DOCUMENTATION OF IUDX PROJECT

IUDX Consent Management (SRIP23-005)

FRONTEND

Intern Name: Vinti Dadheech

WEBSITE DESIGN

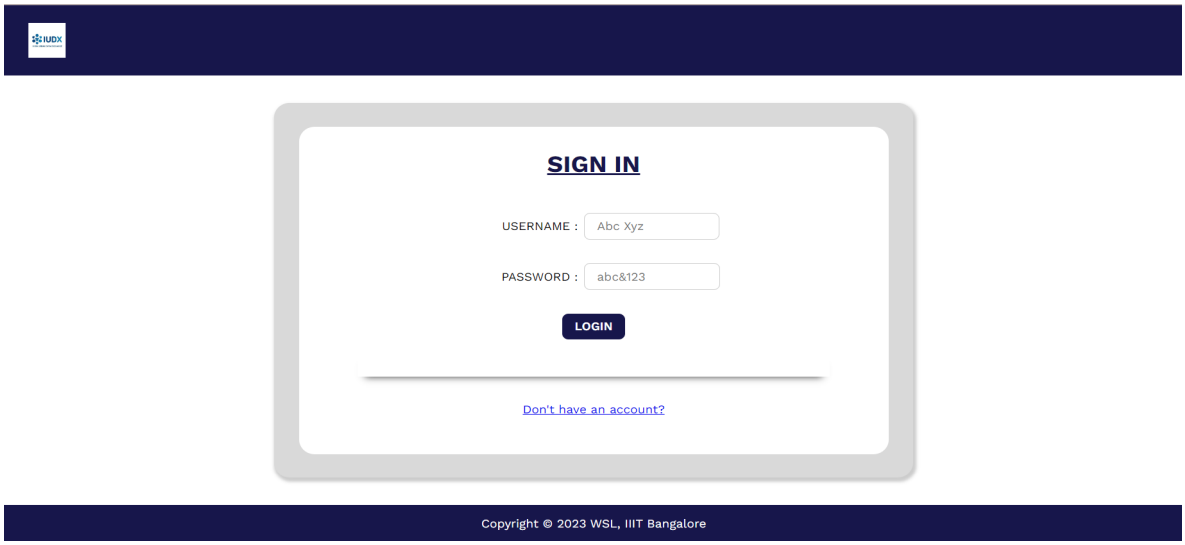
The website was first designed using Figma, a tool that helps create visual layouts. We made several changes to enhance user-friendliness and interactivity of the web pages. The main purpose of the design was to establish a clear direction for our website's frontend. Once the designs were approved, the coding phase of the website began. You can access the Figma files using the link below.

https://www.figma.com/file/w0izanwJRxbB7Nq2YI1wz0/IUDX?type=design&node-id=0%3A1&mode=design&t=LmQDzWvOFI4RnQwt-1

CODING OF THE WEBSITE

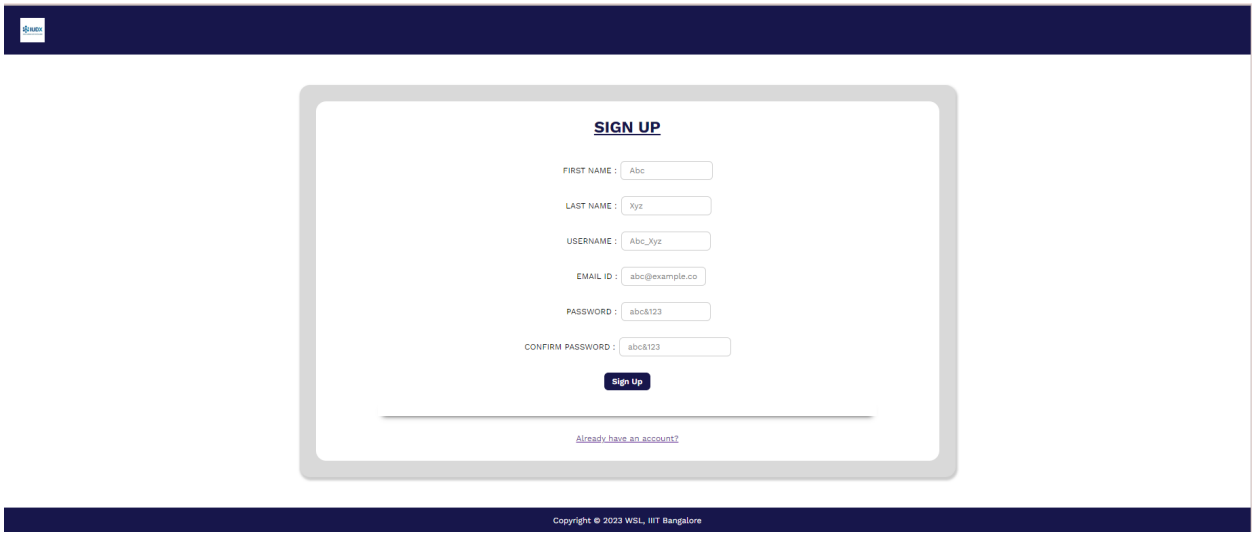
All the webpages were made using HTML, CSS and Javascript. HTML for structuring of the page's content, CSS for styling its appearance and javascript was employed to add interactive functionalities. One thing that is common in all the webpages is to give the page a distinct identity, the IUDX logo was incorporated into the header section, reinforcing branding and recognition. Additionally, a copyright notice was placed in the footer, indicating ownership of the website's content and protecting intellectual property rights. Other than this each webpage plays a different role but are interlinked to each other, a detailed description of every webpage is mentioned below.

