Roles, Permissions, and User Groups as building blocks.

Run as sidecar container...



...or as a standalone service



03

Securing APIs with Rönd: a real world example

The APIs to secure

→ GET /store-info

All requests are accepted even if without an Authorization header.

The APIs to secure

→ GET /store-info

All requests are accepted even if without an Authorization header.

→ POST /receipt

Only authenticated requests are accepted.
Users with role *Salesman* can add only receipts for their store, *StoreManager* can add receipts on the entire chain.

The APIs to secure

→ GET /store-info

All requests are accepted even if without an Authorization header.

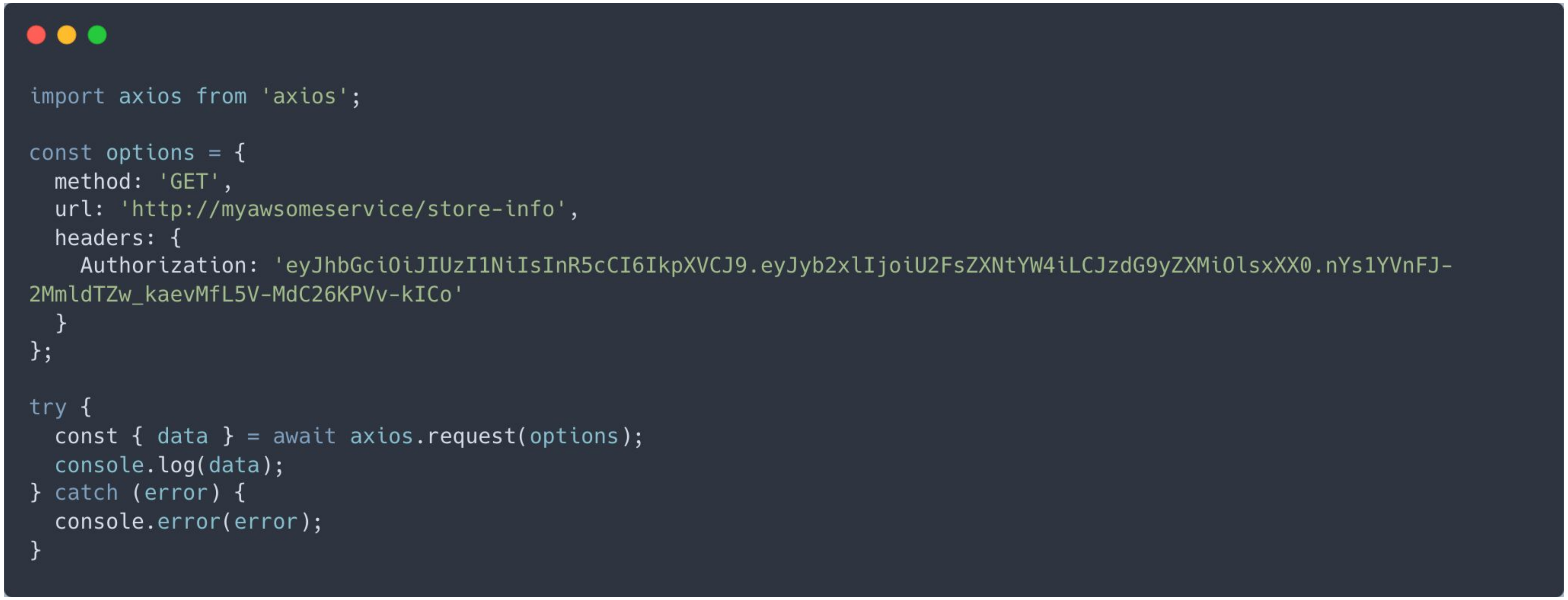
→ POST /receipt

Only authenticated requests are accepted.
Users with role *Salesman* can add only receipts for their store, *StoreManager* can add receipts on the entire chain.

→ GET /receipt

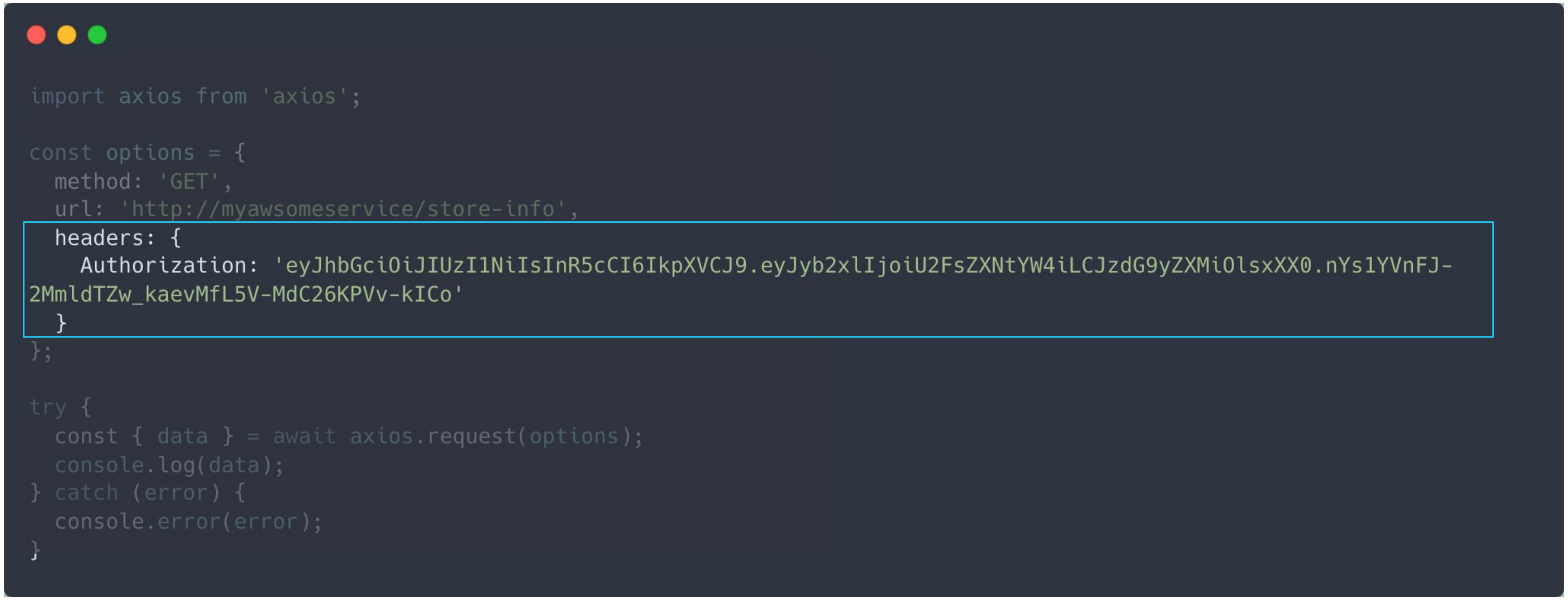
Only authenticated requests are accepted.
Users with role *Salesman* can access the receipts without the payments and customer sensitive data, *StoreManager* can access the receipts also with payment and customer data..

How we get info about user roles?



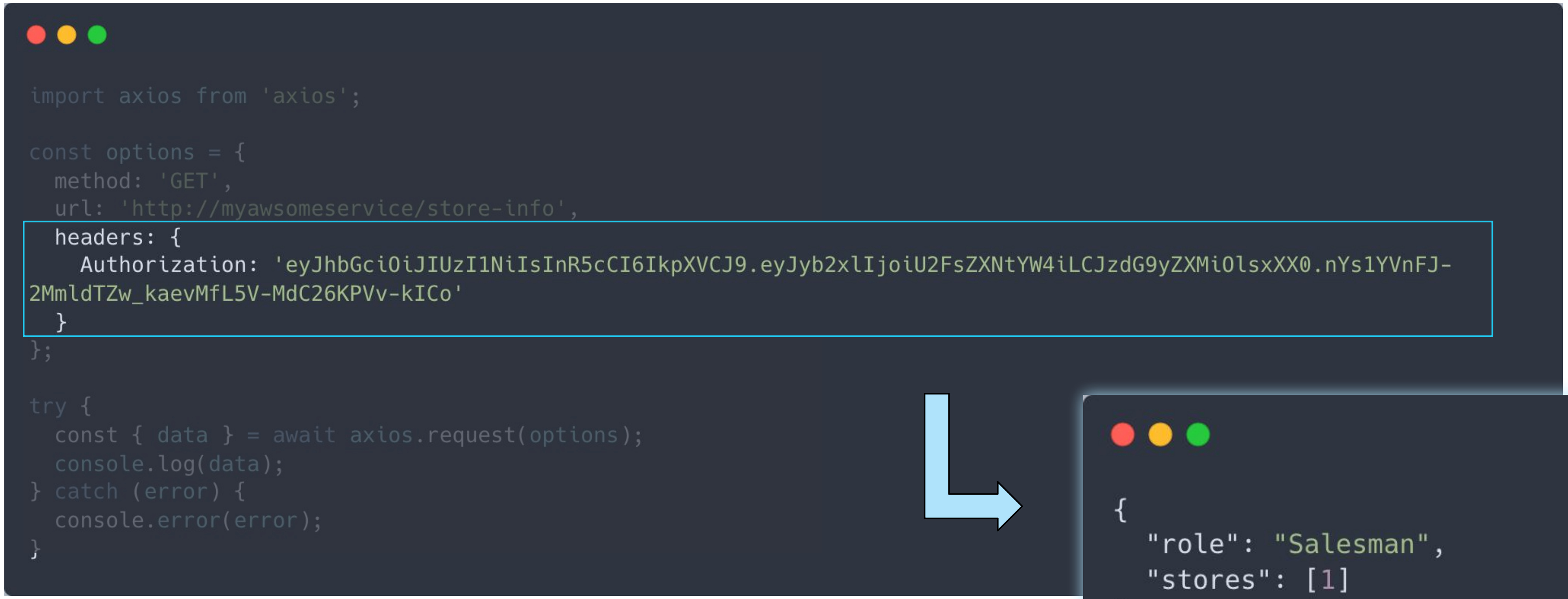
```
import axios from 'axios';

const options = {
  method: 'GET',
  url: 'http://myawsomeservice/store-info',
  headers: {
    Authorization: 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlIjoiU2FsZXNtYW4iLCJzdG9yZXMiOlsxXX0.nYs1YVnFJ-
2MmldTZw_kaevMfL5V-MdC26KPVv-kICo'
  }
};

try {
  const { data } = await axios.request(options);
  console.log(data);
} catch (error) {
  console.error(error);
}
```

How we get info about user roles?

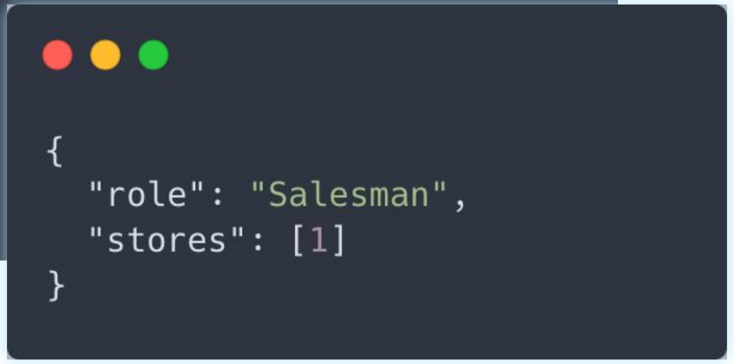


```
import axios from 'axios';

const options = {
  method: 'GET',
  url: 'http://myawsomeservice/store-info',
  headers: {
    Authorization: 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlIjoiU2FsZXNtYW4iLCJzdG9yZXMiOlsxXX0.nYs1YVnFJ-
2MmldTZw_kaevMfL5V-MdC26KPVv-kICo'
  }
};

try {
  const { data } = await axios.request(options);
  console.log(data);
} catch (error) {
  console.error(error);
}
```

How we get info about user roles?

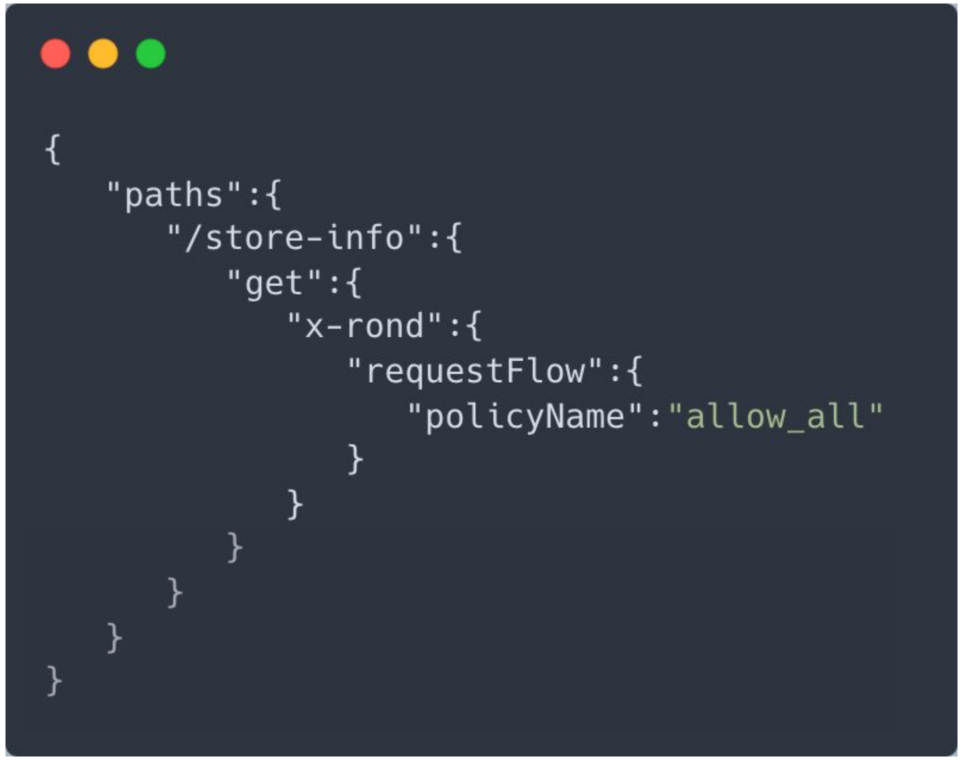


```
import axios from 'axios';

const options = {
  method: 'GET',
  url: 'http://myawsomeservice/store-info',
  headers: {
    Authorization: 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlIjoiU2FsZXNtYW4iLCJzdG9yZXMiOlsxXX0.nYs1YVnFJ-
2MmldTZw_kaevMfL5V-MdC26KPVv-kICo'
  }
};

try {
  const { data } = await axios.request(options);
  console.log(data);
} catch (error) {
  console.error(error);
}
```



```
{
  "role": "Salesman",
  "stores": [1]
}
```

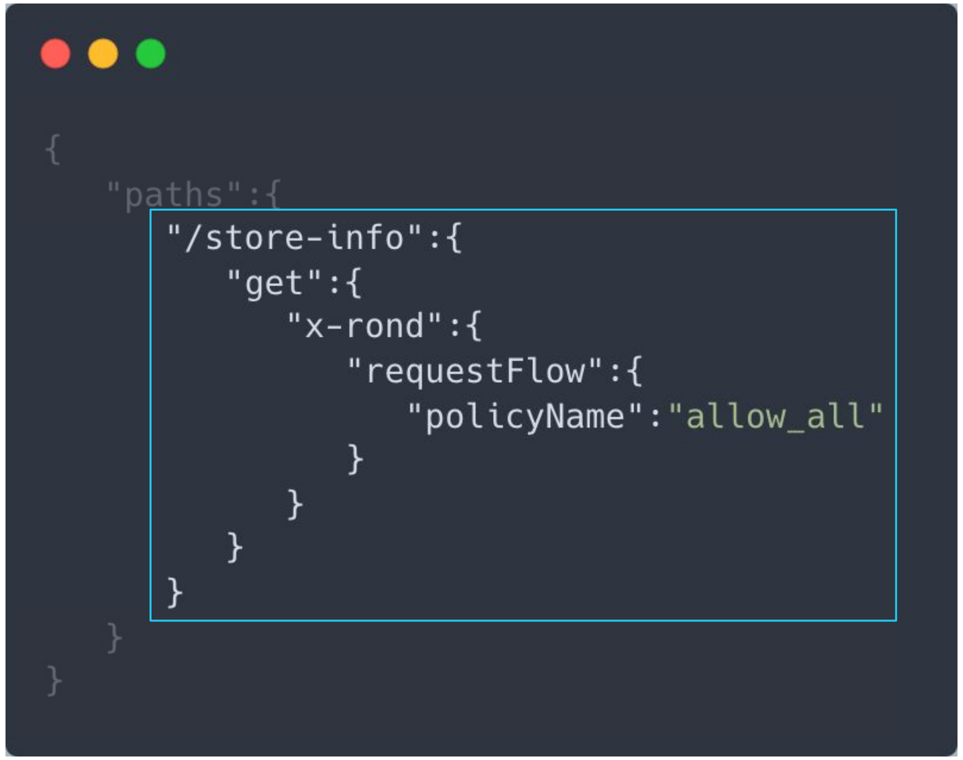
How Rönd configuration works



```
{
    "paths":{
        "/store-info":{
            "get":{
                "x-rond":{
                    "requestFlow":{
                        "policyName":"allow_all"
                    }
                }
            }
        }
    }
}
```

routes.json

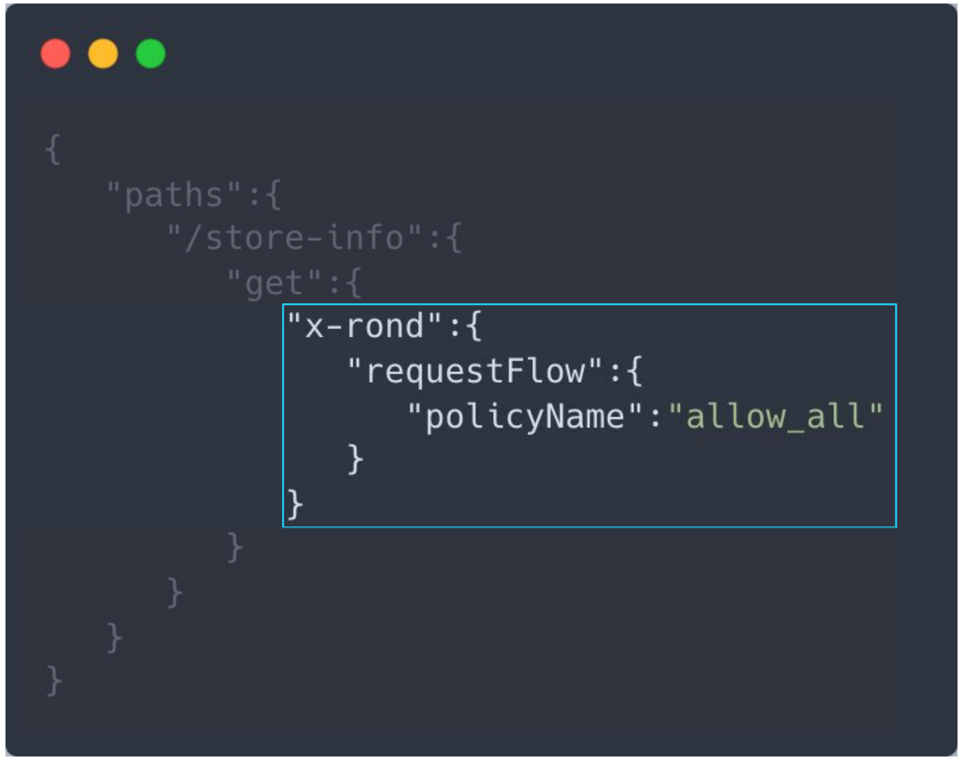
How Rönd configuration works



```
{
    "paths":{
        "/store-info":{
            "get":{
                "x-rond":{
                    "requestFlow":{
                        "policyName":"allow_all"
                    }
                }
            }
        }
    }
}
```

routes.json

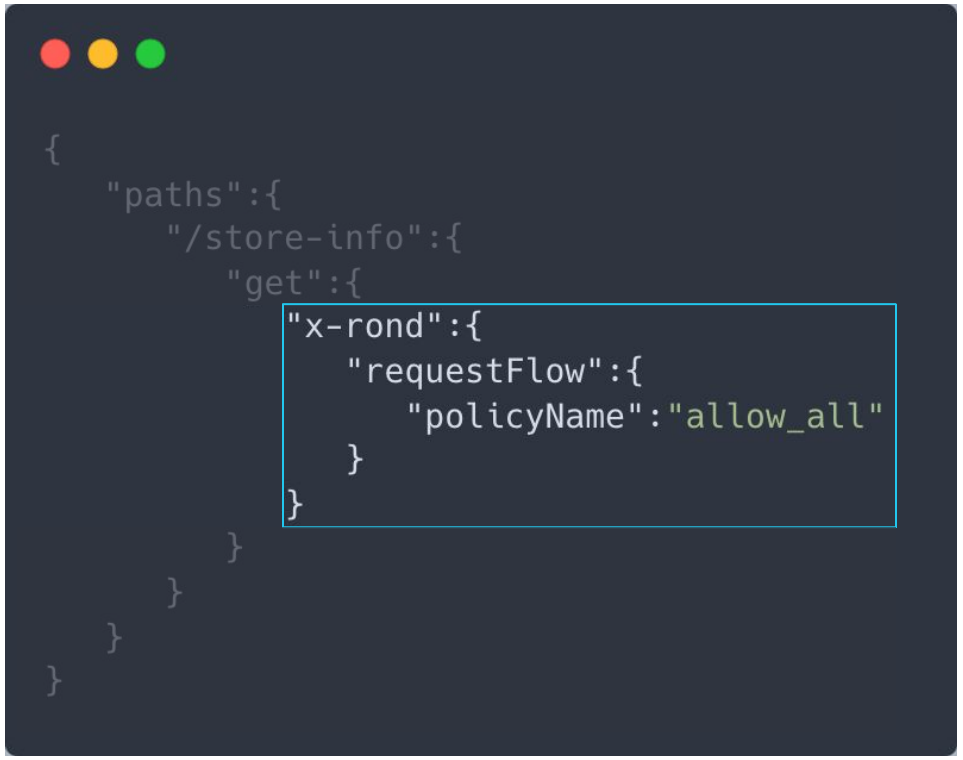
How Rönd configuration works



```
{
   "paths":{
      "/store-info":{
         "get":{
            "x-rond":{
               "requestFlow":{
                  "policyName":"allow_all"
               }
            }
         }
      }
   }
}
```

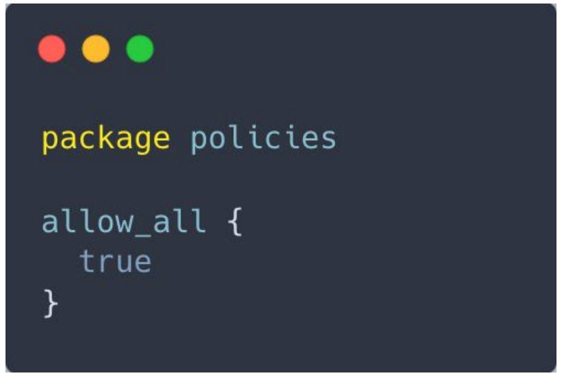
routes.json

How Rönd configuration works



```
{
    "paths":{
        "/store-info":{
            "get":{
                "x-rond":{
                    "requestFlow":{
                        "policyName":"allow_all"
                    }
                }
            }
        }
    }
}
```

routes.json



```
package policies

allow_all {
  true
}
```

policies.rego

The APIs to secure

→ GET /store-info

All requests are accepted even if without an Authorization header.