1. **Sign-In Page**



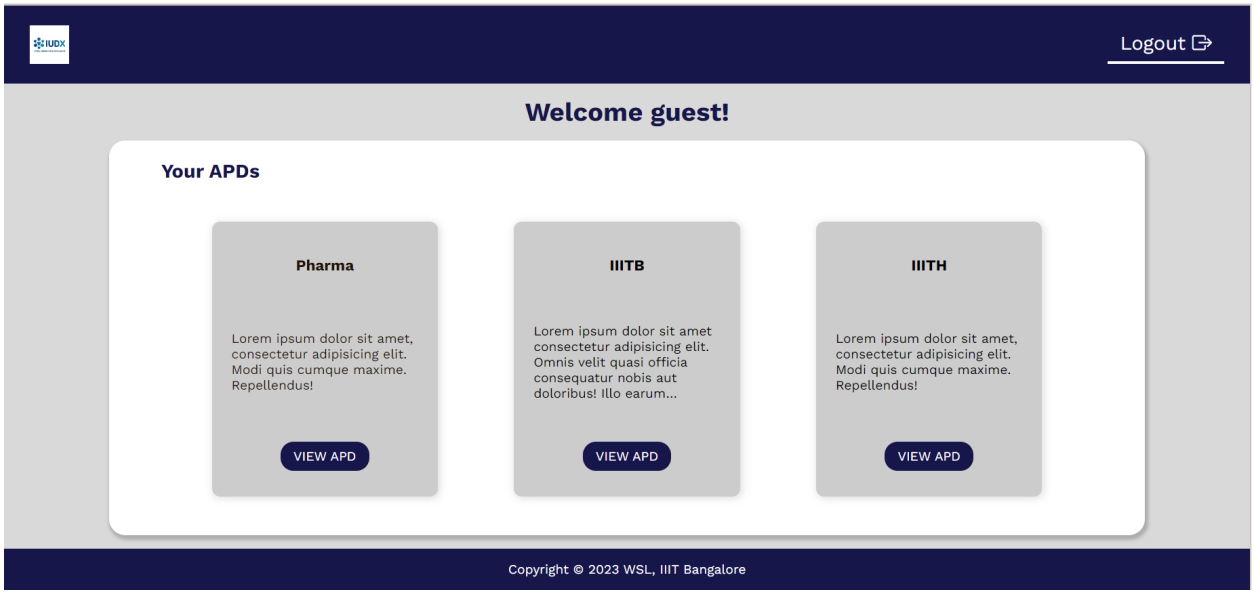
On the Sign-In page, both Users and Admins can access their accounts by entering their respective usernames and passwords. If someone is new to the page and doesn't have an account, they can click on the "Don't have an account?" link, which will take them to the Sign-Up page where they can register themselves.

2. **Sign-Up Page**



The Sign-up page is where new users can create an account for themselves by providing all the necessary information. Once they do that, they will be successfully registered on our website. For those who already have an account, they can simply click on the "Already have an account?" link. This will take them to the Sign-in page, where they can enter their login details to access their existing accounts. This way, we make it easy for both new and returning users to use our website.

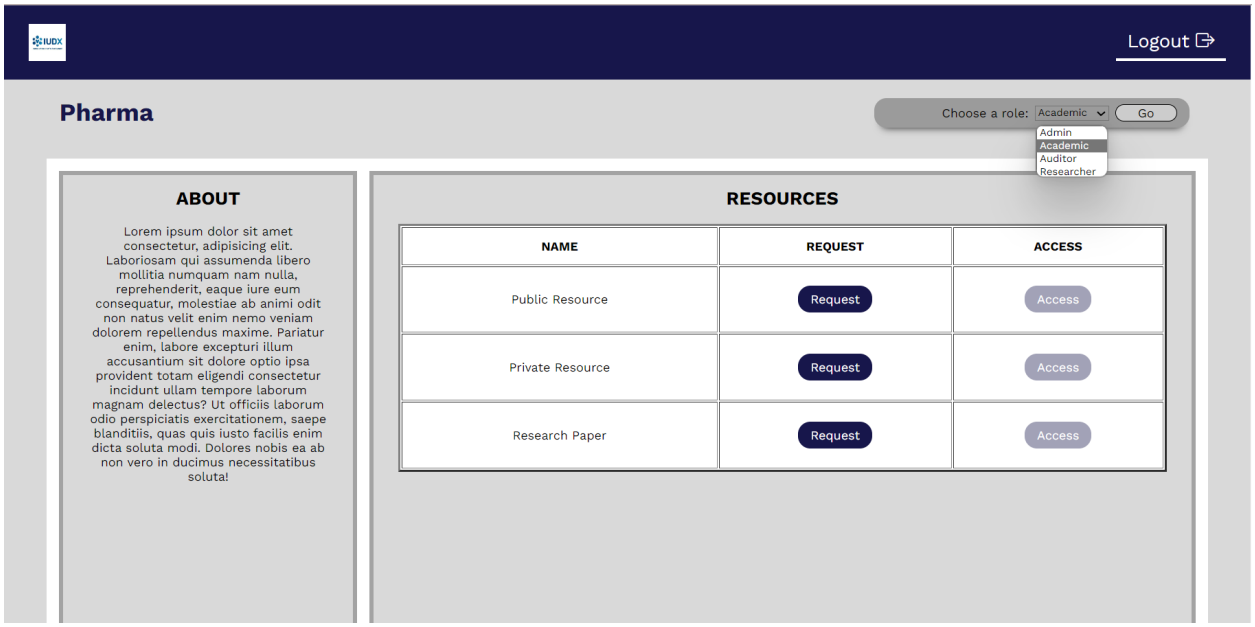
3. **Welcome Page**



NOTE: All the content shown in the above screenshot are just for example.

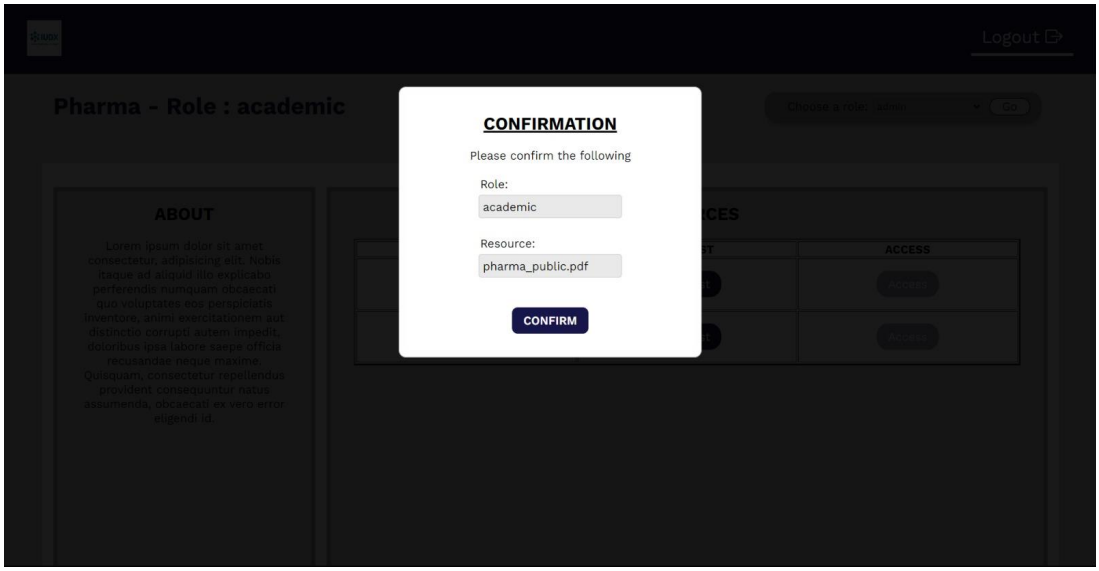
When the user is successfully logged in, a welcome page will appear which will consist of their name in the place of ‘guest’. The page will have a container named ‘Your APDs’ in which it will show the names of all the APD (Access Policy Domain) which they can view in the form of cards. In the cards there will be a little information about the APD it will also have a button named ‘View APD’ and when it is clicked it will direct them to the APD page. There is a logout button in the top right corner which when clicked will take you back to the sign-in page. A little gradient effect was added on the ‘view APD’ button which appears when the cursor is hovered over it.

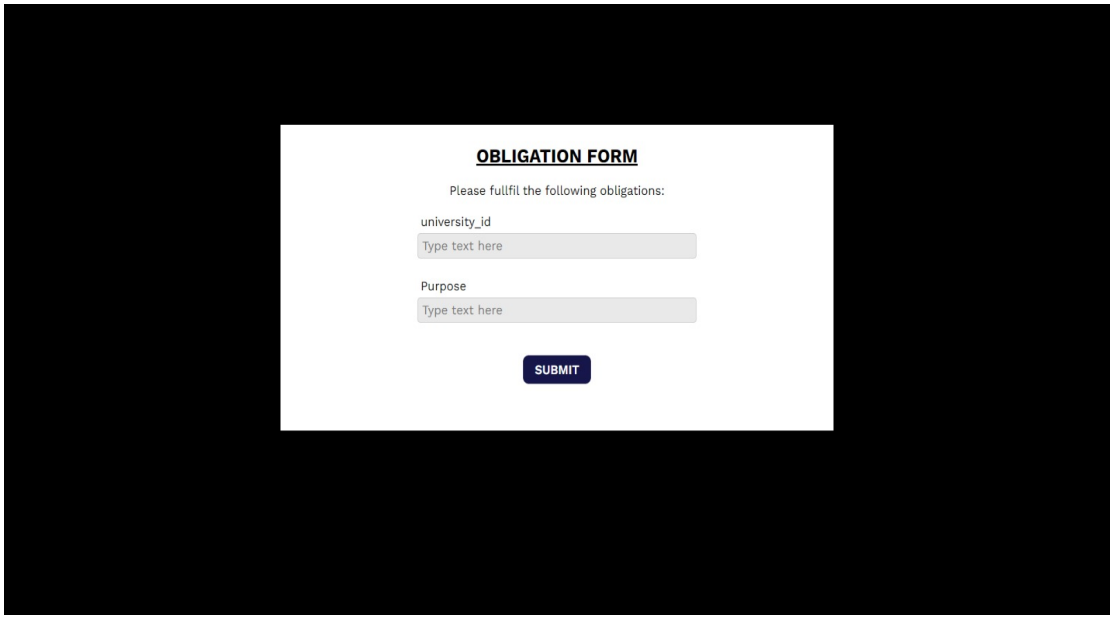
4. APD Page



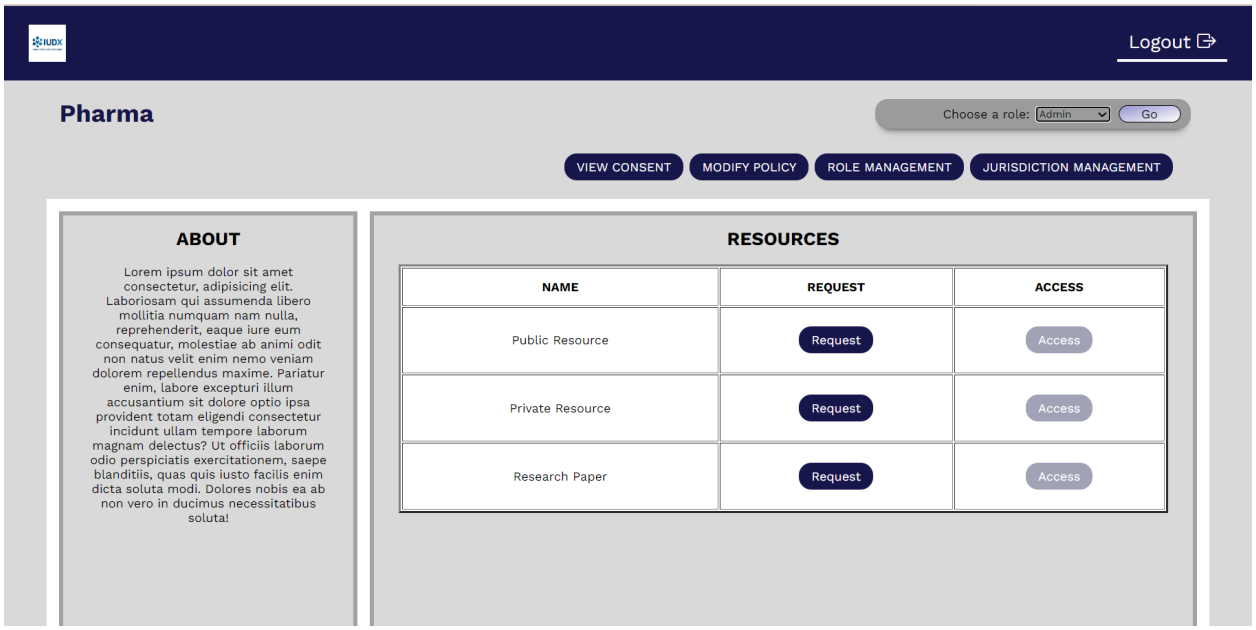
NOTE: All the content shown in the above screenshot are just for example.