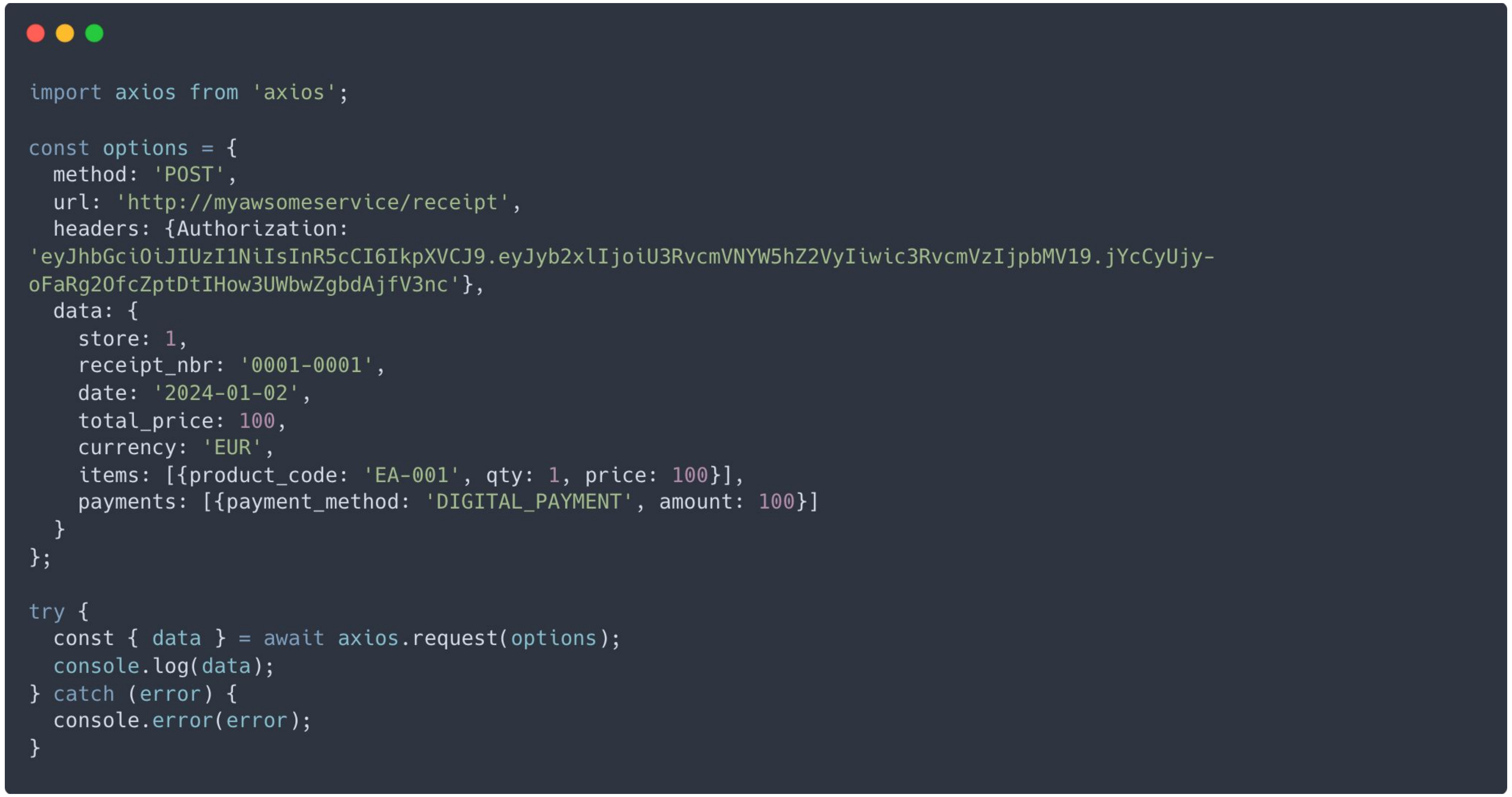
→ POST /receipt

Only authenticated requests are accepted.
Users with role *Salesman* can add only receipts for their store, *StoreManager* can add receipts on the entire chain.

→ GET /receipt

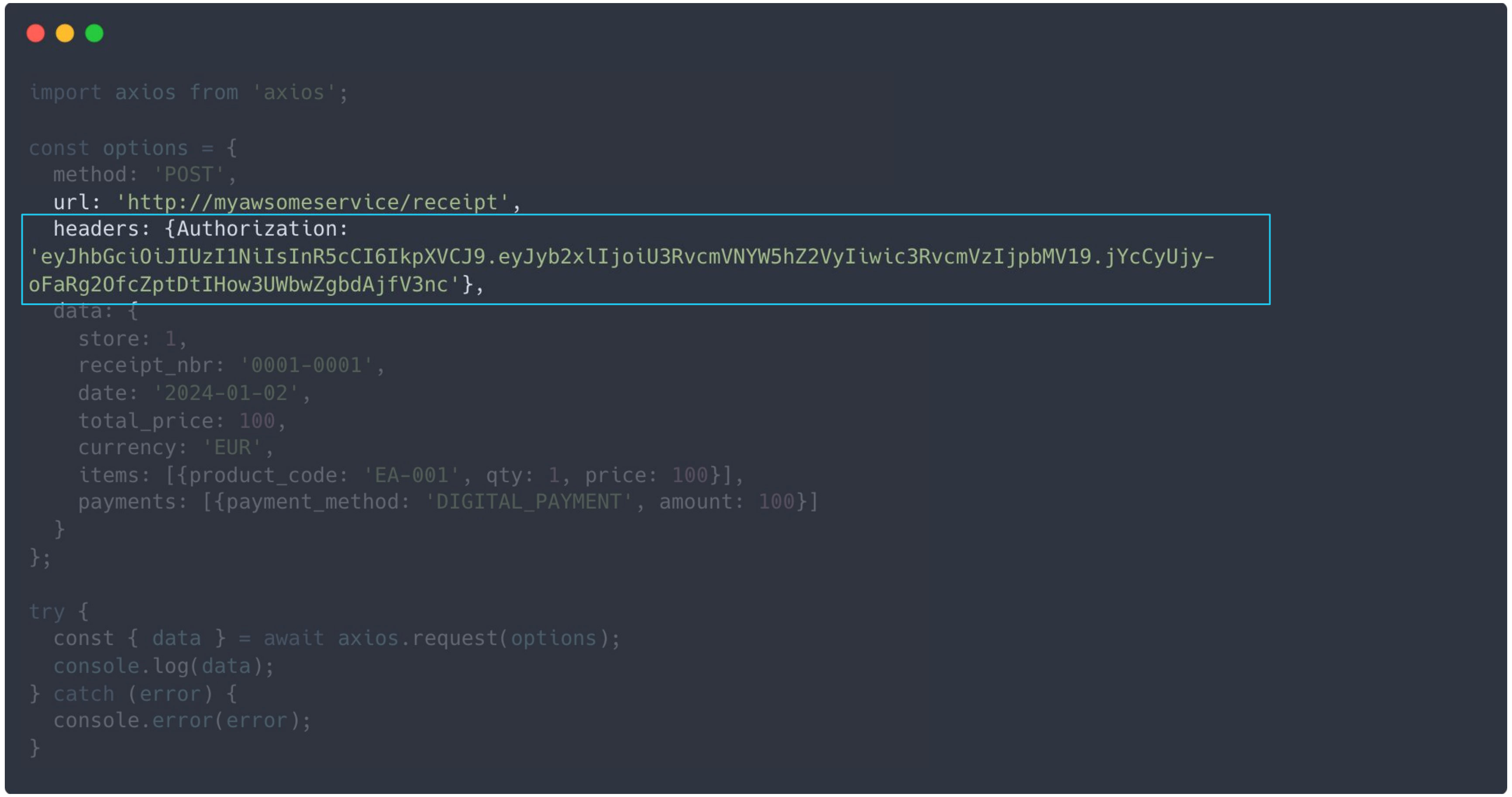
Only authenticated requests are accepted.
Users with role *Salesman* can access the receipts without the payments and customer sensitive data, *StoreManager* can access the receipts also with payment and customer data..

Define a Rego complex policy



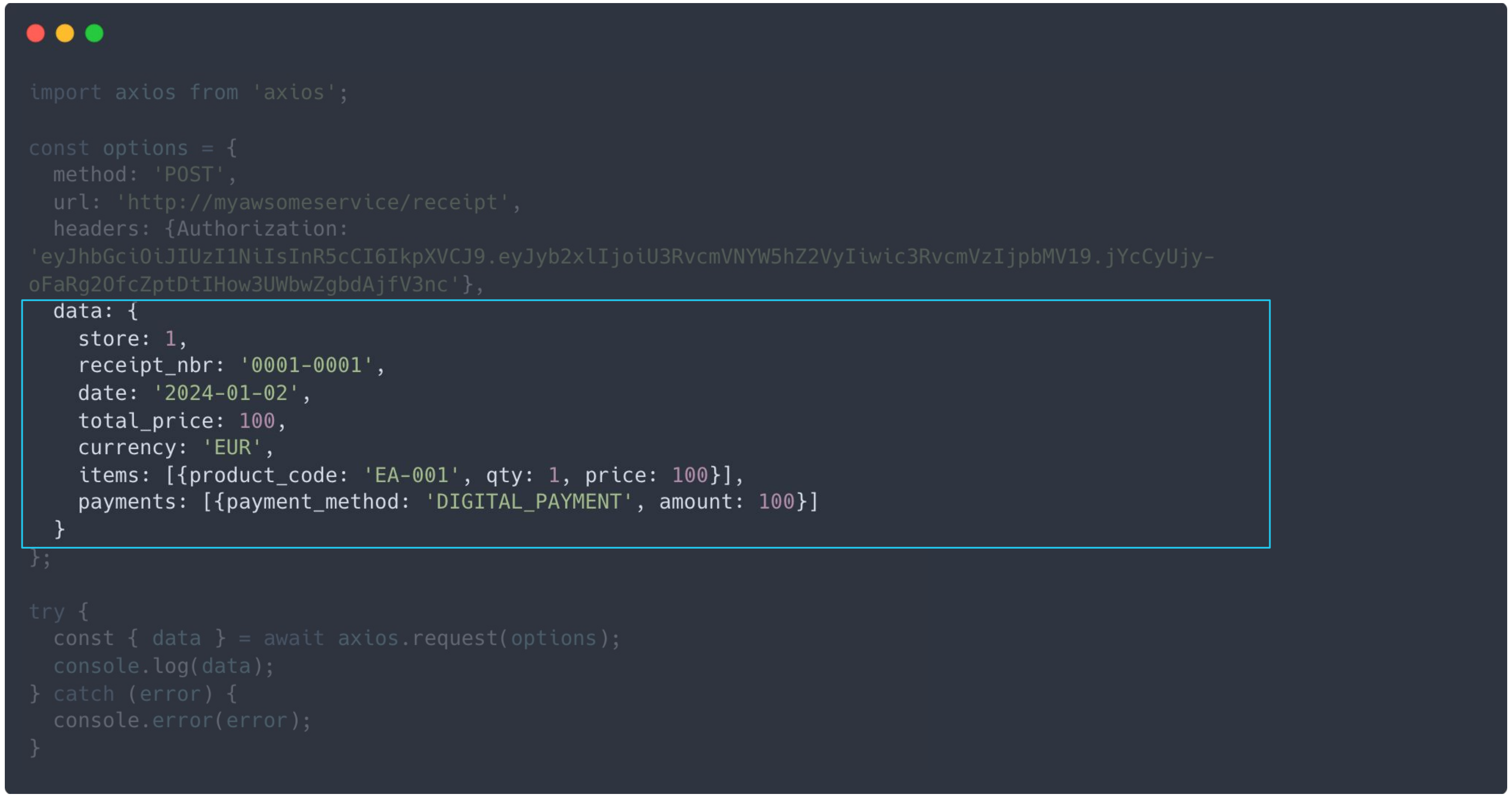
```
import axios from 'axios';

const options = {
  method: 'POST',
  url: 'http://myawsomeservice/receipt',
  headers: {Authorization:
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlIjoiU3RvcmVNYW5hZ2VyIiwic3RvcmVzIjpbMV19.jYcCyUjy-
oFaRg2OfcZptDtIHow3UWbwZgbdAjfV3nc'},
  data: {
    store: 1,
    receipt_nbr: '0001-0001',
    date: '2024-01-02',
    total_price: 100,
    currency: 'EUR',
    items: [{product_code: 'EA-001', qty: 1, price: 100}],
    payments: [{payment_method: 'DIGITAL_PAYMENT', amount: 100}]
  }
};

try {
  const { data } = await axios.request(options);
  console.log(data);
} catch (error) {
  console.error(error);
}
```

Define a Rego complex policy



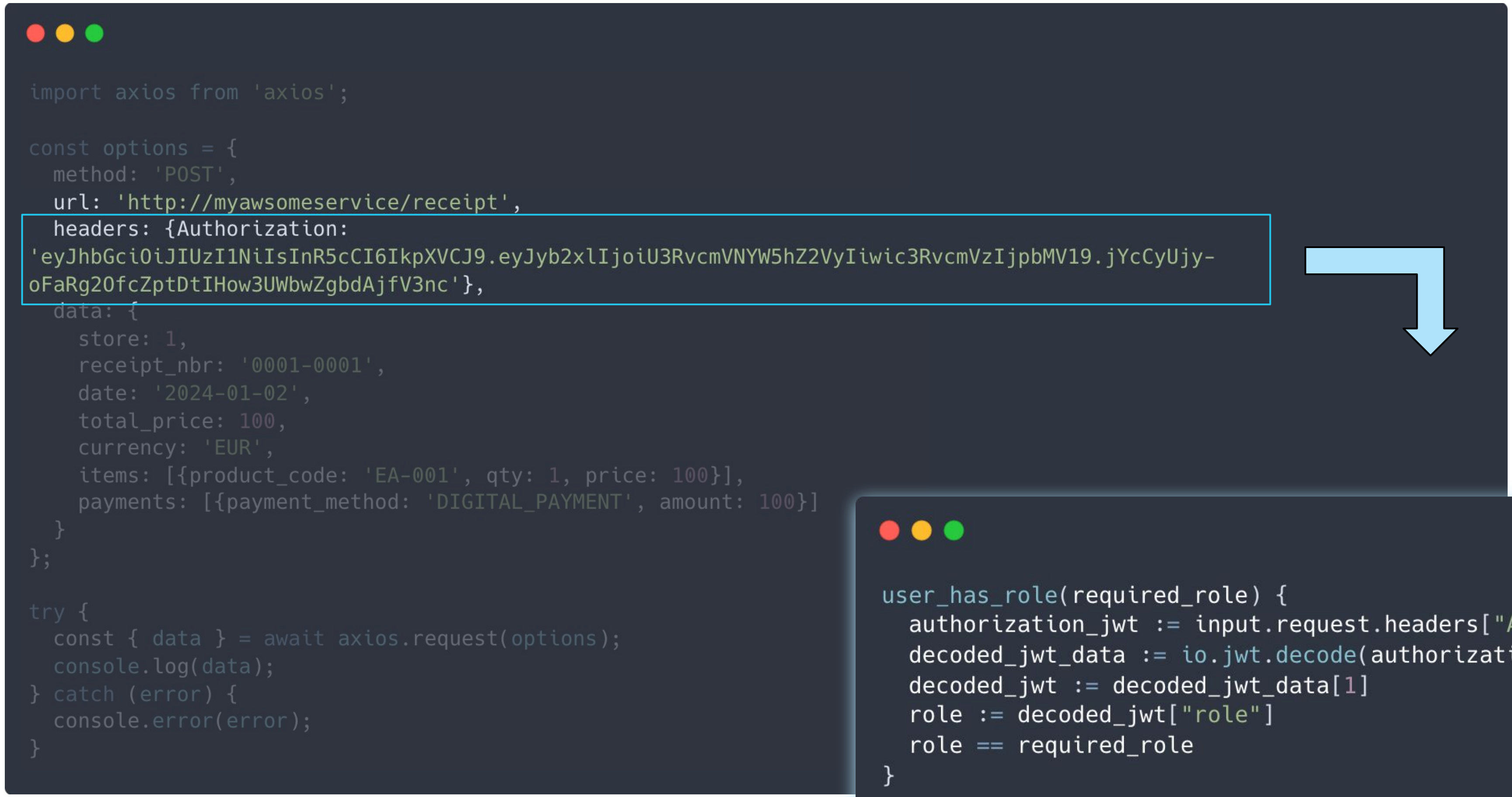
```
import axios from 'axios';

const options = {
  method: 'POST',
  url: 'http://myawsomeservice/receipt',
  headers: {Authorization:
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlIjoiU3RvcmVNYW5hZ2VyIiwic3RvcmVzIjpbMV19.jYcCyUjy-
oFaRg2OfcZptDtIHow3UWbwZgbdAjfV3nc'},
  data: {
    store: 1,
    receipt_nbr: '0001-0001',
    date: '2024-01-02',
    total_price: 100,
    currency: 'EUR',
    items: [{product_code: 'EA-001', qty: 1, price: 100}],
    payments: [{payment_method: 'DIGITAL_PAYMENT', amount: 100}]
  }
};

try {
  const { data } = await axios.request(options);
  console.log(data);
} catch (error) {
  console.error(error);
}
```

Define a Rego complex policy

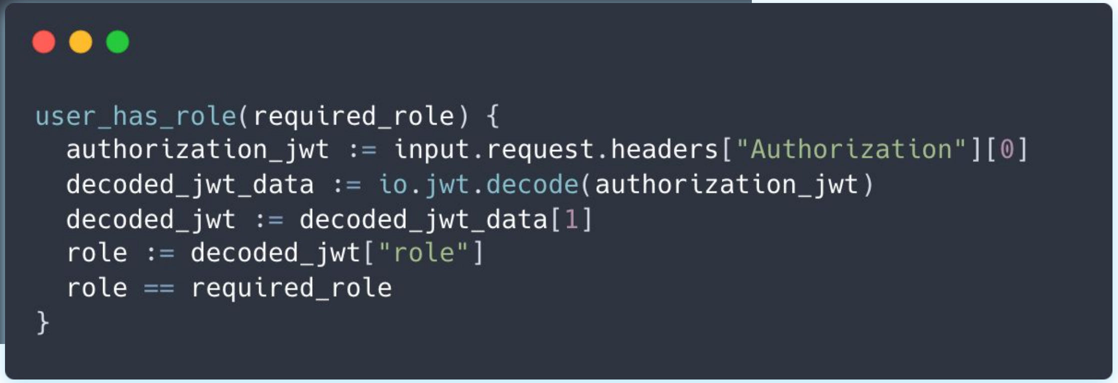


```
import axios from 'axios';

const options = {
  method: 'POST',
  url: 'http://myawsomeservice/receipt',
  headers: {Authorization:
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlIjoiU3RvcmVNYW5hZ2VyIiwic3RvcmVzIjpbMV19.jYcCyUjy-
oFaRg2OfcZptDtIHow3UWbwZgbdAjfV3nc'},
  data: {
    store: 1,
    receipt_nbr: '0001-0001',
    date: '2024-01-02',
    total_price: 100,
    currency: 'EUR',
    items: [{product_code: 'EA-001', qty: 1, price: 100}],
    payments: [{payment_method: 'DIGITAL_PAYMENT', amount: 100}]
  }
};

try {
  const { data } = await axios.request(options);
  console.log(data);
} catch (error) {
  console.error(error);
}
```

Define a Rego complex policy



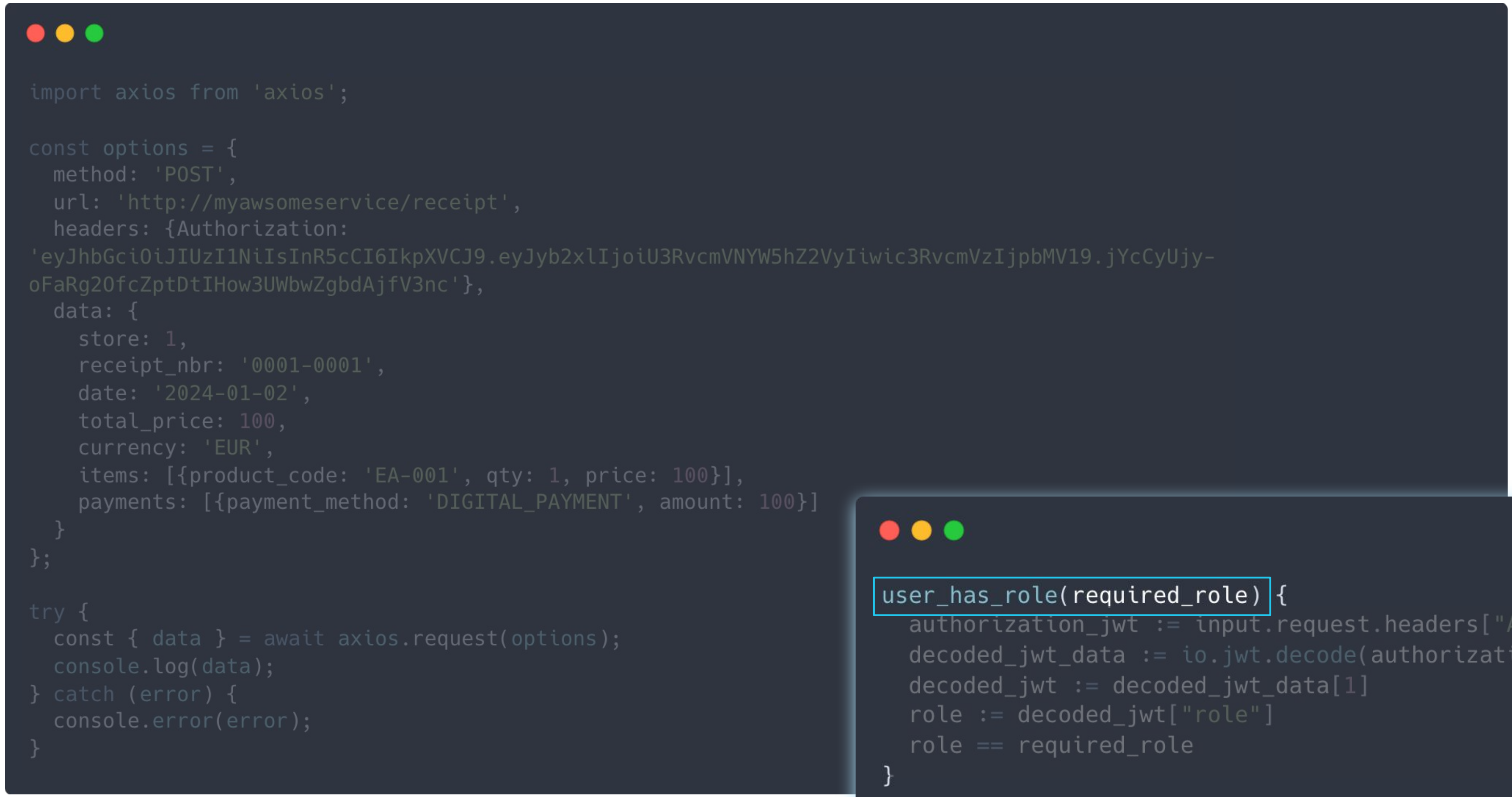
```
import axios from 'axios';

const options = {
  method: 'POST',
  url: 'http://myawsomeservice/receipt',
  headers: {Authorization:
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlIjoiU3RvcmVNYW5hZ2VyIiwic3RvcmVzIjpbMV19.jYcCyUjy-
oFaRg2OfcZptDtIHow3UWbwZgbdAjfV3nc'},
  data: {
    store: 1,
    receipt_nbr: '0001-0001',
    date: '2024-01-02',
    total_price: 100,
    currency: 'EUR',
    items: [{product_code: 'EA-001', qty: 1, price: 100}],
    payments: [{payment_method: 'DIGITAL_PAYMENT', amount: 100}]
  }
};

try {
  const { data } = await axios.request(options);
  console.log(data);
} catch (error) {
  console.error(error);
}
```