In the APD page, you can see the name of the APD and just beside it there is a role drop down from where the user can choose the role that has already been assigned to them and click on ‘GO’ so that the information can be collected from the backend and then shown in the frontend to the user. Below these, there are two things ‘About’ and ‘Resources’. In the About section you can see the information written about the APD and in the Resources section there is a table with all the resources names for which there are two buttons ‘Request’ and ‘Access’. When the request button is clicked a confirmation modal pops-up which when submitted redirects to the obligation form where the user has to fill out the form and if the obligations are filled correctly then the consent artifact will be displayed to the user, and if the wrong obligations are filled then it will throw an error message.





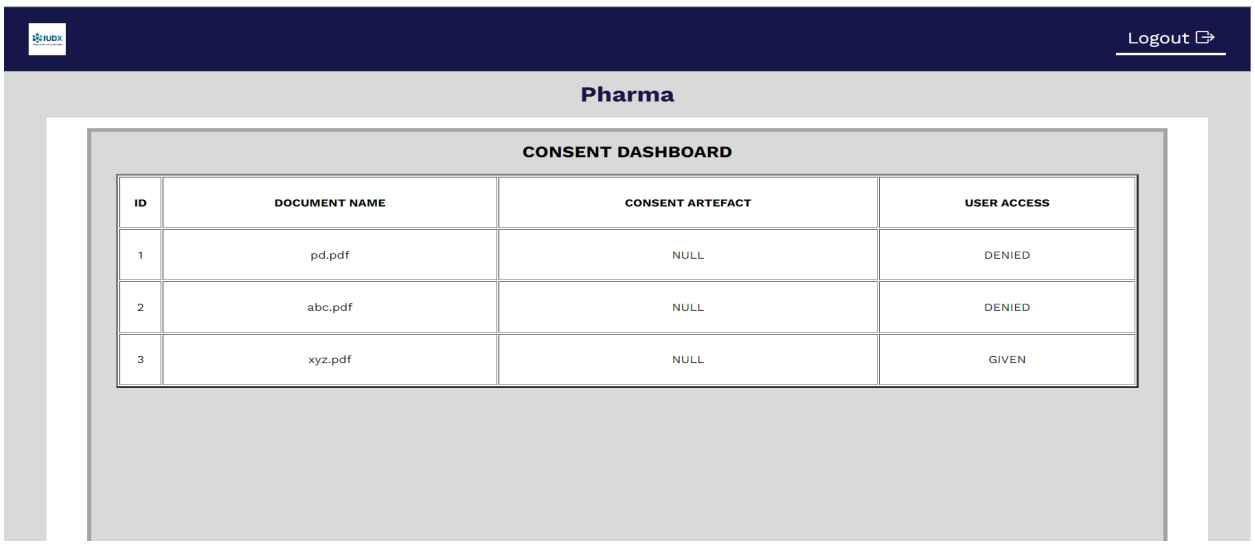
NOTE: All the content shown in the above screenshot are just for example.



NOTE: All the content shown in the above screenshot are just for example.

When the user chooses ‘Admin’ as their role, four buttons will appear- View Consent, Modify Policy, Role Management and Jurisdiction Management and only the user with the admin role will be able to view these buttons and each button opens a new web page when clicked.

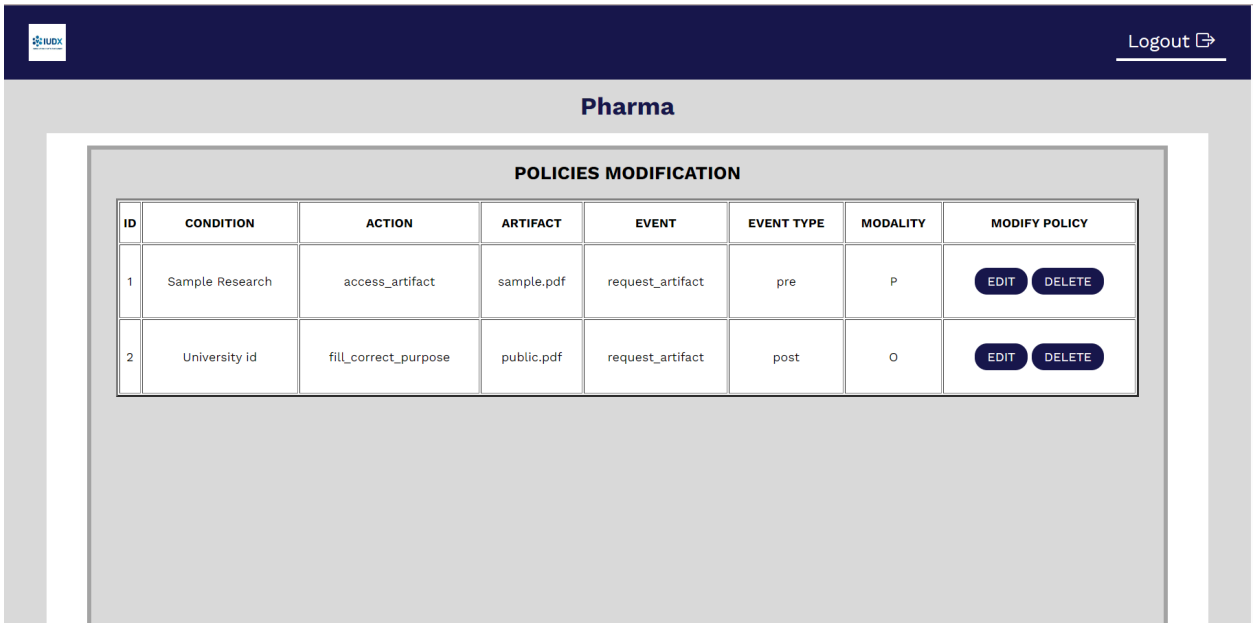
5. **View Consent Page**



NOTE: All the content shown in the above screenshot are just for example.