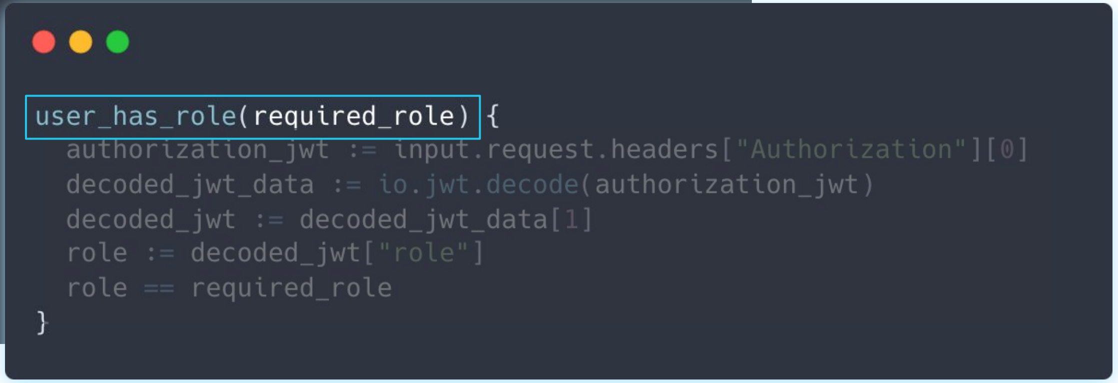
```
user_has_role(required_role) {
  authorization_jwt := input.request.headers["Authorization"][0]
  decoded_jwt_data := io.jwt.decode(authorization_jwt)
  decoded_jwt := decoded_jwt_data[1]
  role := decoded_jwt["role"]
  role == required_role
}
```

Define a Rego function to check if the role of the user performing the request is the same required to perform the requested action.

Define a Rego complex policy



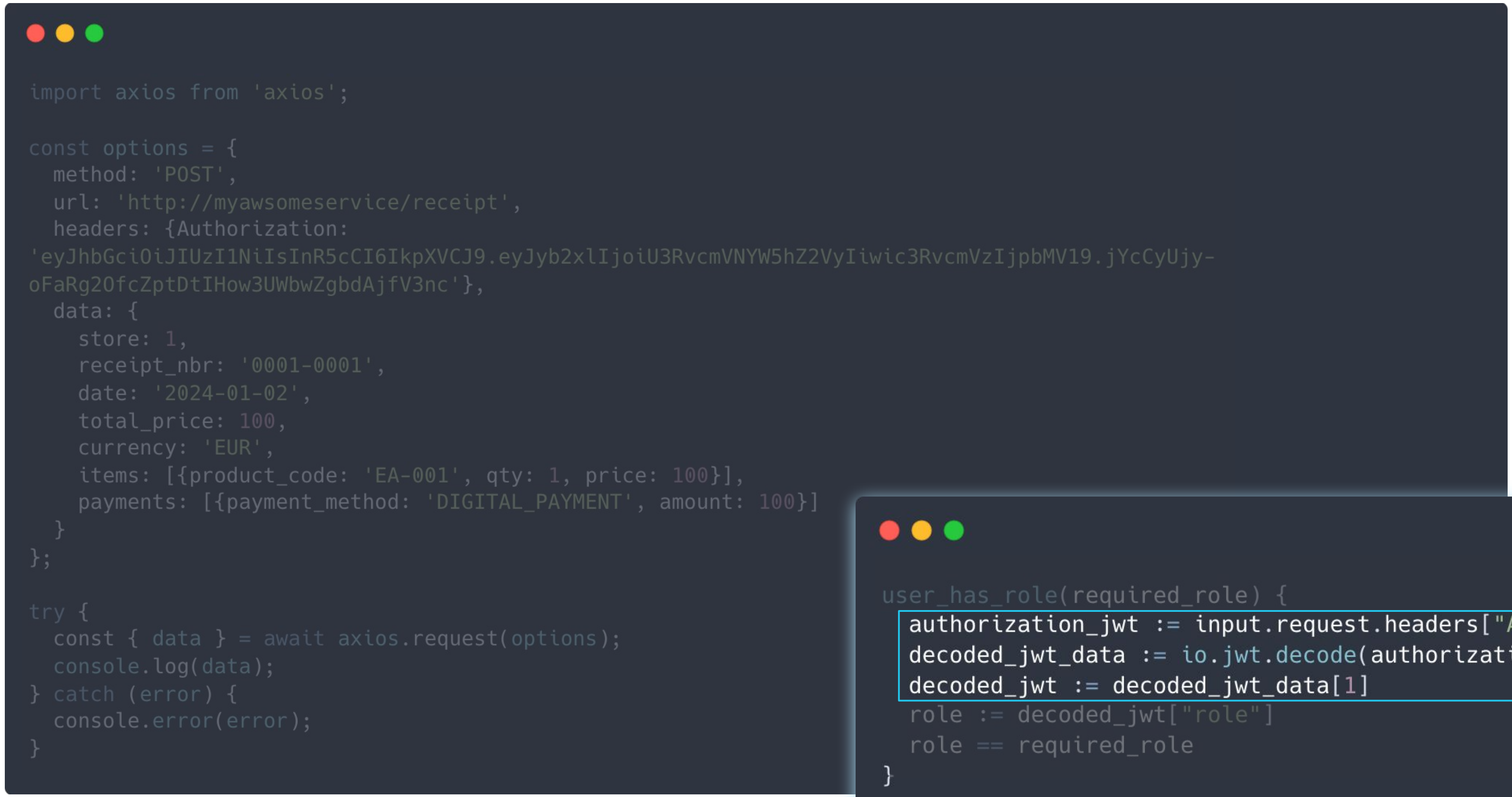
```
import axios from 'axios';

const options = {
  method: 'POST',
  url: 'http://myawsomeservice/receipt',
  headers: {Authorization:
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlIjoiU3RvcmVNYW5hZ2VyIiwic3RvcmVzIjpbMV19.jYcCyUjy-
oFaRg2OfcZptDtIHow3UWbwZgbdAjfV3nc'},
  data: {
    store: 1,
    receipt_nbr: '0001-0001',
    date: '2024-01-02',
    total_price: 100,
    currency: 'EUR',
    items: [{product_code: 'EA-001', qty: 1, price: 100}],
    payments: [{payment_method: 'DIGITAL_PAYMENT', amount: 100}]
  }
};

try {
  const { data } = await axios.request(options);
  console.log(data);
} catch (error) {
  console.error(error);
}
```



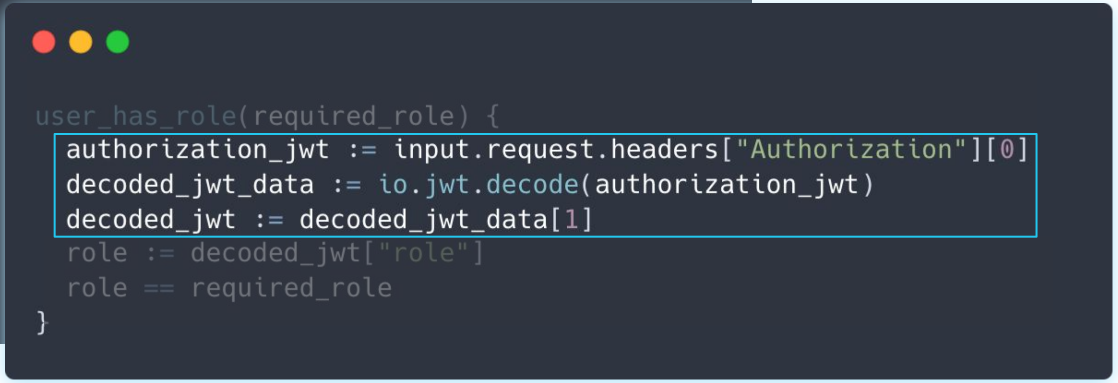
```
user_has_role(required_role) {
  authorization_jwt := input.request.headers["Authorization"][0]
  decoded_jwt_data := io.jwt.decode(authorization_jwt)
  decoded_jwt := decoded_jwt_data[1]
  role := decoded_jwt["role"]
  role == required_role
}
```

Define a Rego function to check if the role of the user performing the request is the same required to perform the requested action.

Define a Rego complex policy



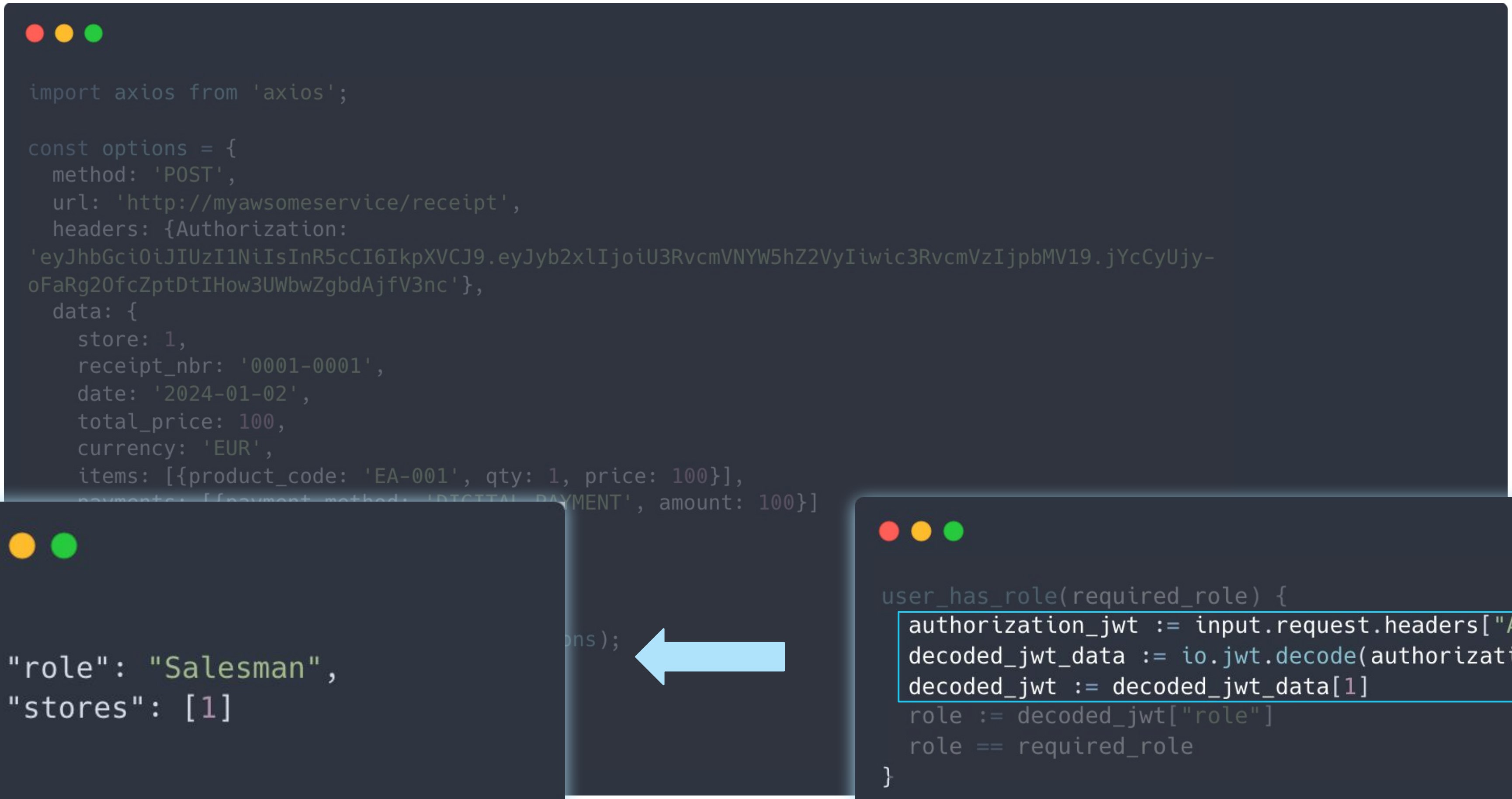
```
import axios from 'axios';

const options = {
  method: 'POST',
  url: 'http://myawsomeservice/receipt',
  headers: {Authorization:
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlIjoiU3RvcmVNYW5hZ2VyIiwic3RvcmVzIjpbMV19.jYcCyUjy-
oFaRg2OfcZptDtIHow3UWbwZgbdAjfV3nc'},
  data: {
    store: 1,
    receipt_nbr: '0001-0001',
    date: '2024-01-02',
    total_price: 100,
    currency: 'EUR',
    items: [{product_code: 'EA-001', qty: 1, price: 100}],
    payments: [{payment_method: 'DIGITAL_PAYMENT', amount: 100}]
  }
};

try {
  const { data } = await axios.request(options);
  console.log(data);
} catch (error) {
  console.error(error);
}
```



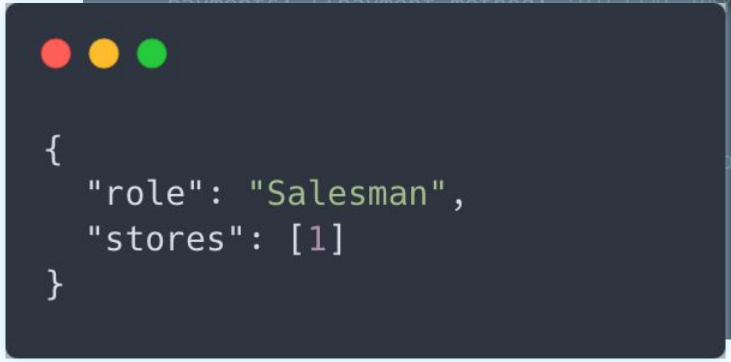
```
user_has_role(required_role) {
  authorization_jwt := input.request.headers["Authorization"][0]
  decoded_jwt_data := io.jwt.decode(authorization_jwt)
  decoded_jwt := decoded_jwt_data[1]
  role := decoded_jwt["role"]
  role == required_role
}
```

Define a Rego function to check if the role of the user performing the request is the same required to perform the requested action.

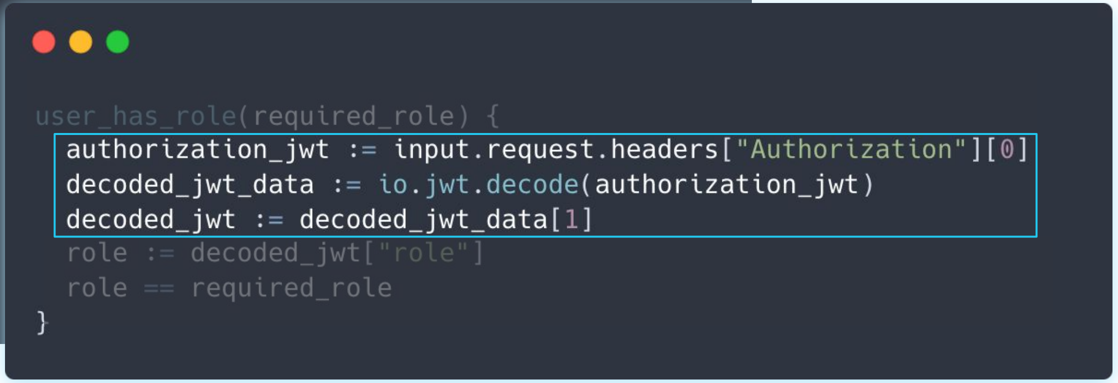
Define a Rego complex policy



```
import axios from 'axios';

const options = {
  method: 'POST',
  url: 'http://myawsomeservice/receipt',
  headers: {Authorization:
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlIjoiU3RvcmVNYW5hZ2VyIiwic3RvcmVzIjpbMV19.jYcCyUjy-
oFaRg2OfcZptDtIHow3UWbwZgbdAjfV3nc'},
  data: {
    store: 1,
    receipt_nbr: '0001-0001',
    date: '2024-01-02',
    total_price: 100,
    currency: 'EUR',
    items: [{product_code: 'EA-001', qty: 1, price: 100}],
    payments: [{payment_method: 'DIGITAL_PAYMENT', amount: 100}]

ons);
```



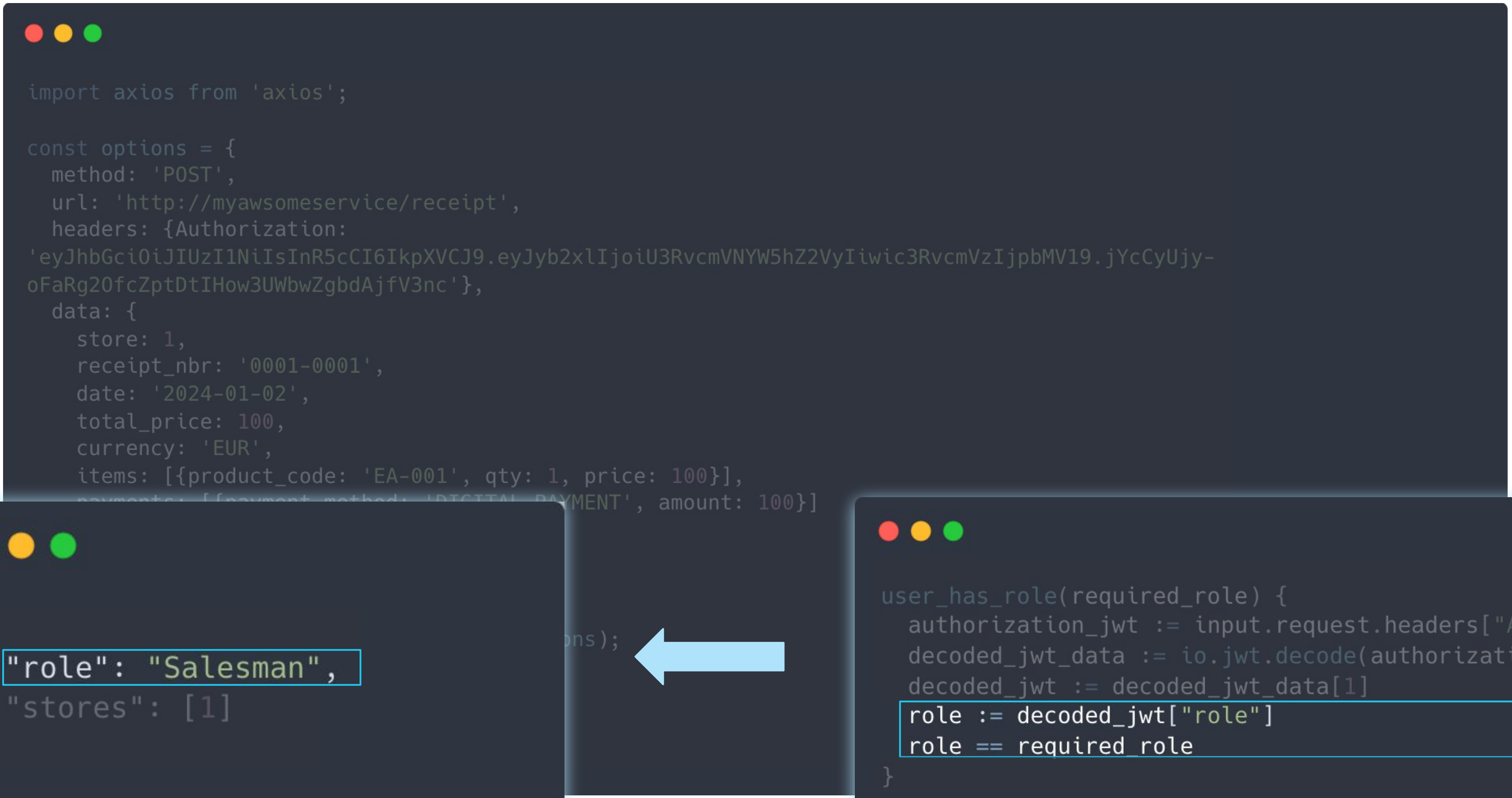
```
{
  "role": "Salesman",
  "stores": [1]
}
```



```
user_has_role(required_role) {
  authorization_jwt := input.request.headers["Authorization"][0]
  decoded_jwt_data := io.jwt.decode(authorization_jwt)
  decoded_jwt := decoded_jwt_data[1]
  role := decoded_jwt["role"]
  role == required_role
}
```

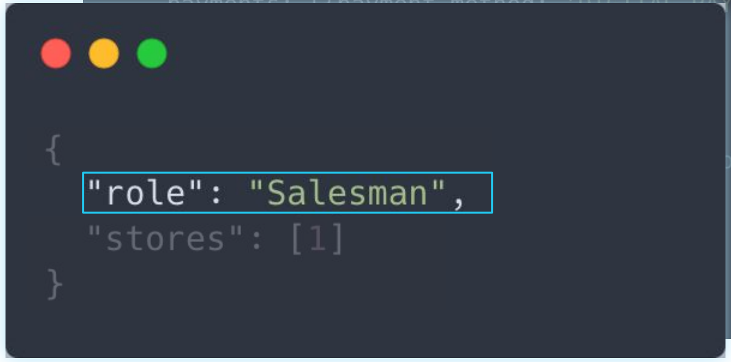
Define a Rego function to check if the role of the user performing the request is the same required to perform the requested action.