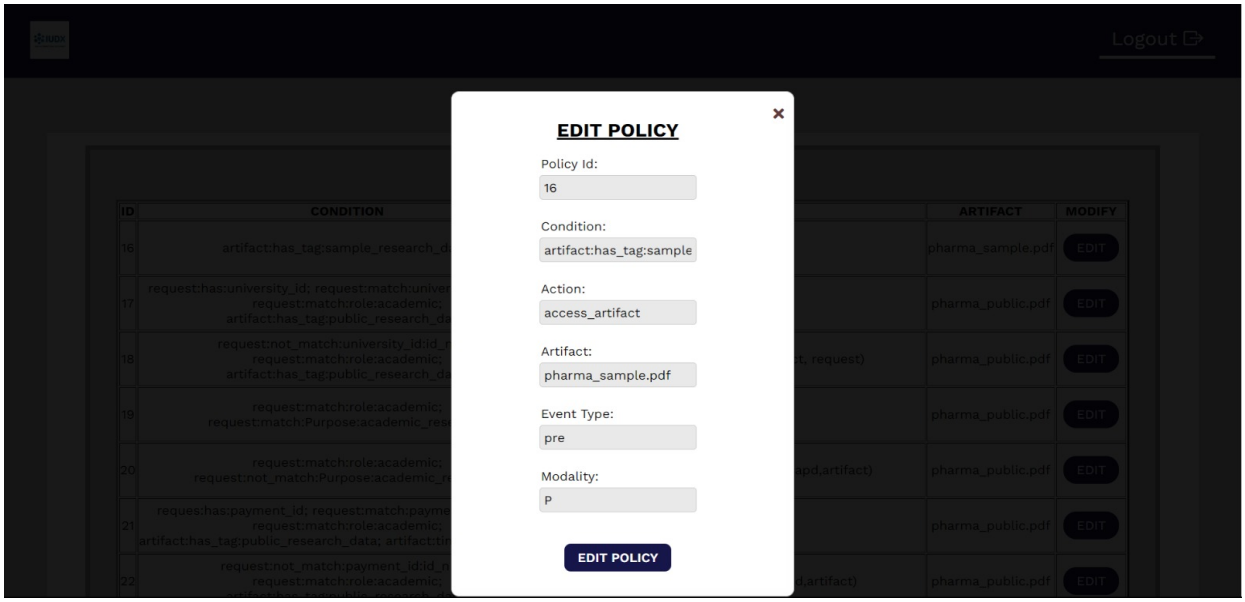
When the 'View Consent' button is clicked , it will open the consent dashboard. In this dashboard, you'll find a table that shows important details. The table includes the document name, the consent artifact, and information about user access. All this data comes directly from the backend database. This setup ensures that you can easily review and manage the consents associated with different documents.

6. Modify Policy Page



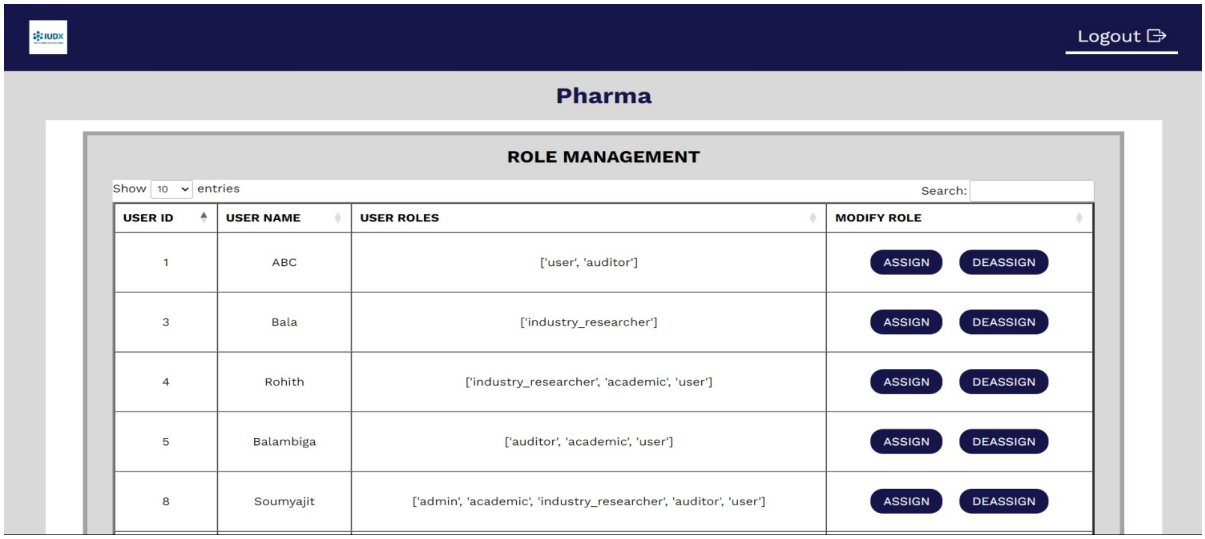
NOTE: All the content shown in the above screenshot are just for example.

When the 'Modify Policy' button is clicked , it will open the policies modification table which shows the condition of the policy with action, artifact, event, event type, modality and a modify policy column. In this page, the admin can modify the policy by editing or deleting the policy. When the admin clicks on the edit button a form will pop-up where they can edit the APD name and many more changes can be made. All the new changes are saved in the backend database. If the admin clicks the delete button that row with its existing policy will be deleted.



NOTE: All the content shown in the above screenshot are just for example.

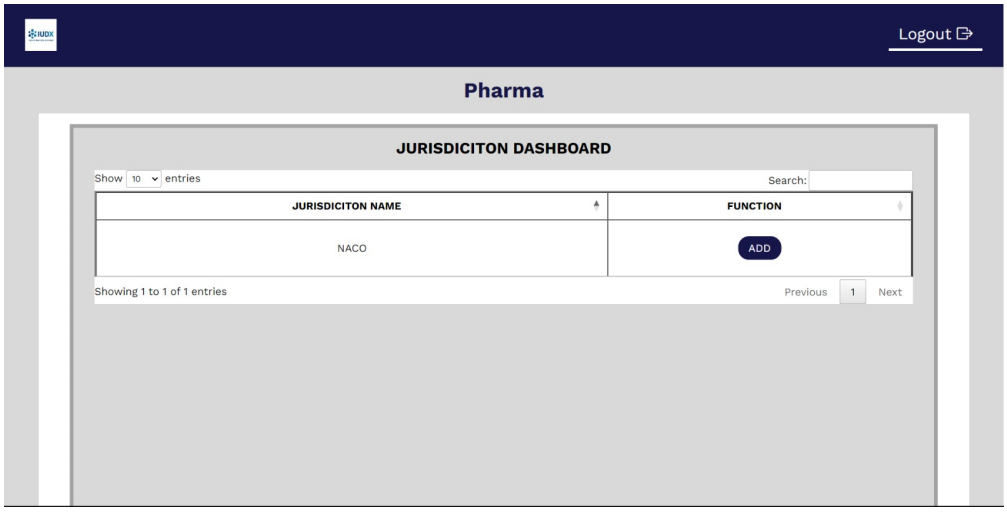
7. **Role Management Page**



NOTE: All the content shown in the above screenshot are just for example.

In the role management page, the admin can assign a new role to the user, or it can deassign the user from any roles that they are already assigned. The admin can control the number of entries they want to see in one page. With the help of the search they can directly search the name of the user and can edit it accordingly.

8. Jurisdiction Management Page



NOTE: All the content shown in the above screenshot are just for example.

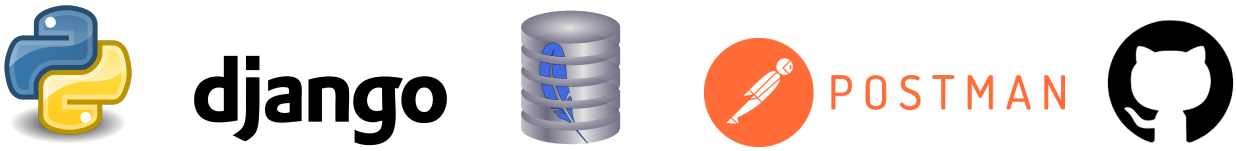
In the jurisdiction management, the admin can see the jurisdiction dashboard where they can see the jurisdiction name and function that the jurisdiction has. And with the ‘add’ button they can add any function to the jurisdiction as they want. With the help of the search option it makes it easier for the admin to search for any jurisdiction by there name.

BACKEND

Intern Name: Soumyajit Mitra

TECH STACKS

For the backend of this website, Python Django has been used and also Django templating for the integration of backend with frontend. The database used is SQLite and Postman is used for API testing. The backend code contains the functionalities through both api calls and views connected to the frontend, both of which can be accessed through any browser or Postman. The whole code is hosted and collaborated using github.



PREPARING THE PROJECT

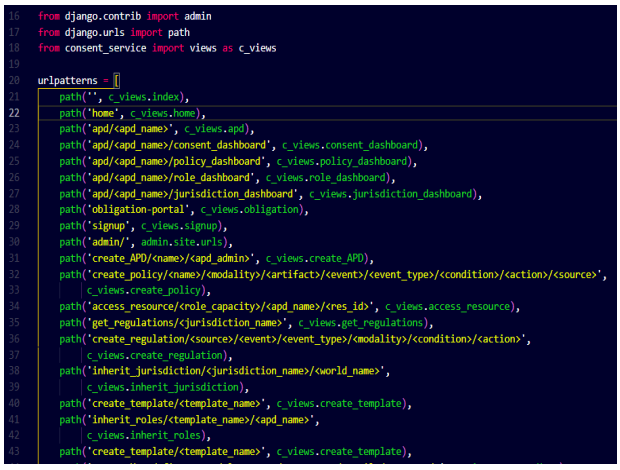
A directory is created for the project and the required environment conditions are applied to it, such that it allows the use of python-django. The ‘startproject’ and ‘startapp’ functions are called using django to set up the coding environment for the project. The required changes are made inside this environment such as connecting the django app to the localhost, providing the address to the base directory for the project, initializing the template and static folder,etc. The django app is now running successfully in the localhost url of the browser.

A private github repository is created and shared with the collaborators with the existing app pushed into the main branch as the default and the collaborators created their separate branches for making their changes to build the project.

CONSTRUCTING THE CODE

Django models are created in models.py which are represented as tables in the database, such as “PolicyBigTable”, “user_role_info”, “Jurisdiction”, etc. These models are then imported into views.py along with other required packages such as “django.contrib.auth” which allows us to authenticate an user. Here, in views.py, all the functionalities are added with specific function names assigned to each one of them such as “create_user”, “Assign_Role”, “access_resource” ,etc. These function names are then called in urls.py for a specific url path along with the required parameters, for e.g. “access_resource/<role_capacity>/<apd_name>/<res_id>”.