Define a Rego complex policy

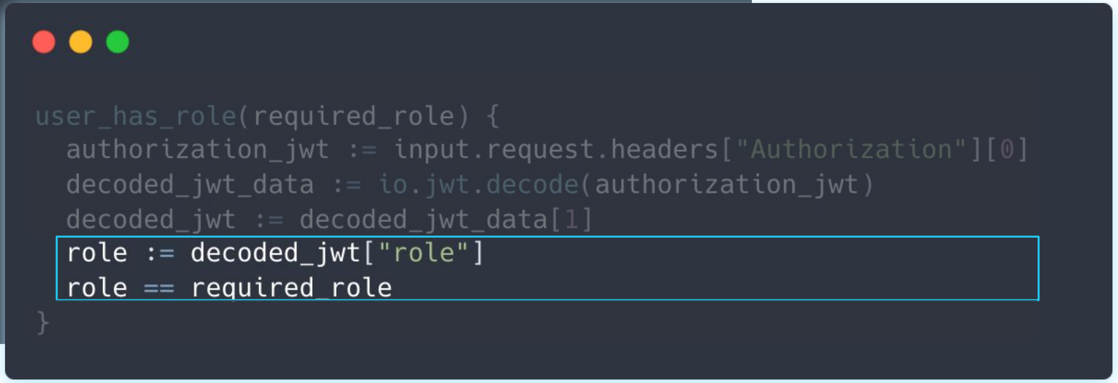


```
import axios from 'axios';

const options = {
  method: 'POST',
  url: 'http://myawsomeservice/receipt',
  headers: {Authorization:
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlIjoiU3RvcmVNYW5hZ2VyIiwic3RvcmVzIjpbMV19.jYcCyUjy-
oFaRg2OfcZptDtIHow3UWbwZgbdAjfV3nc'},
  data: {
    store: 1,
    receipt_nbr: '0001-0001',
    date: '2024-01-02',
    total_price: 100,
    currency: 'EUR',
    items: [{product_code: 'EA-001', qty: 1, price: 100}],
    payments: [{payment_method: 'DIGITAL_PAYMENT', amount: 100}]
```

```
ns);
```



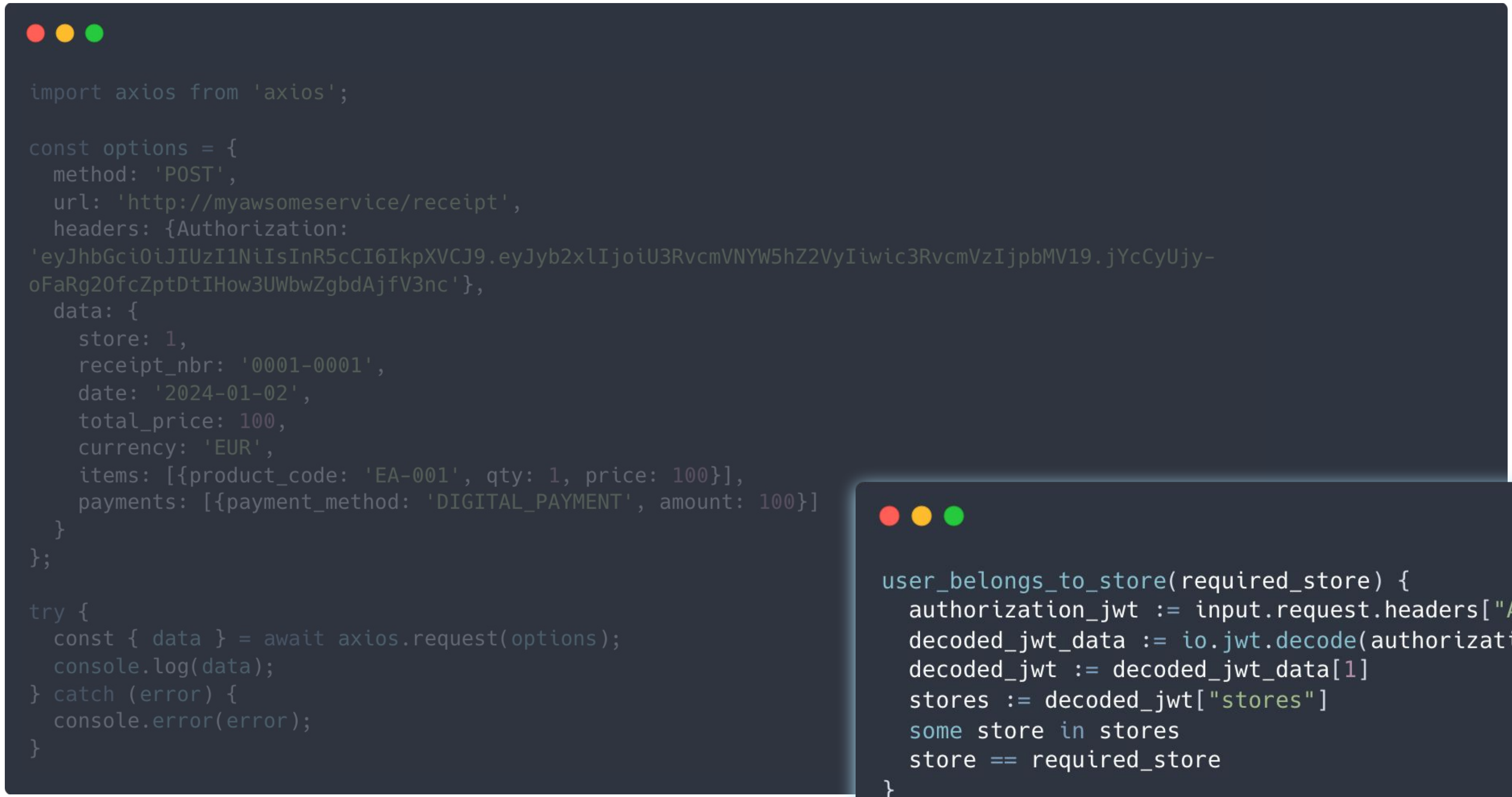
```
{
  "role": "Salesman",
  "stores": [1]
}
```



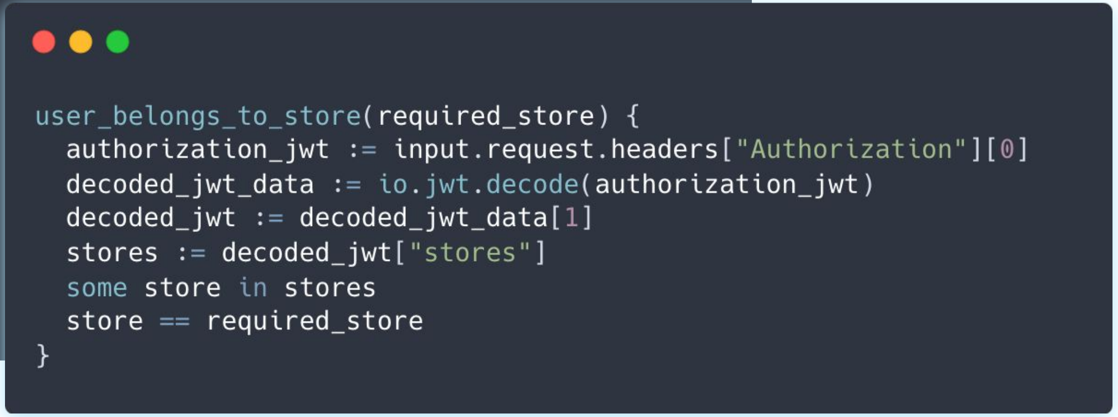
```
user_has_role(required_role) {
  authorization_jwt := input.request.headers["Authorization"][0]
  decoded_jwt_data := io.jwt.decode(authorization_jwt)
  decoded_jwt := decoded_jwt_data[1]
  role := decoded_jwt["role"]
  role == required_role
}
```

Define a Rego function to check if the role of the user performing the request is the same required to perform the requested action.

Define a Rego complex policy



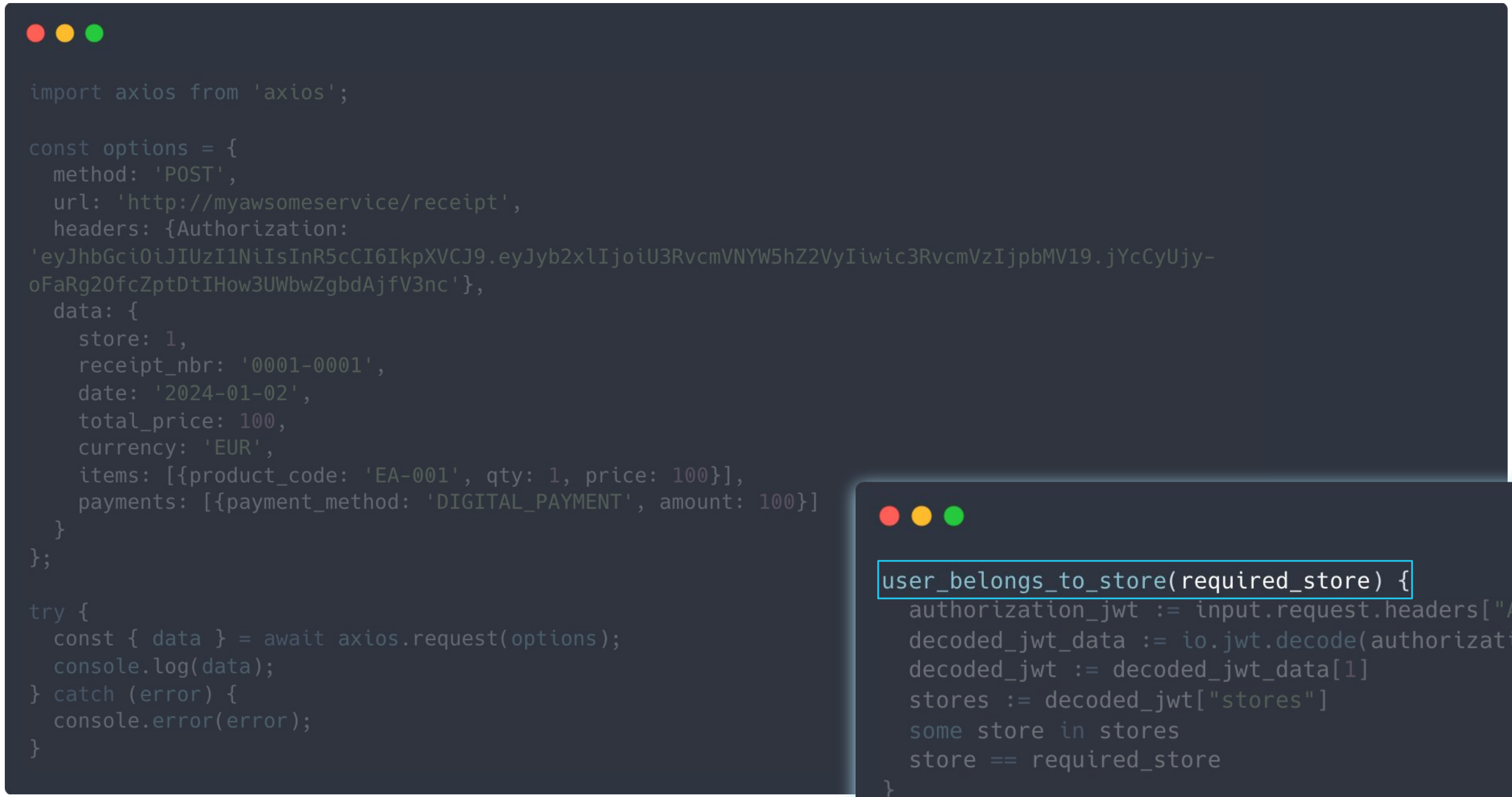
```
import axios from 'axios';

const options = {
  method: 'POST',
  url: 'http://myawsomeservice/receipt',
  headers: {Authorization:
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlIjoiU3RvcmVNYW5hZ2VyIiwic3RvcmVzIjpbMV19.jYcCyUjy-
oFaRg2OfcZptDtIHow3UWbwZgbdAjfV3nc'},
  data: {
    store: 1,
    receipt_nbr: '0001-0001',
    date: '2024-01-02',
    total_price: 100,
    currency: 'EUR',
    items: [{product_code: 'EA-001', qty: 1, price: 100}],
    payments: [{payment_method: 'DIGITAL_PAYMENT', amount: 100}]
  }
};

try {
  const { data } = await axios.request(options);
  console.log(data);
} catch (error) {
  console.error(error);
}
```



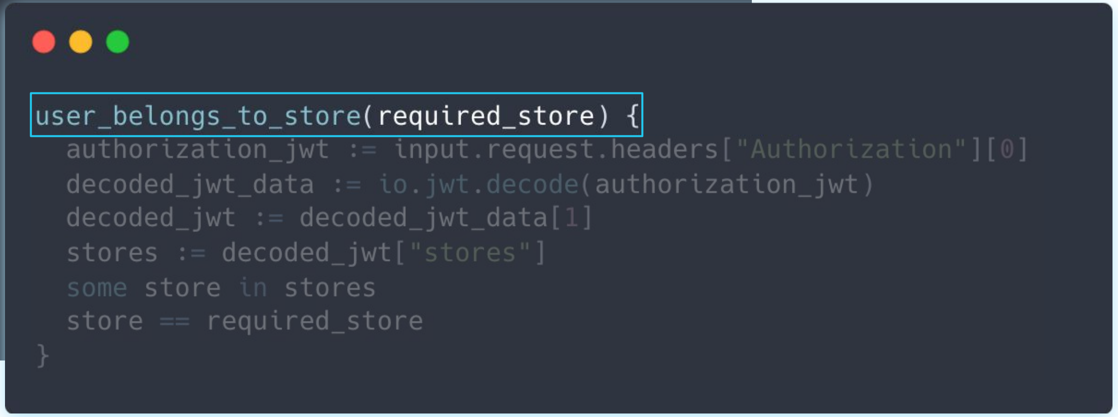
```
user_belongs_to_store(required_store) {
  authorization_jwt := input.request.headers["Authorization"][0]
  decoded_jwt_data := io.jwt.decode(authorization_jwt)
  decoded_jwt := decoded_jwt_data[1]
  stores := decoded_jwt["stores"]
  some store in stores
  store == required_store
}
```

Define a Rego function to check if the user is assigned to the store which the request insists on.

Define a Rego complex policy



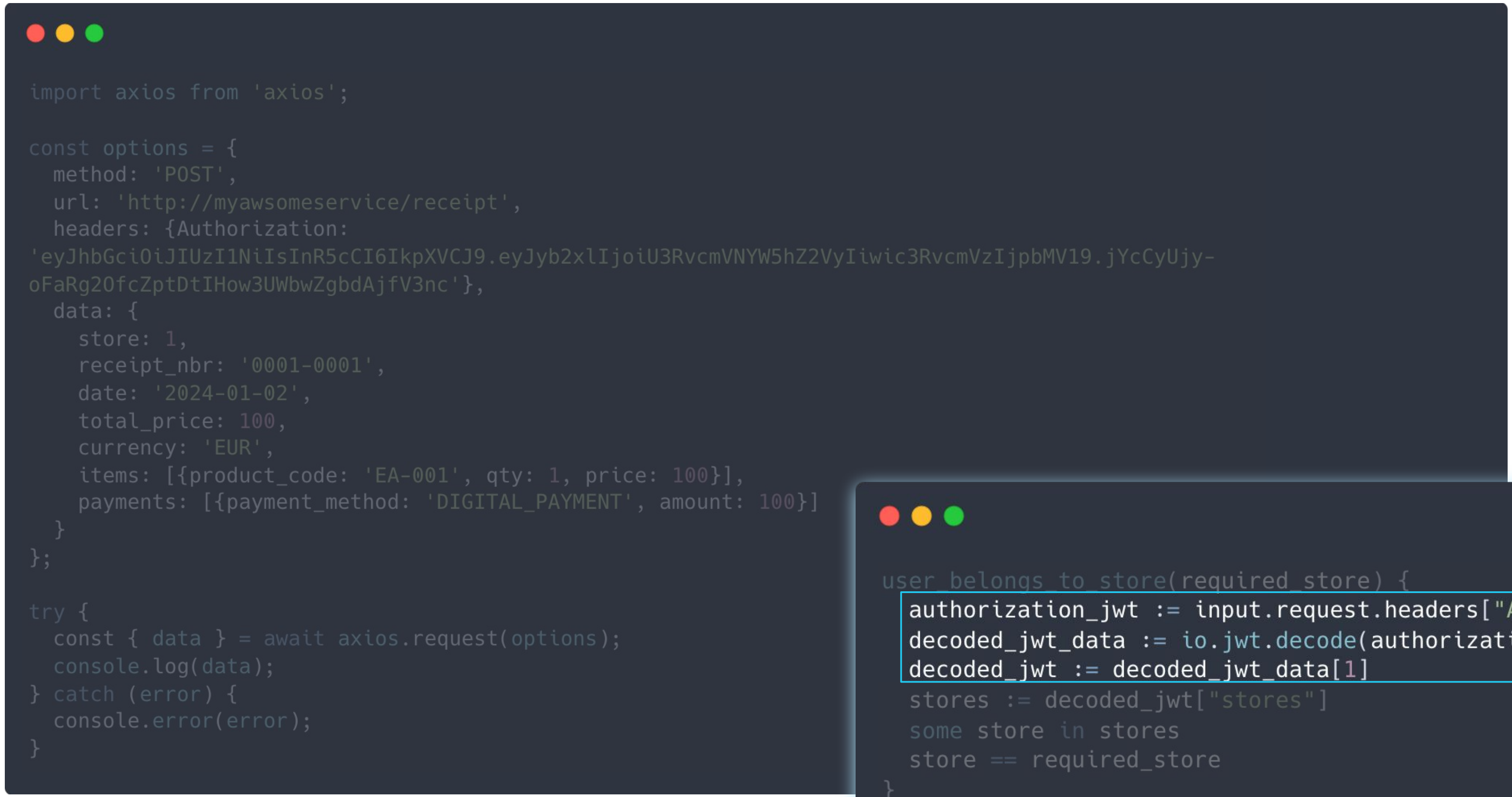
```
import axios from 'axios';

const options = {
  method: 'POST',
  url: 'http://myawsomeservice/receipt',
  headers: {Authorization:
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlIjoiU3RvcmVNYW5hZ2VyIiwic3RvcmVzIjpbMV19.jYcCyUjy-
oFaRg2OfcZptDtIHow3UWbwZgbdAjfV3nc'},
  data: {
    store: 1,
    receipt_nbr: '0001-0001',
    date: '2024-01-02',
    total_price: 100,
    currency: 'EUR',
    items: [{product_code: 'EA-001', qty: 1, price: 100}],
    payments: [{payment_method: 'DIGITAL_PAYMENT', amount: 100}]
  }
};

try {
  const { data } = await axios.request(options);
  console.log(data);
} catch (error) {
  console.error(error);
}
```



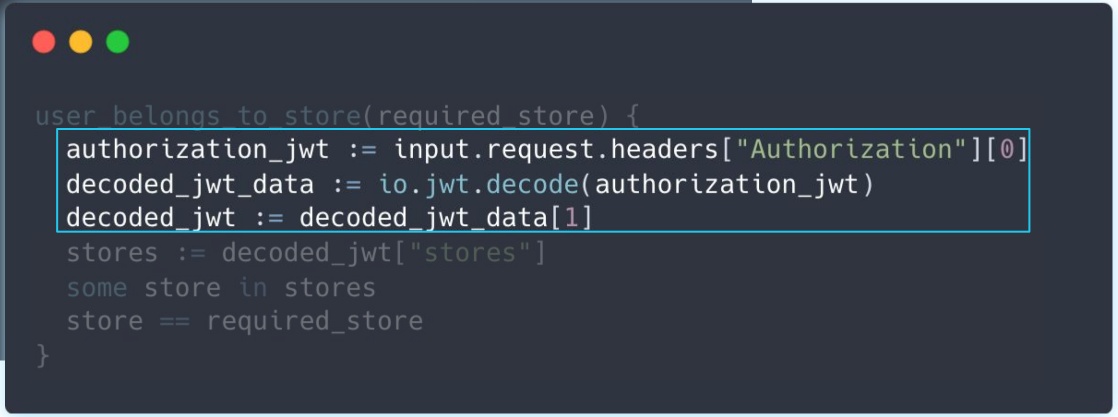
```
user_belongs_to_store(required_store) {
  authorization_jwt := input.request.headers["Authorization"][0]
  decoded_jwt_data := io.jwt.decode(authorization_jwt)
  decoded_jwt := decoded_jwt_data[1]
  stores := decoded_jwt["stores"]
  some store in stores
  store == required_store
}
```

Define a Rego function to check if the user is assigned to the store which the request insists on.