Separate functions are created for each page of the frontend of the website as mentioned in the earlier section, handles the functionalities of the corresponding page and interaction with the user. During their sessions in the website, the user is redirected among these functions and a separate function called “evaluate” is created which evaluates all the inputs provided by the user according to the policies and regulations and returns the consent artifact for the request.



```python
from django.contrib import admin
from django.urls import path
from consent_service import views as c_views

urlpatterns = [
    path('', c_views.index),
    path('home', c_views.home),
    path('apd/<apd_name>', c_views.apd),
    path('apd/<apd_name>/consent_dashboard', c_views.consent_dashboard),
    path('apd/<apd_name>/policy_dashboard', c_views.policy_dashboard),
    path('apd/<apd_name>/role_dashboard', c_views.role_dashboard),
    path('apd/<apd_name>/jurisdiction_dashboard', c_views.jurisdiction_dashboard),
    path('obligation-portal', c_views.obligation),
    path('signup', c_views.signup),
    path('admin/', admin.site.urls),
    path('create_APD/<name>/<apd_admin>', c_views.create_APD),
    path('create_policy/<name>/<modality>/<artifact>/<event>/<event_type>/<condition>/<action>/<source>',
        c_views.create_policy),
    path('access_resource/<role_capacity>/<apd_name>/<res_id>', c_views.access_resource),
    path('get_regulations/<jurisdiction_name>', c_views.get_regulations),
    path('create_regulation/<source>/<event>/<event_type>/<modality>/<condition>/<action>',
        c_views.create_regulation),
    path('inherit_jurisdiction/<jurisdiction_name>/<world_name>',
        c_views.inherit_jurisdiction),
    path('create_template/<template_name>', c_views.create_template),
    path('inherit_roles/<template_name>/<apd_name>',
        c_views.inherit_roles),
    path('create_template/<template_name>', c_views.create_template),
```

```python
class APD(models.Model):
    apd_name = models.CharField(max_length=64, null=False, unique=True)
    template_roles = models.JSONField(null=True, blank=True)
    apd_admin = models.ForeignKey(
        User, default=1, verbose_name="User Id", on_delete=models.SET_DEFAULT)

    def __str__(self):
        return self.name


class PolicyBigTable(models.Model):
    apd_name = models.TextField(null=False, max_length=1000)
    modality = models.TextField(null=True, max_length=1000)
    event = models.TextField(null=True, max_length=1000)
    artifact = models.TextField(null=True, max_length=1000)
    condition = models.TextField(null=True, max_length=1000)
    event_type = models.TextField(null=True,max_length=1000)
    action = models.TextField(null=True,max_length=1000)
    Source = models.TextField(null=True, max_length=1000)
    # def __str__(self):
    # return self.user.name

class Resource(models.Model):
    documents = models.FileField(upload_to='document/')
    document_type = models.TextField(null=True)
    document_name = models.TextField(null=True)
    consent_artefact = models.TextField(null=True, blank=True)
    resource_apd = models.TextField(max_length=1000)
    resource_inode = models.JSONField(null=True, blank=True)
```

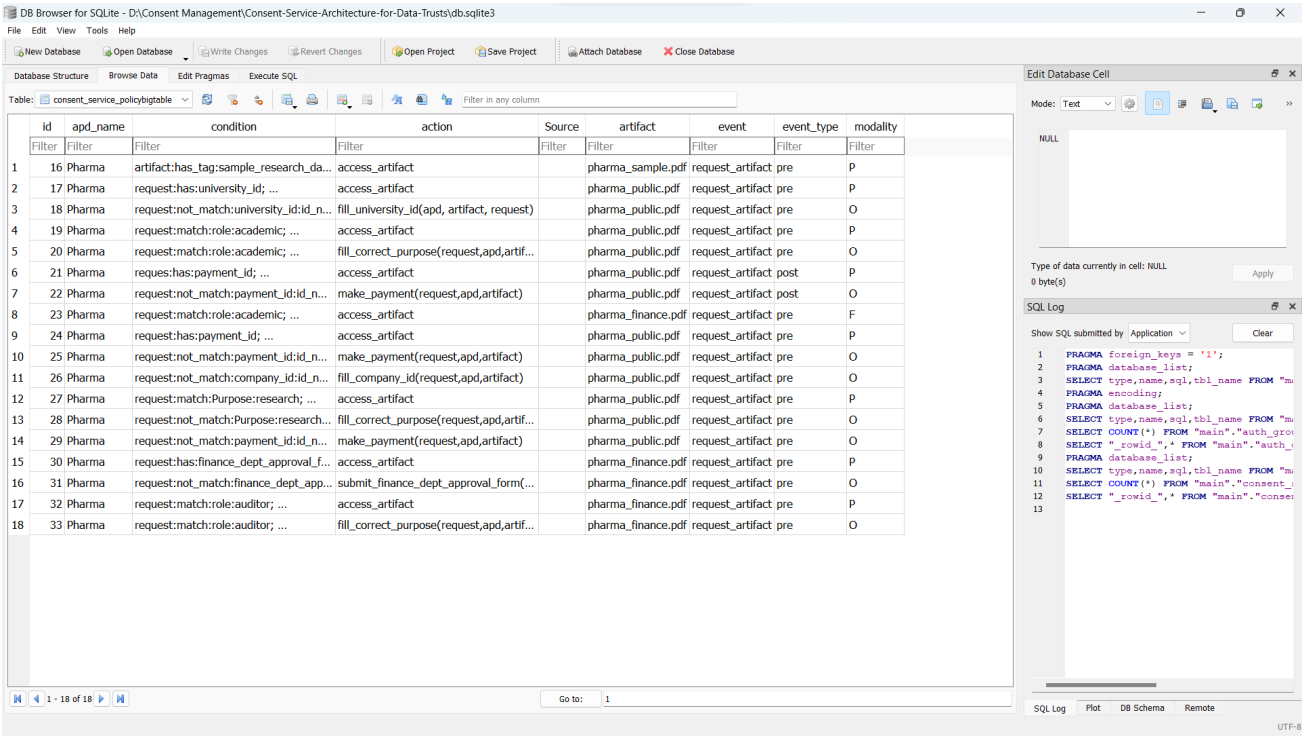
```python
def obligation(request):

def evaluate(res_tag,role_cond_list,user_input,policy_ids,policy_evaluation):

def consent_dashboard(request,apd_name):
    # Displaying the resources and their consent artifacts only for the admin
    resources = Resource.objects.filter(resource_apd=apd_name)
    return render(request, 'consent.html', {'apd_name':apd_name, 'resources':resources})
```

DIVING INTO DATABASE

The database is in the form of a sqlite file inside the directory which can be opened by using DB Browser (SQLite). The database consists of tables created as django models in the code and contain every information that is to be displayed and worked upon. The database also contains the record of every user created to be a part of the website. The data present in the database can easily be accessed by calling the name of the django models into the code.