Define a Rego complex policy



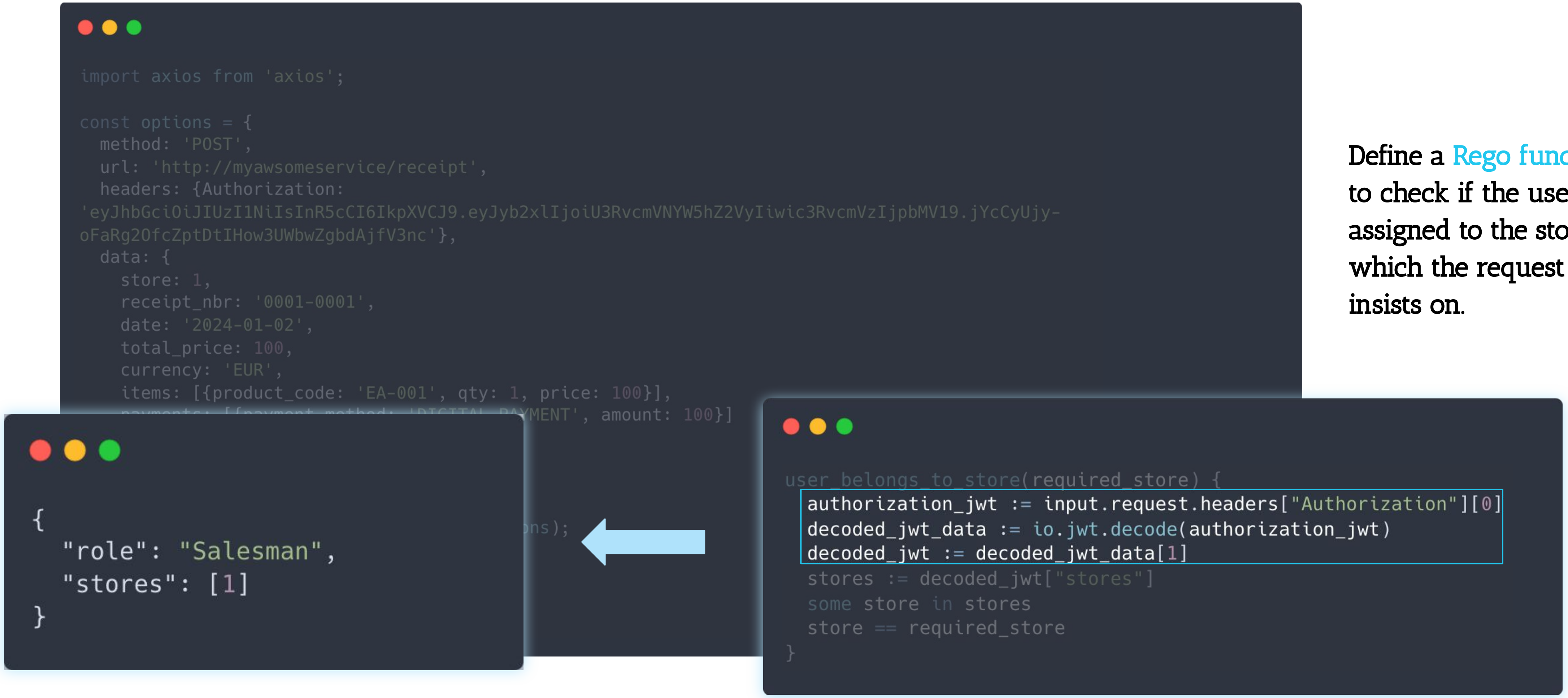
```
import axios from 'axios';

const options = {
  method: 'POST',
  url: 'http://myawsomeservice/receipt',
  headers: {Authorization:
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlIjoiU3RvcmVNYW5hZ2VyIiwic3RvcmVzIjpbMV19.jYcCyUjy-
oFaRg2OfcZptDtIHow3UWbwZgbdAjfV3nc'},
  data: {
    store: 1,
    receipt_nbr: '0001-0001',
    date: '2024-01-02',
    total_price: 100,
    currency: 'EUR',
    items: [{product_code: 'EA-001', qty: 1, price: 100}],
    payments: [{payment_method: 'DIGITAL_PAYMENT', amount: 100}]
  }
};

try {
  const { data } = await axios.request(options);
  console.log(data);
} catch (error) {
  console.error(error);
}
```



```
user_belongs_to_store(required_store) {
  authorization_jwt := input.request.headers["Authorization"][0]
  decoded_jwt_data := io.jwt.decode(authorization_jwt)
  decoded_jwt := decoded_jwt_data[1]
  stores := decoded_jwt["stores"]
  some store in stores
  store == required_store
}
```

Define a Rego function to check if the user is assigned to the store which the request insists on.

Define a Rego complex policy



```
import axios from 'axios';

const options = {
  method: 'POST',
  url: 'http://myawsomeservice/receipt',
  headers: {Authorization:
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlIjoiU3RvcmVNYW5hZ2VyIiwic3RvcmVzIjpbMV19.jYcCyUjy-
oFaRg2OfcZptDtIHow3UWbwZgbdAjfV3nc'},
  data: {
    store: 1,
    receipt_nbr: '0001-0001',
    date: '2024-01-02',
    total_price: 100,
    currency: 'EUR',
    items: [{product_code: 'EA-001', qty: 1, price: 100}],
    payments: [{payment_method: 'DIGITAL_PAYMENT', amount: 100}]
```

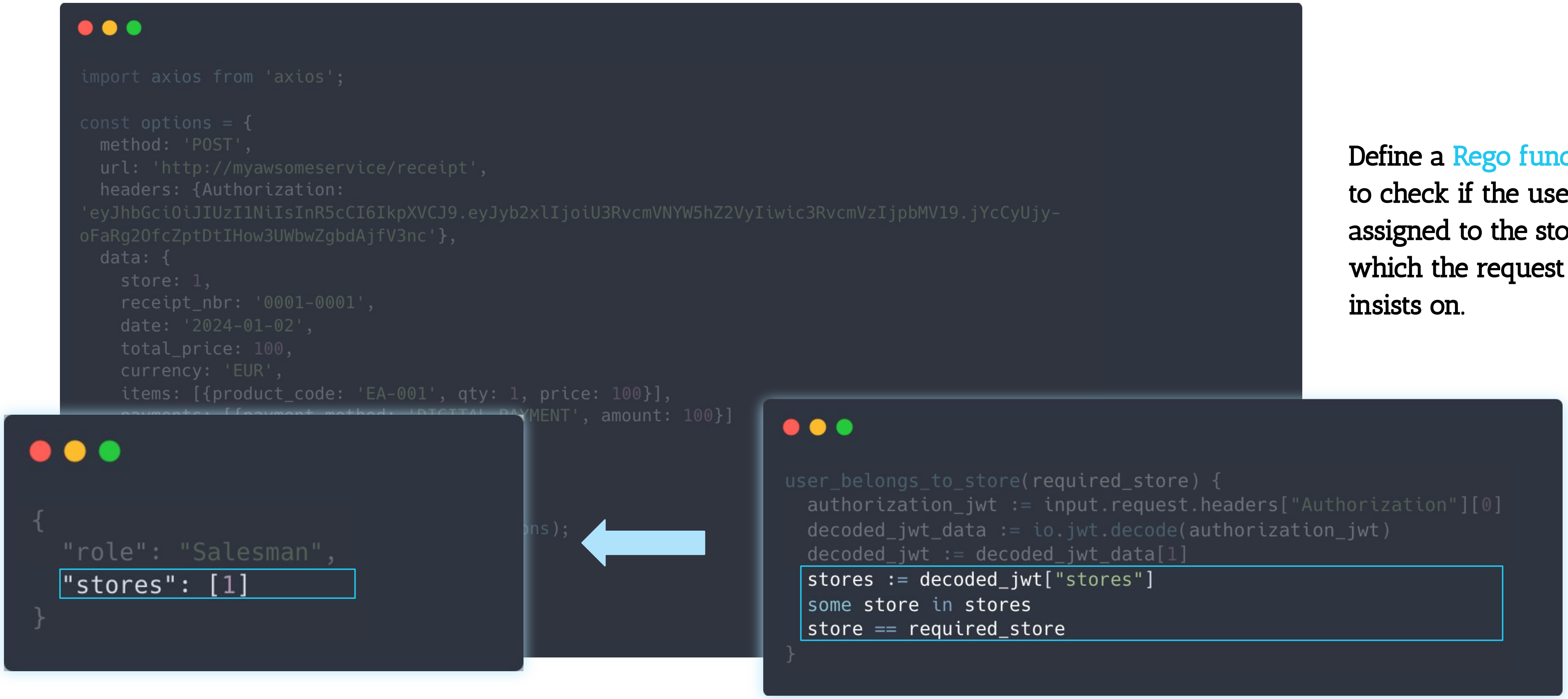
```
ons);
```

```
{
  "role": "Salesman",
  "stores": [1]
}
```

```
user_belongs_to_store(required_store) {
  authorization_jwt := input.request.headers["Authorization"][0]
  decoded_jwt_data := io.jwt.decode(authorization_jwt)
  decoded_jwt := decoded_jwt_data[1]
  stores := decoded_jwt["stores"]
  some store in stores
  store == required_store
}
```

Define a Rego function to check if the user is assigned to the store which the request insists on.

Define a Rego complex policy



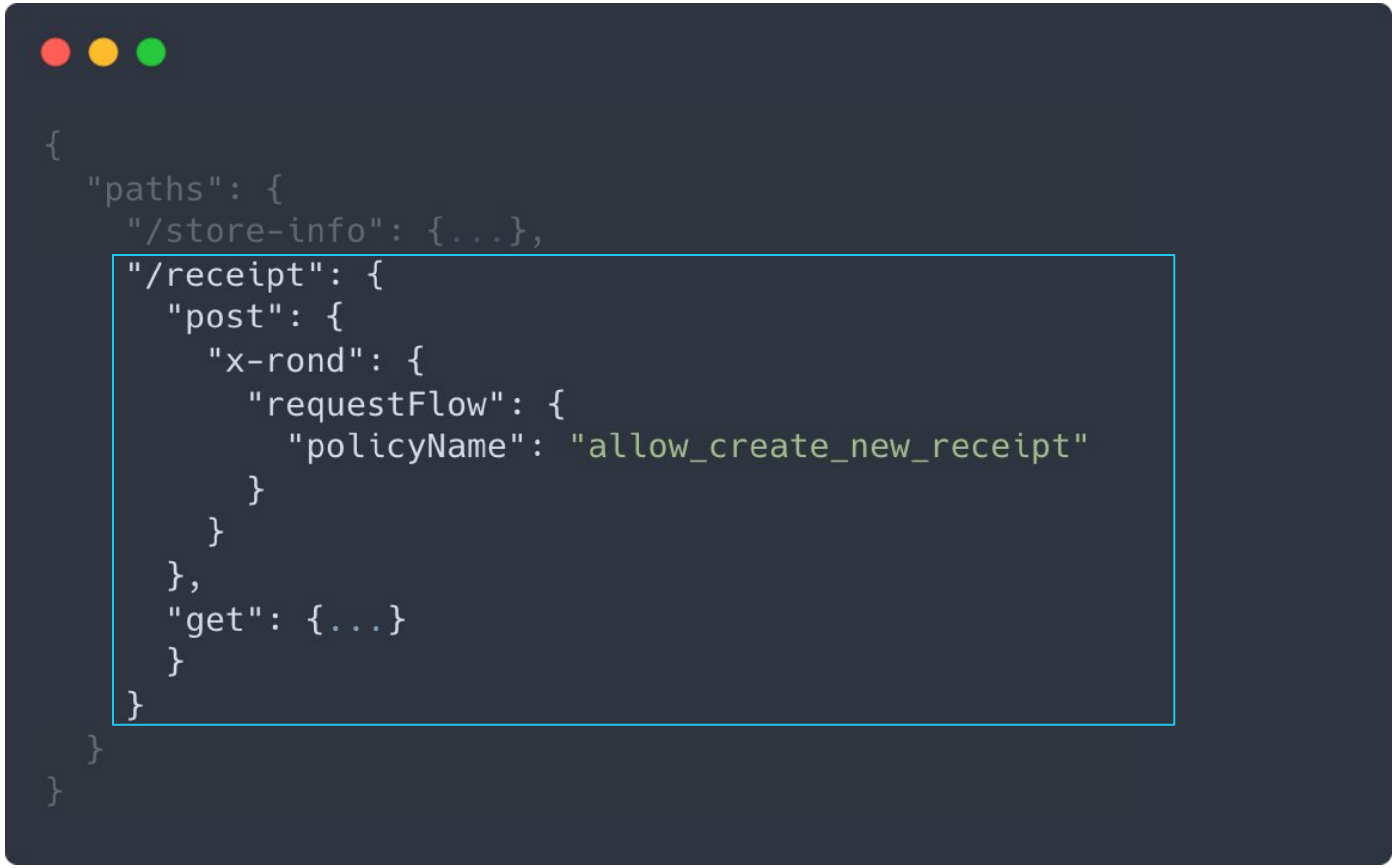
```
import axios from 'axios';

const options = {
  method: 'POST',
  url: 'http://myawsomeservice/receipt',
  headers: {Authorization:
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlIjoiU3RvcmVNYW5hZ2VyIiwic3RvcmVzIjpbMV19.jYcCyUjy-
oFaRg2OfcZptDtIHow3UWbwZgbdAjfV3nc'},
  data: {
    store: 1,
    receipt_nbr: '0001-0001',
    date: '2024-01-02',
    total_price: 100,
    currency: 'EUR',
    items: [{product_code: 'EA-001', qty: 1, price: 100}],
    payments: [{payment_method: 'DIGITAL_PAYMENT', amount: 100}]

ns);
```

```
{
  "role": "Salesman",
  "stores": [1]
}
```

```
user_belongs_to_store(required_store) {
  authorization_jwt := input.request.headers["Authorization"][0]
  decoded_jwt_data := io.jwt.decode(authorization_jwt)
  decoded_jwt := decoded_jwt_data[1]
  stores := decoded_jwt["stores"]
  some store in stores
  store == required_store
}
```

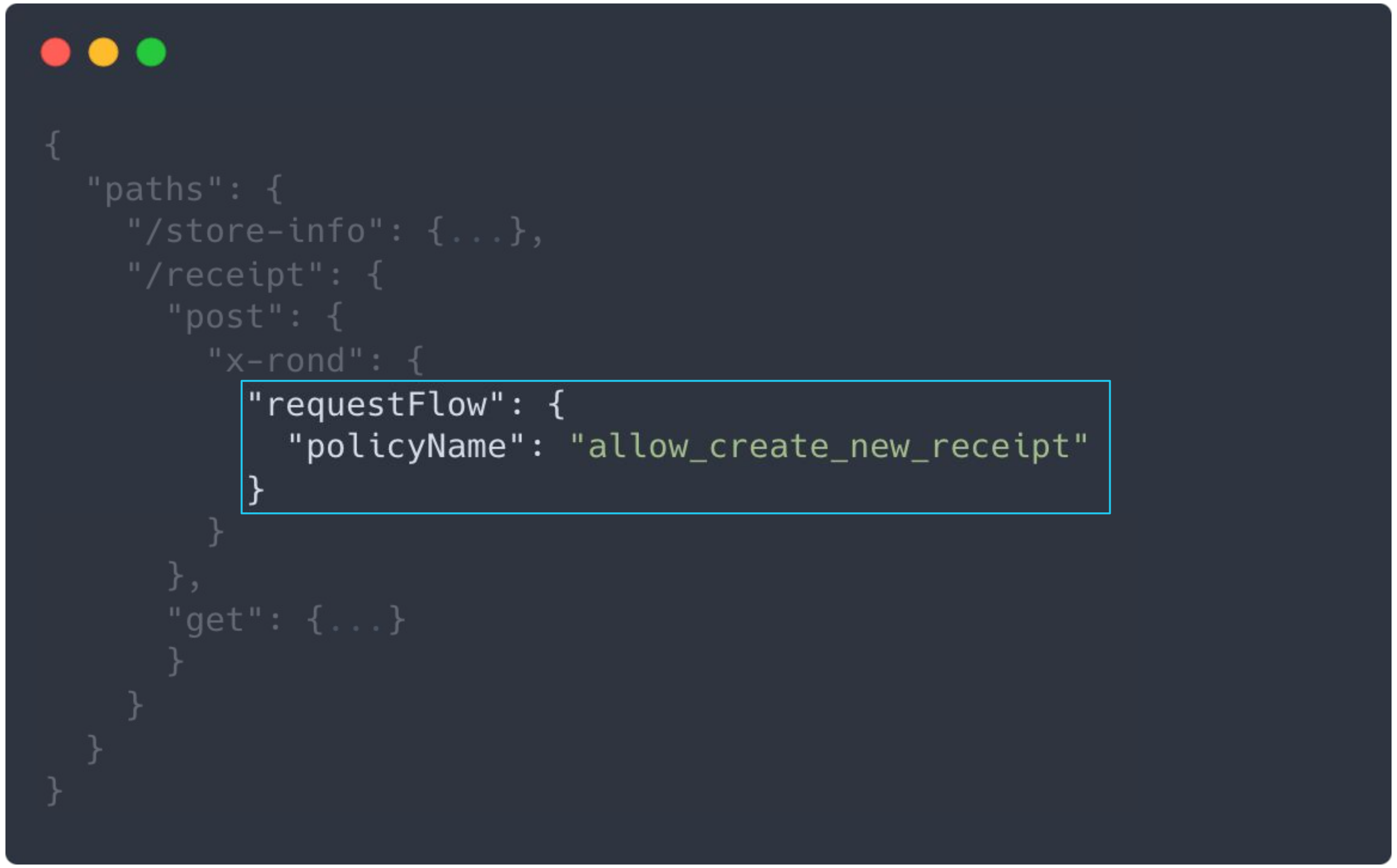
Define a Rego function to check if the user is assigned to the store which the request insists on.

Define a Rego complex policy



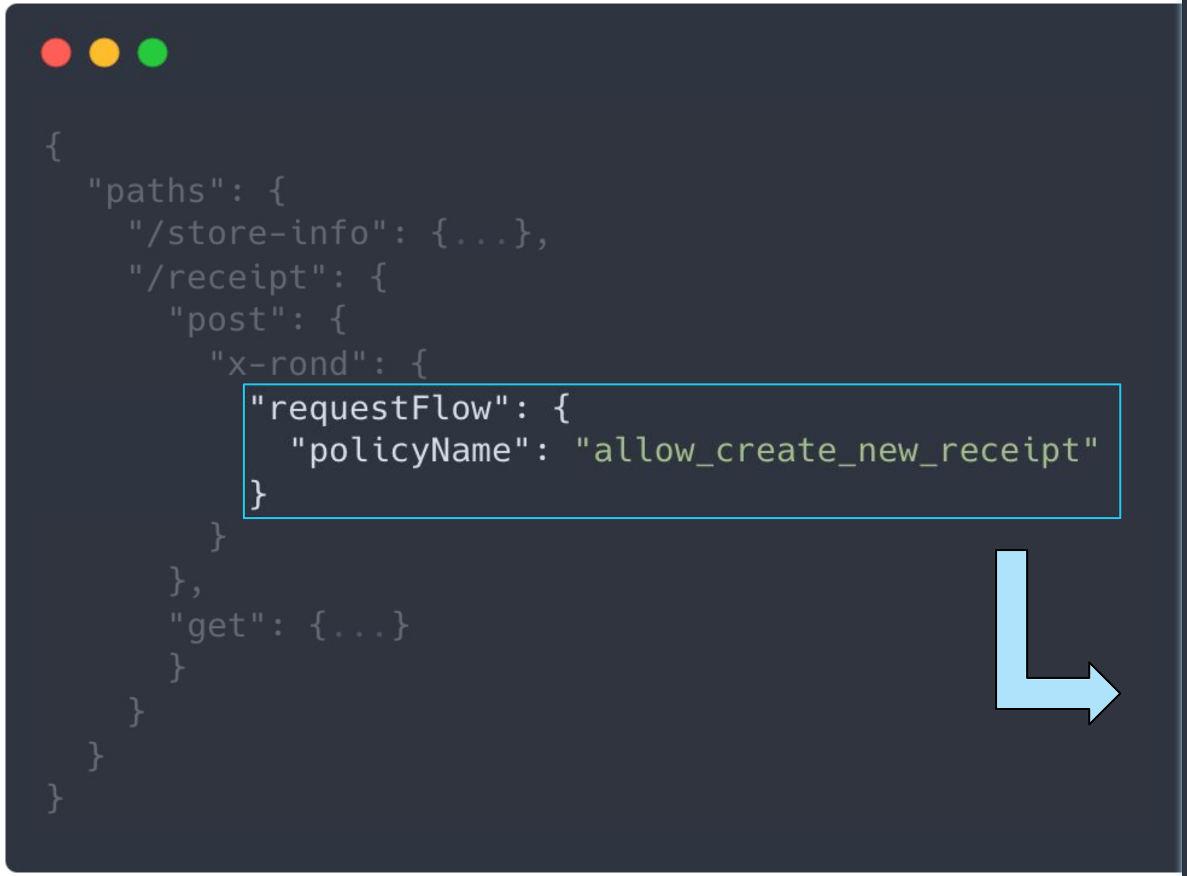
```
{
  "paths": {
    "/store-info": {...},
    "/receipt": {
      "post": {
        "x-rond": {
          "requestFlow": {
            "policyName": "allow_create_new_receipt"
          }
        }
      },
      "get": {...}
    }
  }
}
```

Define a Rego complex policy

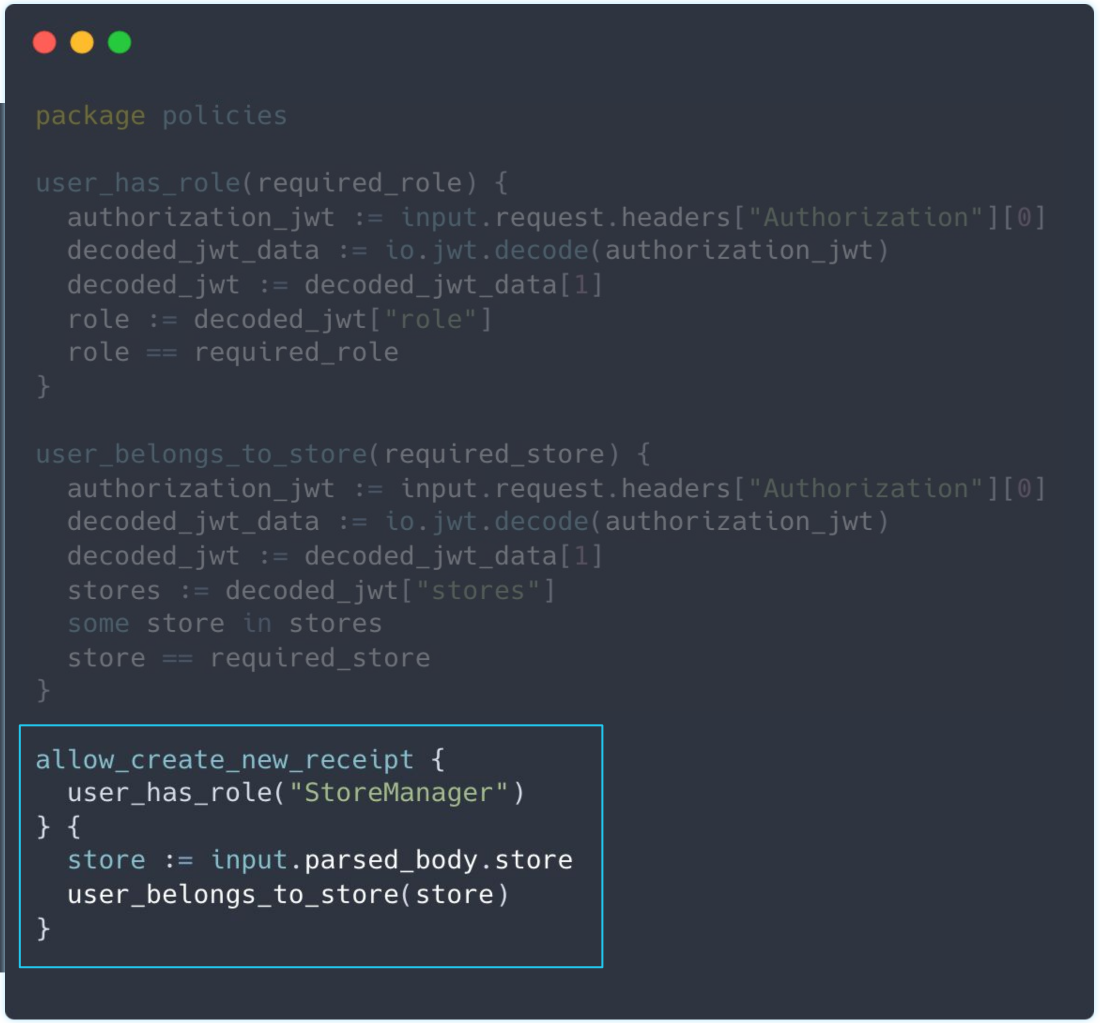


```
{
  "paths": {
    "/store-info": {...},
    "/receipt": {
      "post": {
        "x-rond": {
          "requestFlow": {
            "policyName": "allow_create_new_receipt"
          }
        }
      },
      "get": {...}
      }
    }
  }
}
```

Define a Rego complex policy

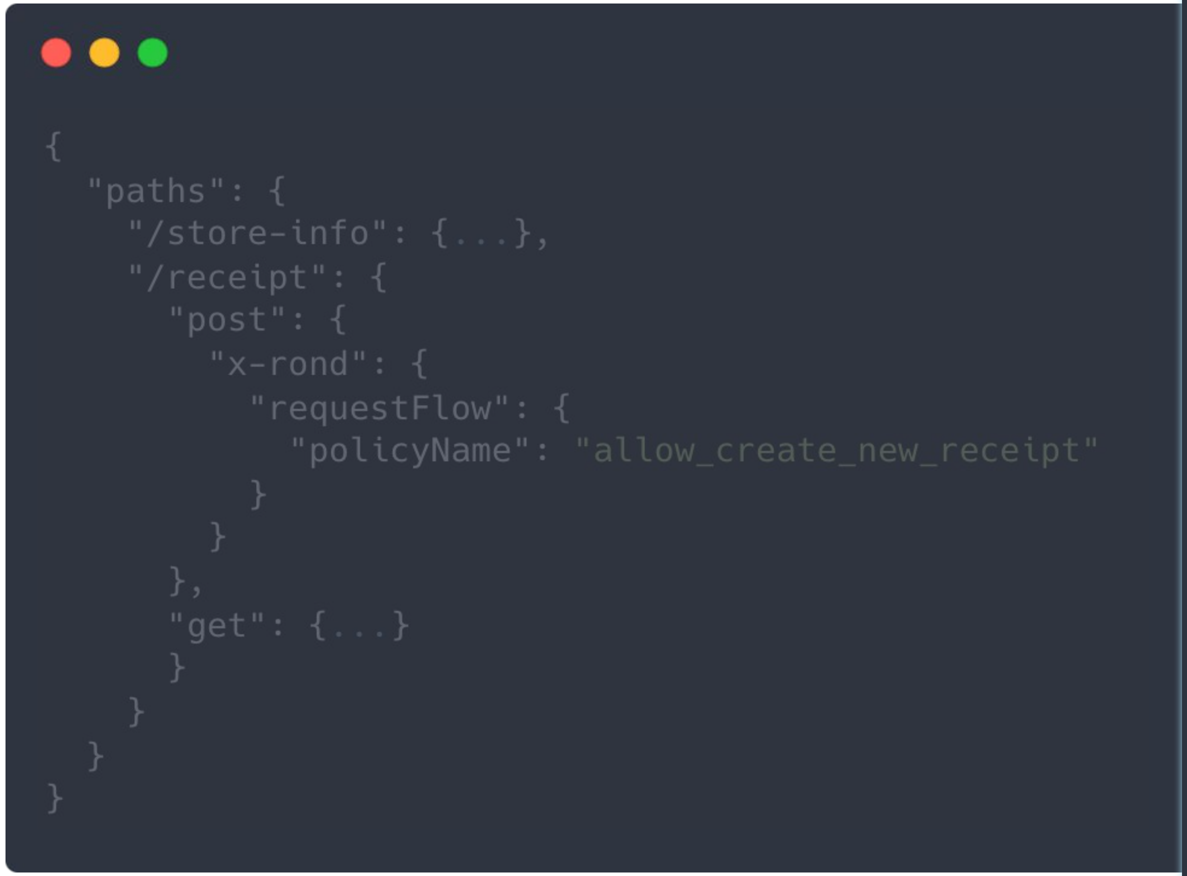


```
{
  "paths": {
    "/store-info": {...},
    "/receipt": {
      "post": {
        "x-rond": {
          "requestFlow": {
            "policyName": "allow_create_new_receipt"
          }
        }
      },
      "get": {...}
      }
    }
  }
}
```

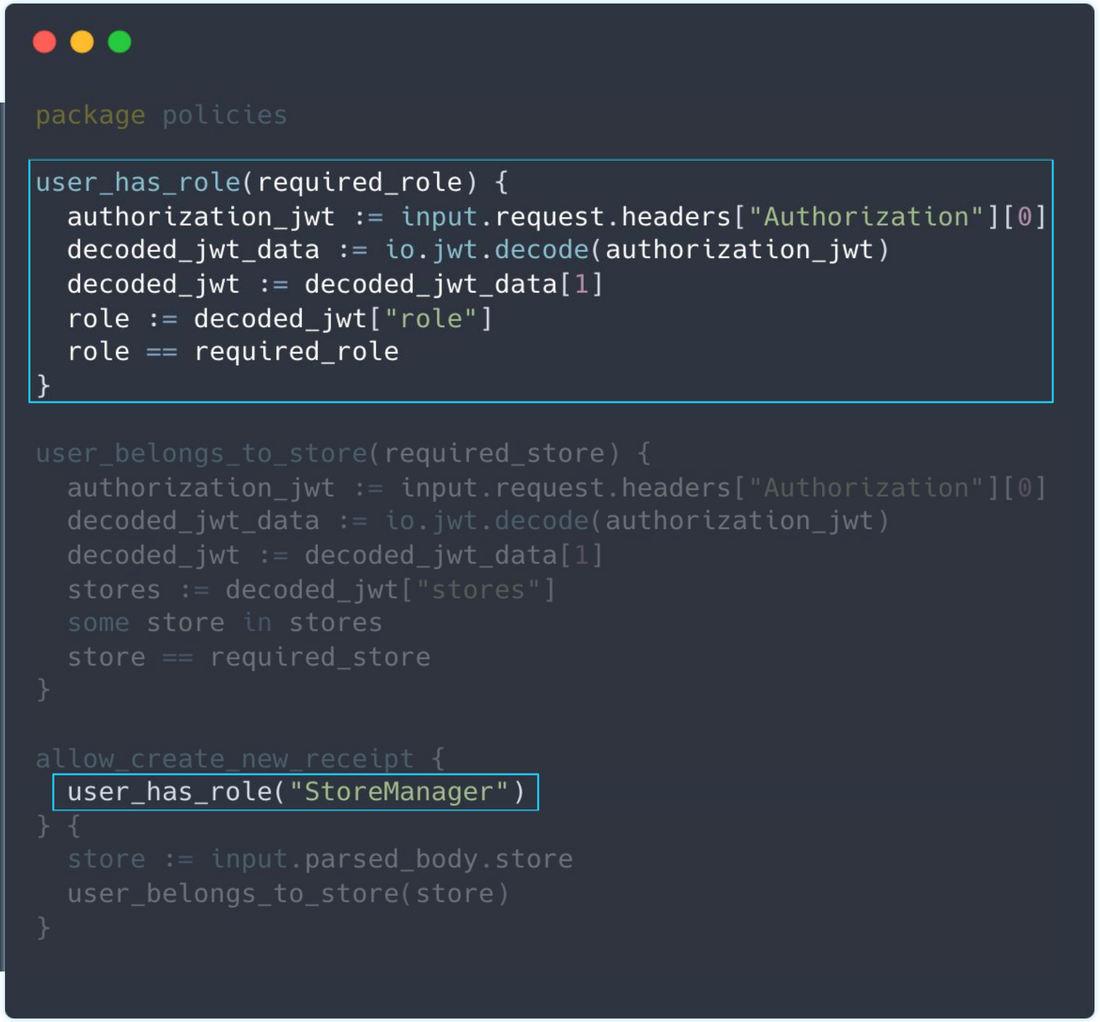


```
package policies

user_has_role(required_role) {
  authorization_jwt := input.request.headers["Authorization"][0]
  decoded_jwt_data := io.jwt.decode(authorization_jwt)
  decoded_jwt := decoded_jwt_data[1]
  role := decoded_jwt["role"]
  role == required_role
}

user_belongs_to_store(required_store) {
  authorization_jwt := input.request.headers["Authorization"][0]
  decoded_jwt_data := io.jwt.decode(authorization_jwt)
  decoded_jwt := decoded_jwt_data[1]
  stores := decoded_jwt["stores"]
  some store in stores
  store == required_store
}

allow_create_new_receipt {
  user_has_role("StoreManager")
} {
  store := input.parsed_body.store
  user_belongs_to_store(store)
}
```

Define a Rego complex policy

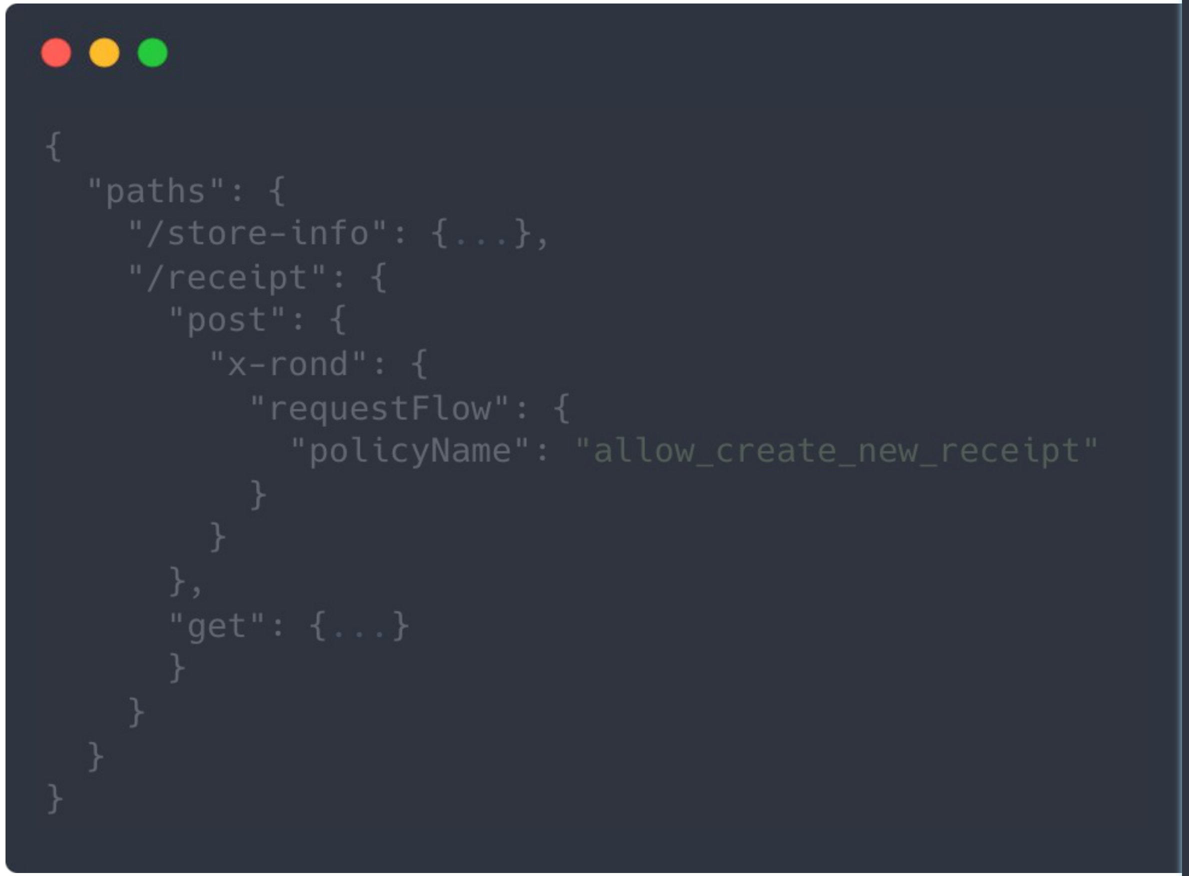


```
{
  "paths": {
    "/store-info": {...},
    "/receipt": {
      "post": {
        "x-rond": {
          "requestFlow": {
            "policyName": "allow_create_new_receipt"
          }
        }
      },
      "get": {...}
    }
  }
}
```

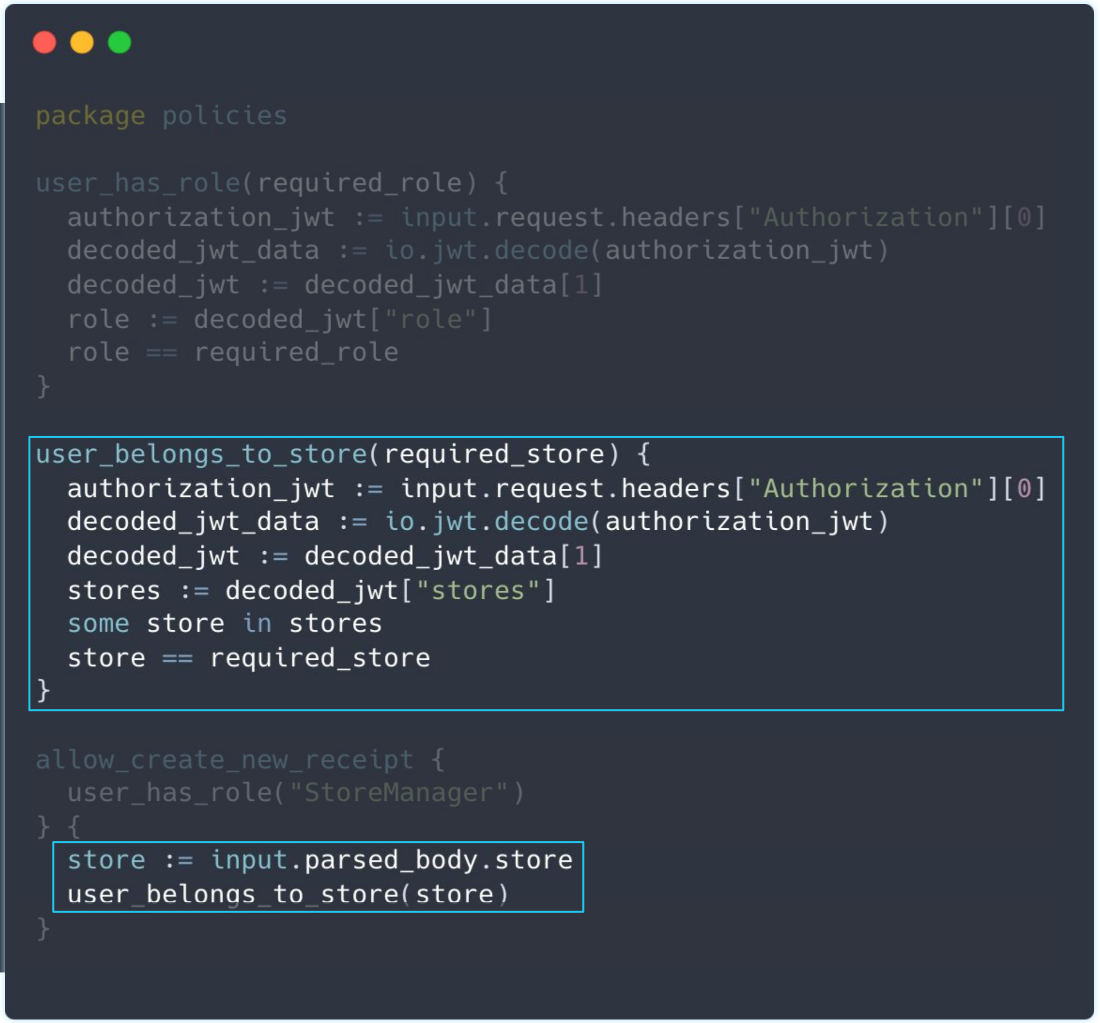


```
package policies

user_has_role(required_role) {
  authorization_jwt := input.request.headers["Authorization"][0]
  decoded_jwt_data := io.jwt.decode(authorization_jwt)
  decoded_jwt := decoded_jwt_data[1]
  role := decoded_jwt["role"]
  role == required_role
}

user_belongs_to_store(required_store) {
  authorization_jwt := input.request.headers["Authorization"][0]
  decoded_jwt_data := io.jwt.decode(authorization_jwt)
  decoded_jwt := decoded_jwt_data[1]
  stores := decoded_jwt["stores"]
  some store in stores
  store == required_store
}

allow_create_new_receipt {
  user_has_role("StoreManager")
} {
  store := input.parsed_body.store
  user_belongs_to_store(store)
}
```

Define a Rego complex policy



```
{
  "paths": {
    "/store-info": {...},
    "/receipt": {
      "post": {
        "x-rond": {
          "requestFlow": {
            "policyName": "allow_create_new_receipt"
          }
        }
      },
      "get": {...}
    }
  }
}
```



```
package policies

user_has_role(required_role) {
  authorization_jwt := input.request.headers["Authorization"][0]
  decoded_jwt_data := io.jwt.decode(authorization_jwt)
  decoded_jwt := decoded_jwt_data[1]
  role := decoded_jwt["role"]
  role == required_role
}

user_belongs_to_store(required_store) {
  authorization_jwt := input.request.headers["Authorization"][0]
  decoded_jwt_data := io.jwt.decode(authorization_jwt)
  decoded_jwt := decoded_jwt_data[1]
  stores := decoded_jwt["stores"]
  some store in stores
  store == required_store
}

allow_create_new_receipt {
  user_has_role("StoreManager")
} {
  store := input.parsed_body.store
  user_belongs_to_store(store)
}
```

The APIs to secure

→ GET /store-info

All requests are accepted even if without an Authorization header.

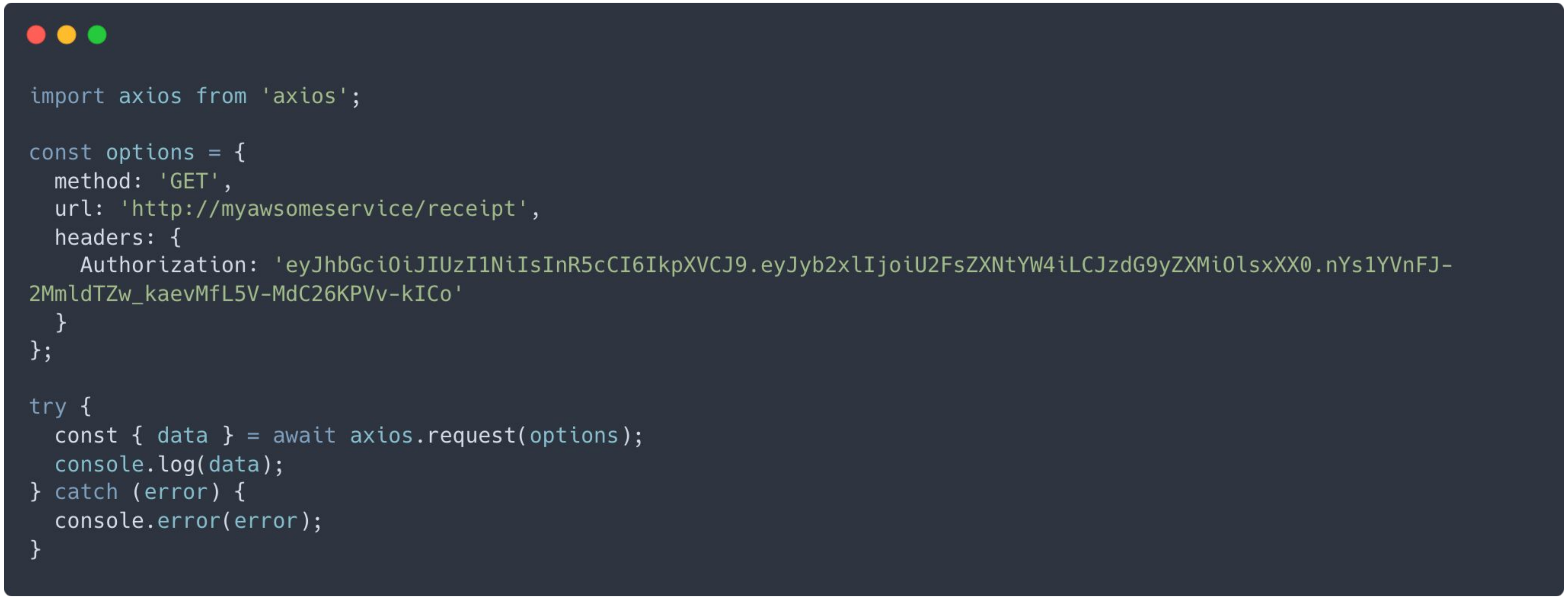
→ POST /receipt

Only authenticated requests are accepted.
Users with role *Salesman* can add only receipts for their store, *StoreManager* can add receipts on the entire chain.