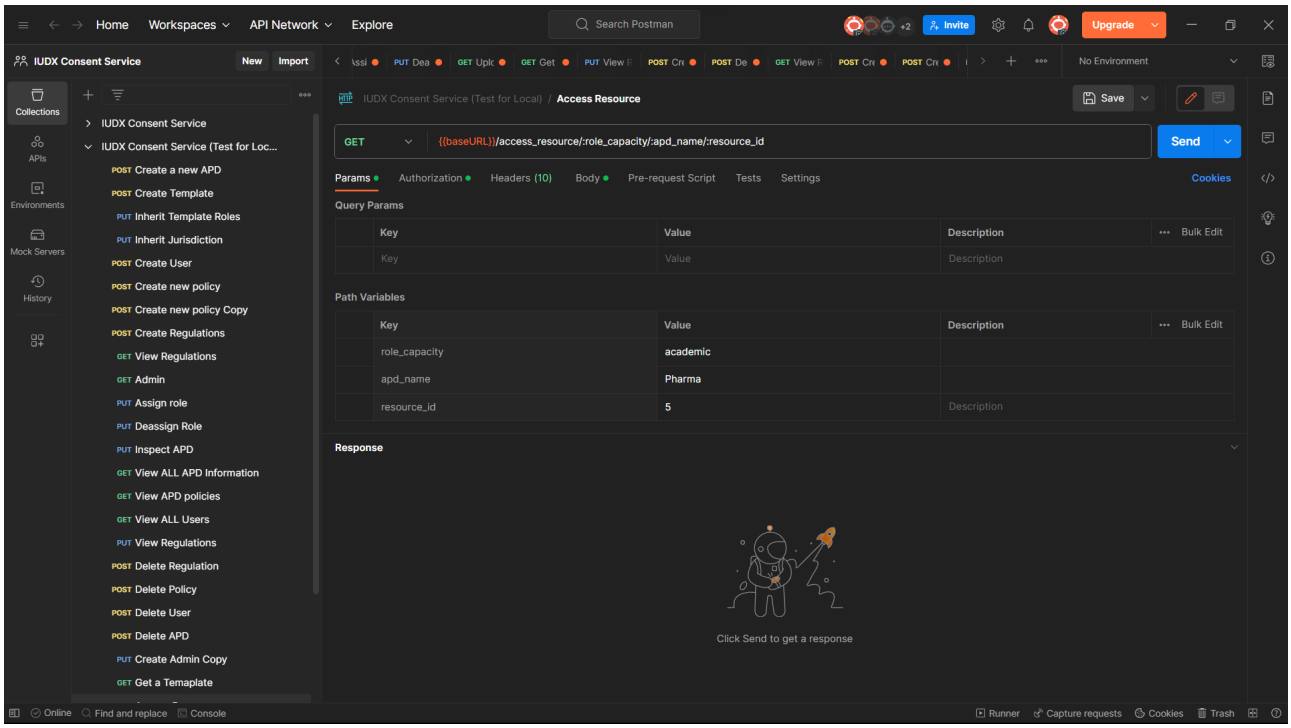


	id	apd_name	condition	action	Source	artifact	event	event_type	modality
1	16	Pharma	artifact:has_tag:sample_research_da...	access_artifact		pharma_sample.pdf	request_artifact	pre	P
2	17	Pharma	request:has:university_id; ...	access_artifact		pharma_public.pdf	request_artifact	pre	P
3	18	Pharma	request:not_match:university_id:id_n...	fill_university_id(apd, artifact, request)		pharma_public.pdf	request_artifact	pre	O
4	19	Pharma	request:match:role:academic; ...	access_artifact		pharma_public.pdf	request_artifact	pre	P
5	20	Pharma	request:match:role:academic; ...	fill_correct_purpose(request,apd,artif...		pharma_public.pdf	request_artifact	pre	O
6	21	Pharma	reques:has:payment_id; ...	access_artifact		pharma_public.pdf	request_artifact	post	P
7	22	Pharma	request:not_match:payment_id:id_n...	make_payment(request,apd,artifact)		pharma_public.pdf	request_artifact	post	O
8	23	Pharma	request:match:role:academic; ...	access_artifact		pharma_finance.pdf	request_artifact	pre	F
9	24	Pharma	request:has:payment_id; ...	access_artifact		pharma_public.pdf	request_artifact	pre	P
10	25	Pharma	request:not_match:payment_id:id_n...	make_payment(request,apd,artifact)		pharma_public.pdf	request_artifact	pre	O
11	26	Pharma	request:not_match:company_id:id_n...	fill_company_id(request,apd,artifact)		pharma_public.pdf	request_artifact	pre	O
12	27	Pharma	request:match:Purpose:research; ...	access_artifact		pharma_public.pdf	request_artifact	pre	P
13	28	Pharma	request:not_match:Purpose:research...	fill_correct_purpose(request,apd,artif...		pharma_public.pdf	request_artifact	pre	O
14	29	Pharma	request:not_match:payment_id:id_n...	make_payment(request,apd,artifact)		pharma_public.pdf	request_artifact	pre	O
15	30	Pharma	request:has:finance_dept_approval_f...	access_artifact		pharma_finance.pdf	request_artifact	pre	P
16	31	Pharma	request:not_match:finance_dept_app...	submit_finance_dept_approval_form(...		pharma_finance.pdf	request_artifact	pre	O
17	32	Pharma	request:match:role:auditor; ...	access_artifact		pharma_finance.pdf	request_artifact	pre	P
18	33	Pharma	request:match:role:auditor; ...	fill_correct_purpose(request,apd,artif...		pharma_finance.pdf	request_artifact	pre	O

A superuser can manually edit the data present in the tables using the “Edit Database Cell” section or delete a record if required.The SQL log also shows the latest queries used by the code. Every entry present in the database should be thoroughly checked and verified by a superuser as a small mistake in the database could lead to bugs and dysfunction of the website.

PEEKING INTO POSTMAN

Postman is an application used for api testing, here it contains all the api functions created through the code and bears a different name for each of them.



It also contains a “Params” section which holds the values of the parameters of the function to be called, a “Authorization” section which asks for the username and password of the admin of the apd for a function only authorized to the admin, a “Body” section which passess information regarding the request of the user.

After filling all these sections and clicking on “Send” an HttpResponse corresponding to the request is displayed in the “Response” section along with the status code.

This postman interface contains more functionalities than the website such as deleting an user or an apd, creating and inheriting templates, etc. which can only be operated by a superuser.