→ GET /receipt

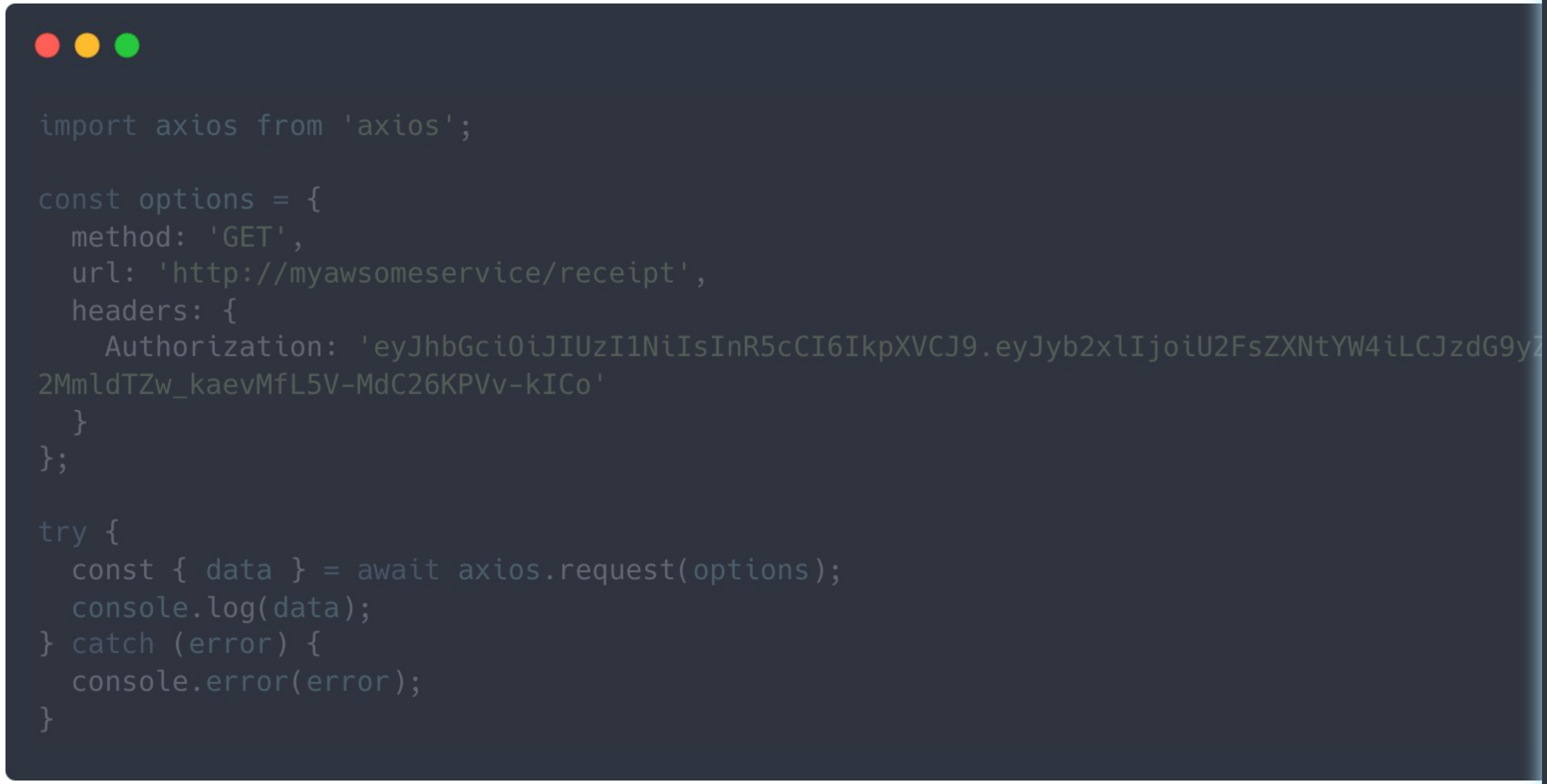
Only authenticated requests are accepted.
Users with role *Salesman* can access the receipts without the payments and customer sensitive data, *StoreManager* can access the receipts also with payment and customer data..

Modify response with Rönd

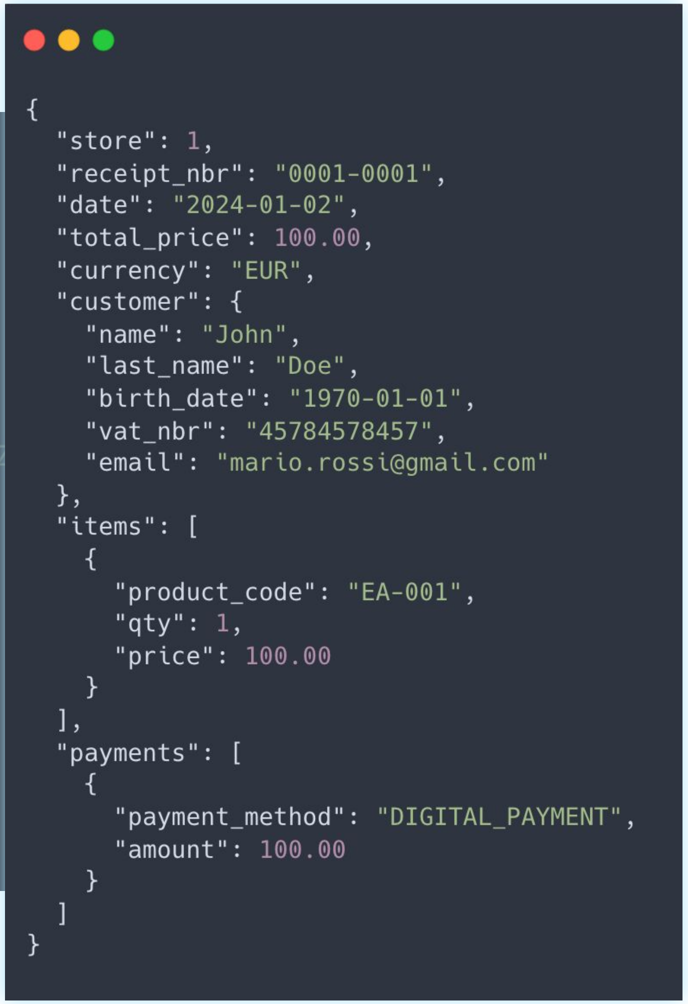


```
import axios from 'axios';

const options = {
  method: 'GET',
  url: 'http://myawsomeservice/receipt',
  headers: {
    Authorization: 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlIjoiU2FsZXNtYW4iLCJzdG9yZXMiOlsxXX0.nYs1YVnFJ-
2MmldTZw_kaevMfL5V-MdC26KPVv-kICo'
  }
};

try {
  const { data } = await axios.request(options);
  console.log(data);
} catch (error) {
  console.error(error);
}
```

Modify response with Rönd

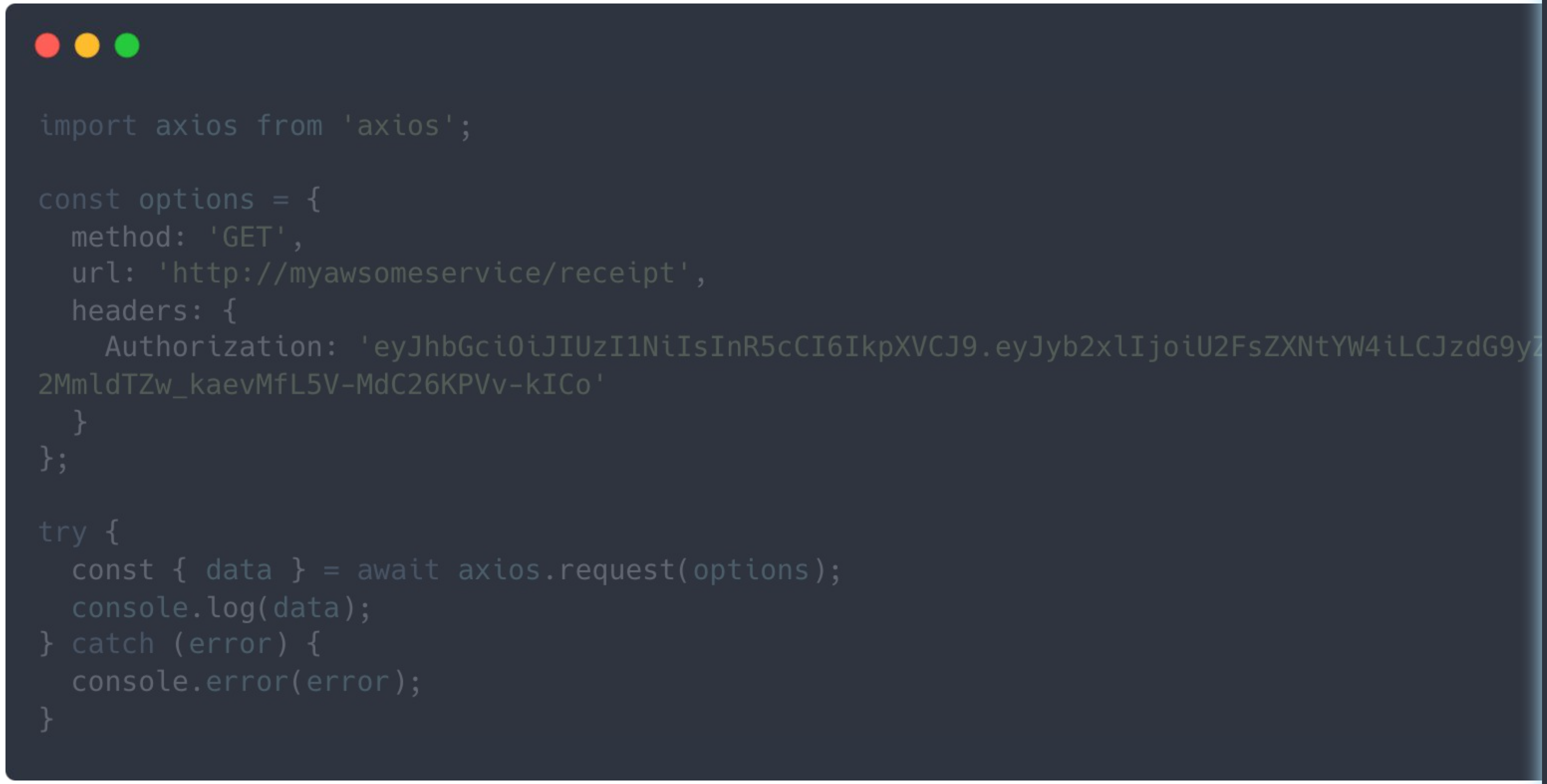


```
import axios from 'axios';

const options = {
  method: 'GET',
  url: 'http://myawsomeservice/receipt',
  headers: {
    Authorization: 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlIjoiU2FsZXNtYW4iLCJzdG9yZ
2MmldTZw_kaevMfL5V-MdC26KPVv-kICo'
  }
};

try {
  const { data } = await axios.request(options);
  console.log(data);
} catch (error) {
  console.error(error);
}
```

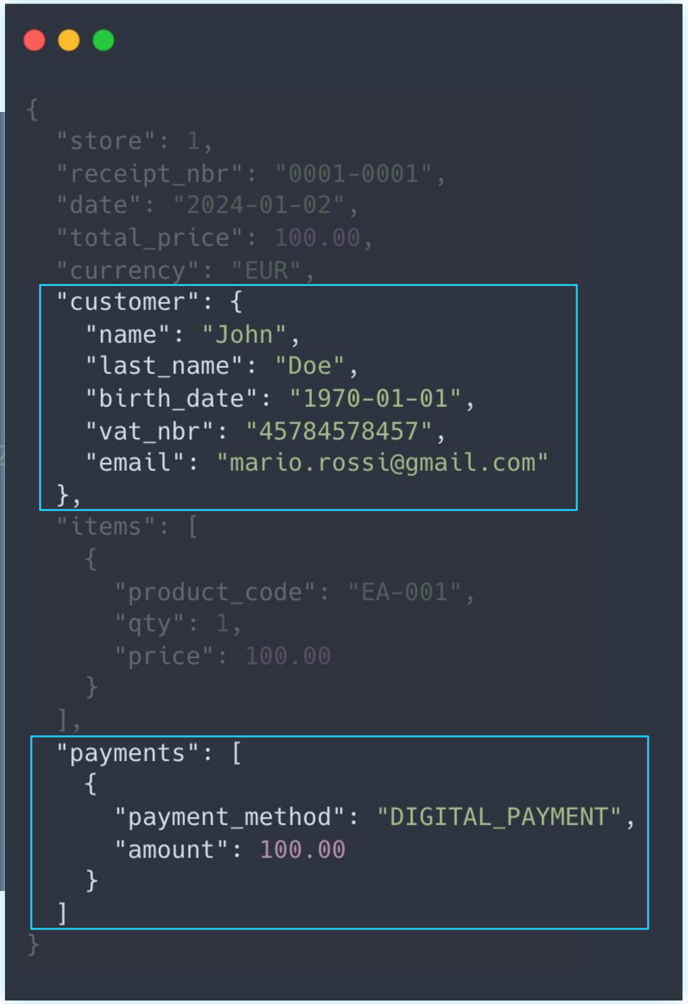


```
{
  "store": 1,
  "receipt_nbr": "0001-0001",
  "date": "2024-01-02",
  "total_price": 100.00,
  "currency": "EUR",
  "customer": {
    "name": "John",
    "last_name": "Doe",
    "birth_date": "1970-01-01",
    "vat_nbr": "45784578457",
    "email": "mario.rossi@gmail.com"
  },
  "items": [
    {
      "product_code": "EA-001",
      "qty": 1,
      "price": 100.00
    }
  ],
  "payments": [
    {
      "payment_method": "DIGITAL_PAYMENT",
      "amount": 100.00
    }
  ]
}
```

Modify response with Rönd



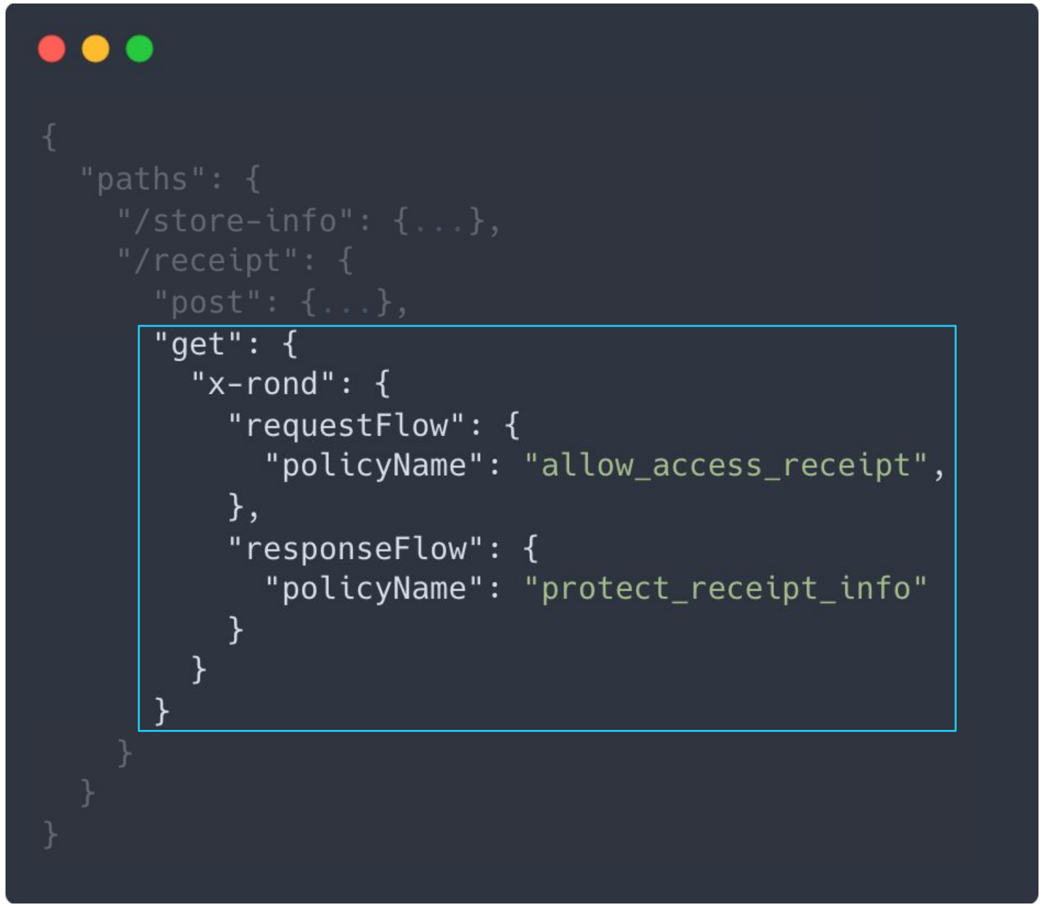
```
import axios from 'axios';

const options = {
  method: 'GET',
  url: 'http://myawsomeservice/receipt',
  headers: {
    Authorization: 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlIjoiU2FsZXNtYW4iLCJzdG9y
2MmldTZw_kaevMfL5V-MdC26KPVv-kICo'
  }
};

try {
  const { data } = await axios.request(options);
  console.log(data);
} catch (error) {
  console.error(error);
}
```



```
{
  "store": 1,
  "receipt_nbr": "0001-0001",
  "date": "2024-01-02",
  "total_price": 100.00,
  "currency": "EUR",
  "customer": {
    "name": "John",
    "last_name": "Doe",
    "birth_date": "1970-01-01",
    "vat_nbr": "45784578457",
    "email": "mario.rossi@gmail.com"
  },
  "items": [
    {
      "product_code": "EA-001",
      "qty": 1,
      "price": 100.00
    }
  ],
  "payments": [
    {
      "payment_method": "DIGITAL_PAYMENT",
      "amount": 100.00
    }
  ]
}
```

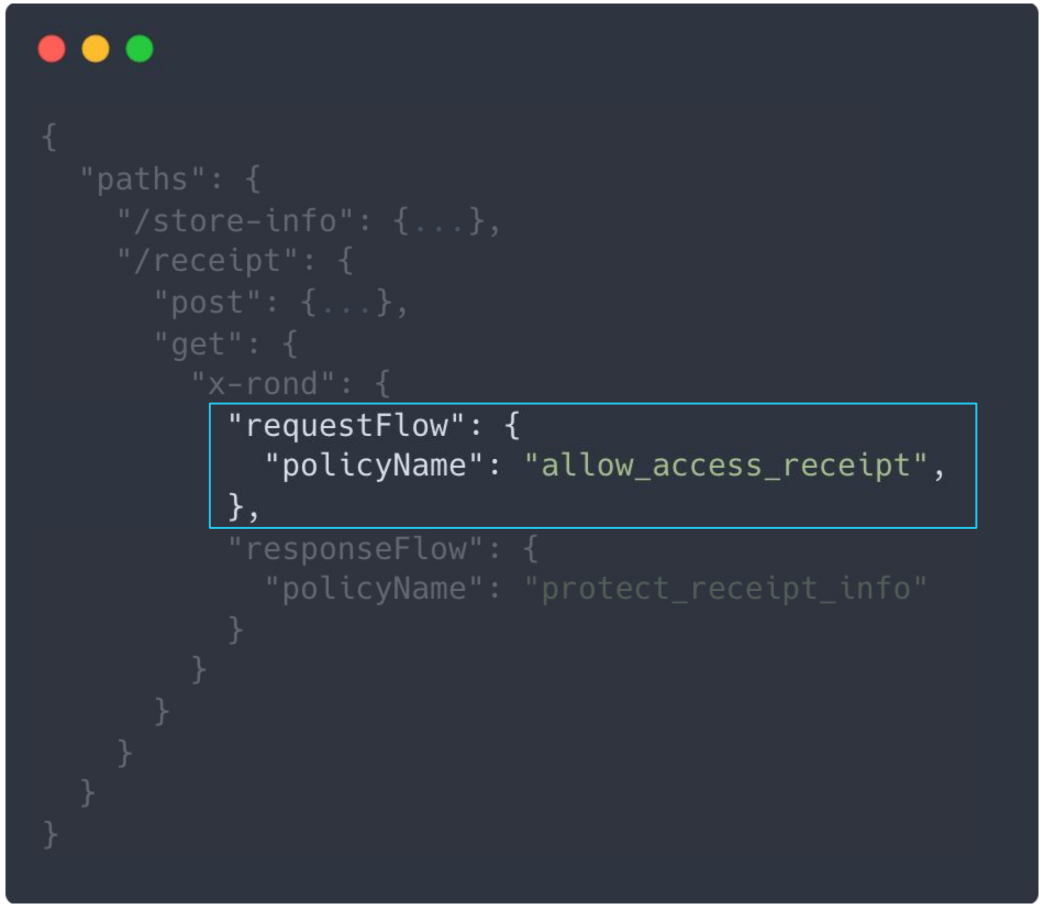
We have to remove *customer* and *payments* information form the response for *Salesman* users.

Modify response with Rönd



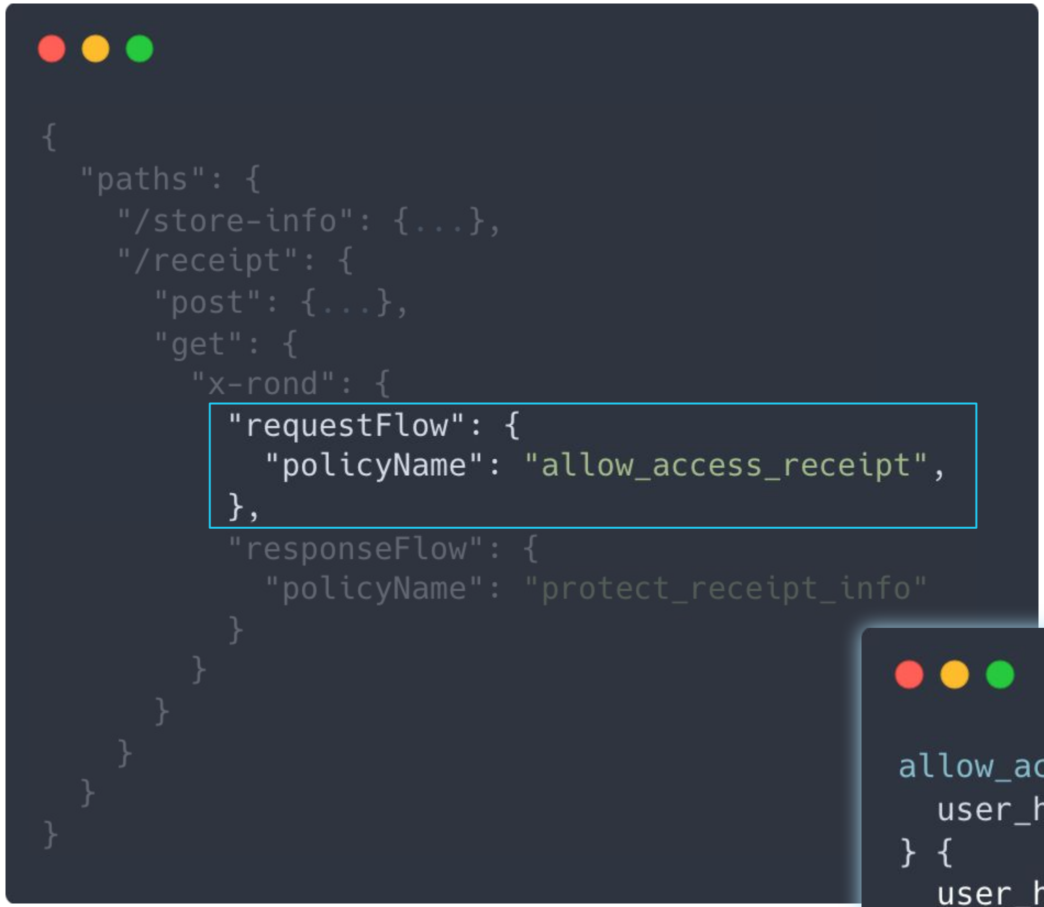
```
{
  "paths": {
    "/store-info": {...},
    "/receipt": {
      "post": {...},
      "get": {
        "x-rond": {
          "requestFlow": {
            "policyName": "allow_access_receipt",
          },
          "responseFlow": {
            "policyName": "protect_receipt_info"
          }
        }
      }
    }
  }
}
```

Modify response with Rönd



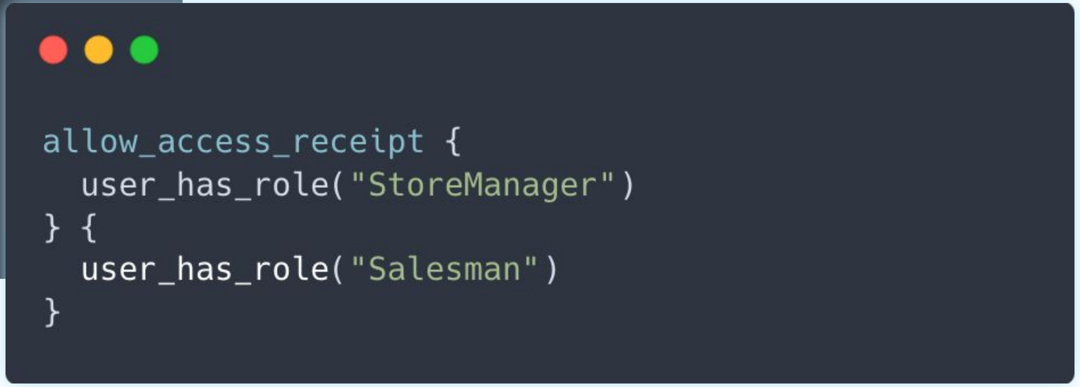
```
{
  "paths": {
    "/store-info": {...},
    "/receipt": {
      "post": {...},
      "get": {
        "x-rond": {
          "requestFlow": {
            "policyName": "allow_access_receipt",
          },
          "responseFlow": {
            "policyName": "protect_receipt_info"
          }
        }
      }
    }
  }
}
```

Modify response with Rönd



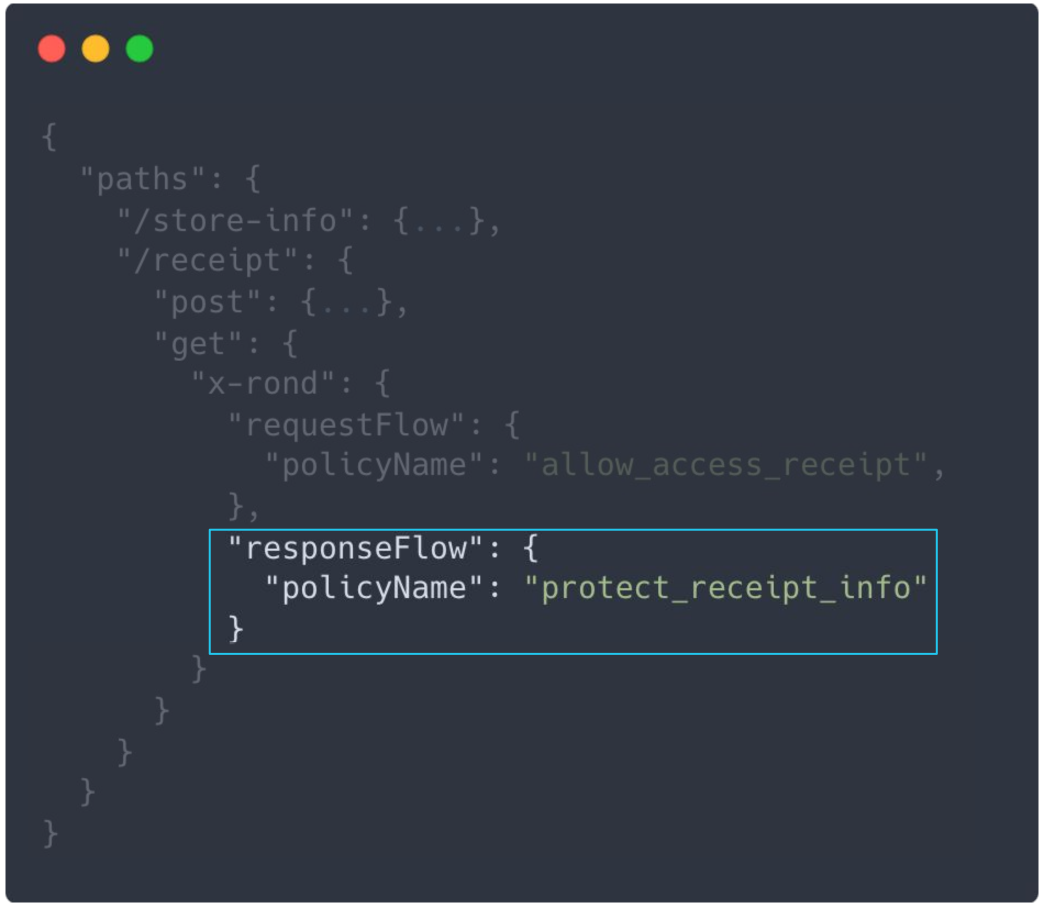
```
{
  "paths": {
    "/store-info": {...},
    "/receipt": {
      "post": {...},
      "get": {
        "x-rond": {
          "requestFlow": {
            "policyName": "allow_access_receipt",
          },
          "responseFlow": {
            "policyName": "protect_receipt_info"
          }
        }
      }
    }
  }
}
```

First of all check that the request is authenticated and that the user belongs to *StoreManager* of *Salesman* role.



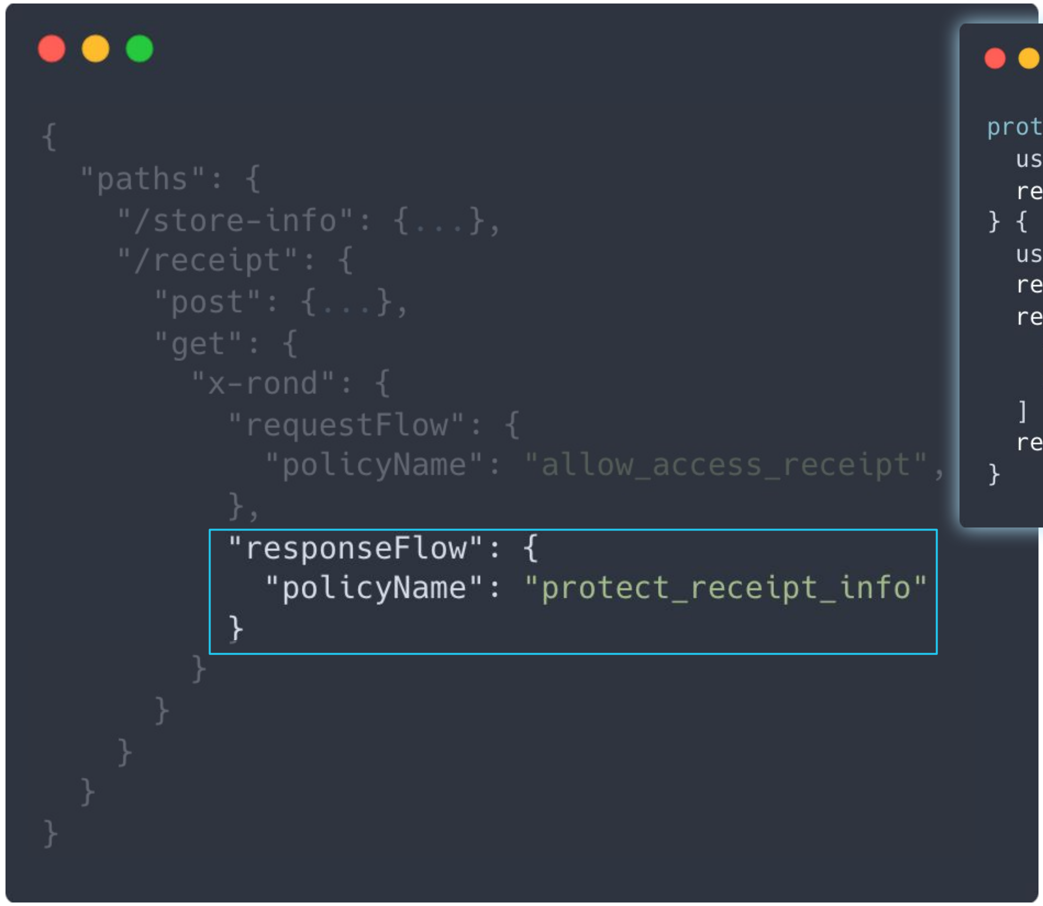
```
allow_access_receipt {
  user_has_role("StoreManager")
} {
  user_has_role("Salesman")
}
```

Modify response with Rönd

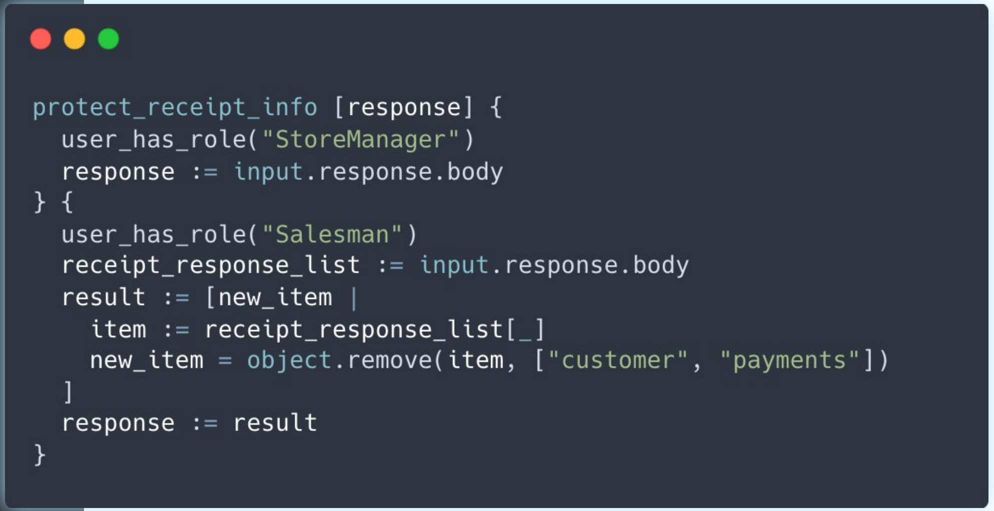


```
{
  "paths": {
    "/store-info": {...},
    "/receipt": {
      "post": {...},
      "get": {
        "x-rond": {
          "requestFlow": {
            "policyName": "allow_access_receipt",
          },
          "responseFlow": {
            "policyName": "protect_receipt_info"
          }
        }
      }
    }
  }
}
```

Modify response with Rönd



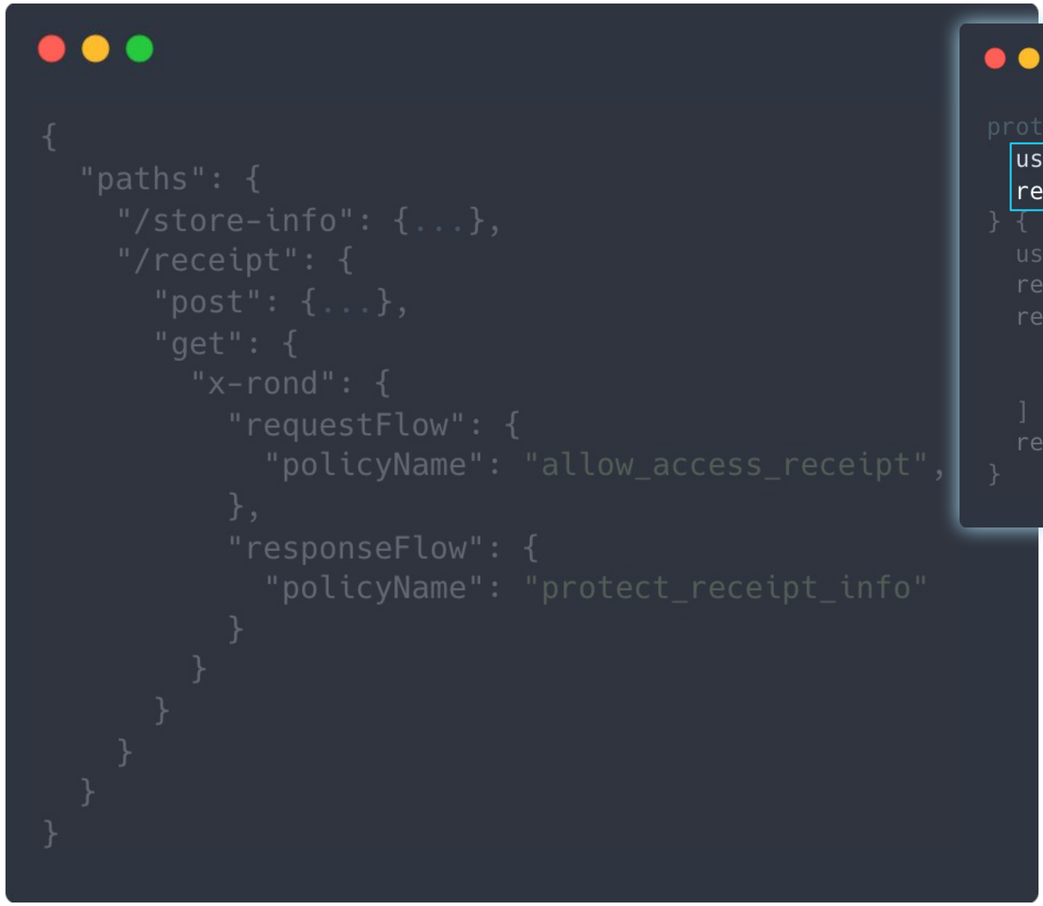
```
{
  "paths": {
    "/store-info": {...},
    "/receipt": {
      "post": {...},
      "get": {
        "x-rond": {
          "requestFlow": {
            "policyName": "allow_access_receipt",
          },
          "responseFlow": {
            "policyName": "protect_receipt_info"
          }
        }
      }
    }
  }
}
```



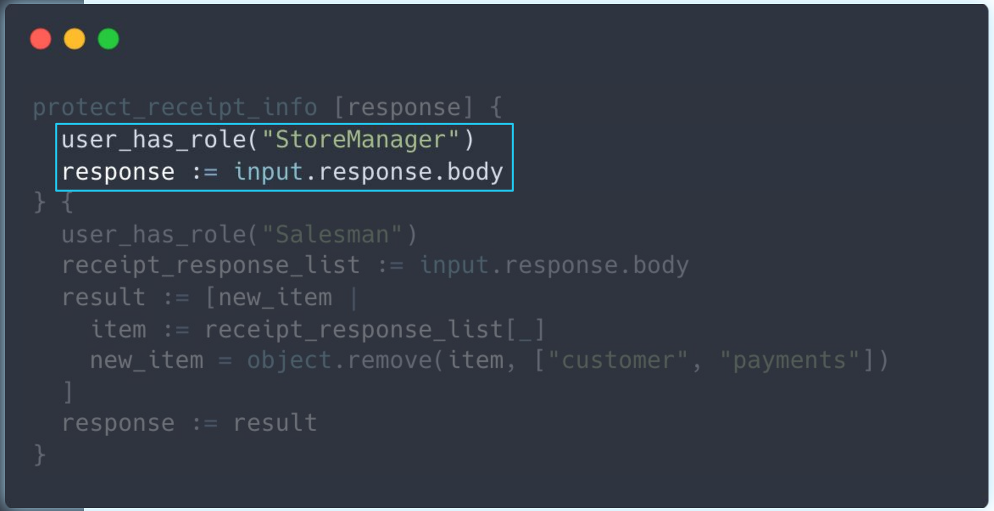
```
protect_receipt_info [response] {
  user_has_role("StoreManager")
  response := input.response.body
} {
  user_has_role("Salesman")
  receipt_response_list := input.response.body
  result := [new_item |
    item := receipt_response_list[_]
    new_item = object.remove(item, ["customer", "payments"])
  ]
  response := result
}
```

Based on the role of the user performing the request we filter the response to remove *customer* and *payments* object from the response in case the user is a *Salesman*.

Modify response with Rönd

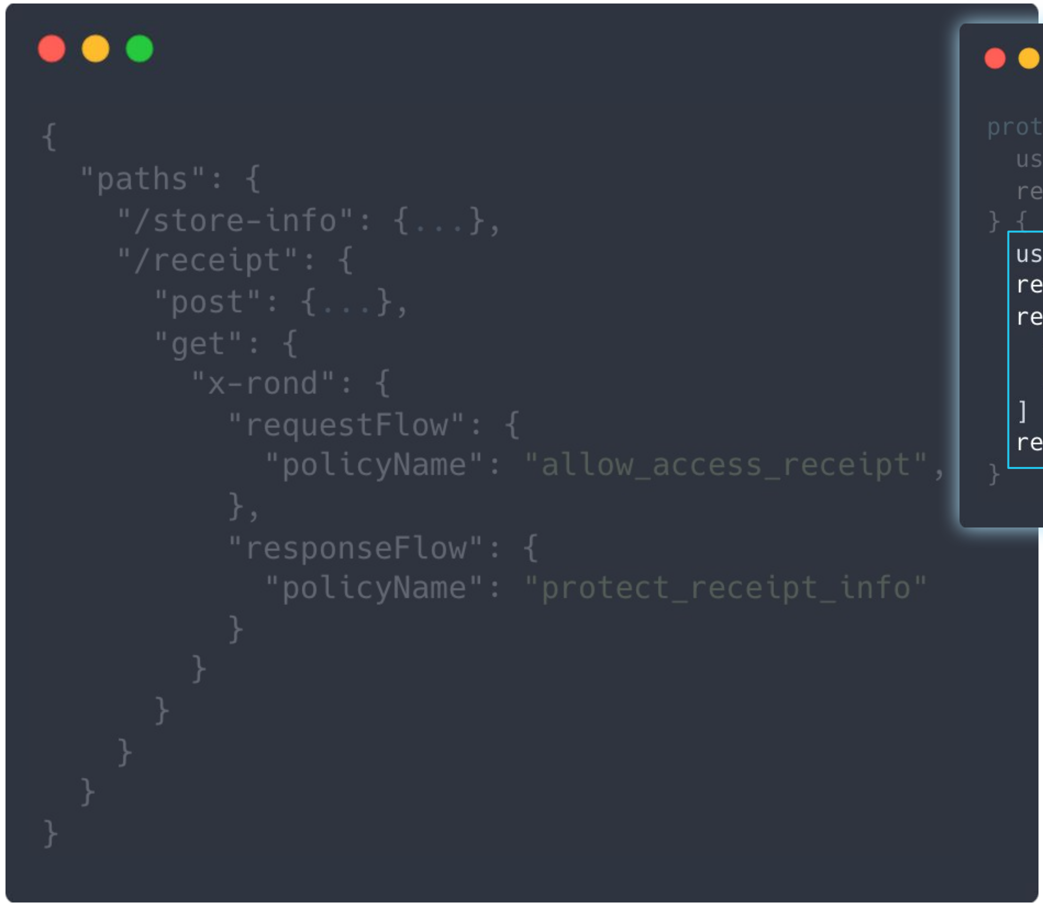


```
{
  "paths": {
    "/store-info": {...},
    "/receipt": {
      "post": {...},
      "get": {
        "x-rond": {
          "requestFlow": {
            "policyName": "allow_access_receipt",
          },
          "responseFlow": {
            "policyName": "protect_receipt_info"
          }
        }
      }
    }
  }
}
```

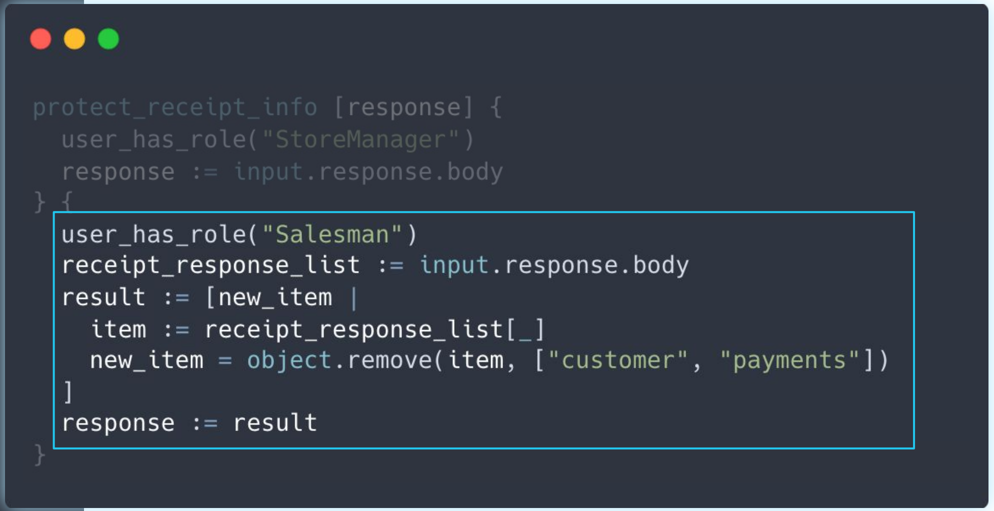


```
protect_receipt_info [response] {
  user_has_role("StoreManager")
  response := input.response.body
} {
  user_has_role("Salesman")
  receipt_response_list := input.response.body
  result := [new_item |
    item := receipt_response_list[_]
    new_item = object.remove(item, ["customer", "payments"])
  ]
  response := result
}
```

Modify response with Rönd



```
{
  "paths": {
    "/store-info": {...},
    "/receipt": {
      "post": {...},
      "get": {
        "x-rond": {
          "requestFlow": {
            "policyName": "allow_access_receipt",
          },
          "responseFlow": {
            "policyName": "protect_receipt_info"
          }
        }
      }
    }
  }
}
```



```
protect_receipt_info [response] {
  user_has_role("StoreManager")
  response := input.response.body
} {
  user_has_role("Salesman")
  receipt_response_list := input.response.body
  result := [new_item |
    item := receipt_response_list[_]
    new_item = object.remove(item, ["customer", "payments"])
  ]
  response := result
}
```

That’s all!

Key session takeaways

01 **PaC: your swiss knife for policy management**

That's all!

Key session takeaways

01 **PaC: your swiss knife for policy management**

02 **Open Policy Agent: the tool for policy decoupling**

That’s all!

Key session takeaways

01 **PaC: your swiss knife for policy management**

02 **Open Policy Agent: the tool for policy decoupling**

03 **Rönd: implement your RBAC architecture with ease**

Q&A Time

Feel free to ask any question now or reach me around the conference to have a chat!

Leave your feedback here! →

https://bit.ly/feedback-graz

Grazie!

Graziano Casto

graziano.casto@mia-platform.eu
@castograziano
castograziano.com

https://bit.ly/feedback-graz