Mobile Web Component Development

Coursework 2 Assignment

Module Leader:

Module Code:

Submitted By

Table of Contents

Resources *4*

Introduction *4*

Background **4**

Objectives **5**

System Architecture *5*

Overview **5**

Key System Components **6**

Type Model Implementation *6*

User Model **8**

Project Group **9**

Client Model **10**

Office Branch **11**

Project Model **11**

Contribution Model **12**

Credential Package Model **13**

Document Model **13**

Announcement Model **14**

Comment Model **14**

Functional Requirements and UI Implementation *15*

User Authentication and Authorization **15**

Client Onboarding **17**

Information Exchange **21**

Knowledge Categorization **22**

Search and Retrieval **23**

Reward Management and Redemption **24**

Announcement Management **25**

Non-Functional Requirements ***26***

Performance **26**

Overview 26

Global Accessibility 26

Responsiveness Metrics 26

Scalability 26

Security **27**

Security Protocols 27

User Authentication 27

Access Control 27

Deployment ***27***

Back End Deployment **27**

Front End Deployment **28**

Conclusion ***29***

Appendices ***31***

References ***30***

Resources

S.N.	Source Name	Link
1.	React Code (GitHub)	https://github.com/suryahangam/kms_frontend
2.	Django Code (GitHub)	https://github.com/suryahangam/kms_backend
3.	Front End URL	https://handk.netlify.app/
4.	Back End URL	https://suryahangam.pythonanywhere.com/admin/login/?next=/admin/

Introduction

Background

In the ever-changing environment of global public relations, the importance of information cannot be understated. Recognizing the need of information sharing, collaboration, and preserving organizational memory, Hill & Knowlton (H&K), a renowned worldwide public relations business, went on a revolutionary journey in early 1999. The idea was triggered by a fundamental realization: their existing knowledge management system was inadequate for meeting the dynamic and sophisticated requirements of a project-centric industry.

As one of the world's leading public relations organizations, with 1,900 personnel and 68 offices in 34 countries, H&K is at the vanguard of communication, where knowledge is not only an advantage but a strategic need. The nature of their projects involves a constant interchange of huge amounts of information between team members, clients, and the ever-changing external landscape. This complicated dance of information flow necessitates a sophisticated knowledge

management system that not only records the essence of previous events, but also acts as a real-time conduit for insights and expertise.

Objectives

The main goals of the technical implementation were multifarious. H&K sought to improve the effectiveness of information sharing within project teams, streamline access to internal, external, and client-specific knowledge, and strengthen the organization's ability to preserve its valuable institutional memory. Understanding the importance of timely and effective onboarding for new employees, as well as the need to reduce the impact of employee turnover, H&K set out to build a knowledge management system that would serve as a dependable repository of collective learning and a dynamic platform for ongoing collaboration.

The journey to redefine their knowledge management approach resulted in the conceptualization and implementation of the hk.net Knowledge Management System, a robust Extranet designed to overcome previous limitations and set new standards for efficient knowledge utilization within the organization.

This extended introduction gives a more thorough backdrop for Hill & Knowlton's development of the hk.net Knowledge Management System, highlighting the industry's complexities, the global structure of operations, and the technical initiative's multifarious goals.

System Architecture

Overview

The Knowledge Management System is a cutting-edge web-based Extranet that was precisely designed to act as a smooth conduit for information exchange between H&K's diversified worldwide workforce and their valued clients. The system is based on the belief that effective knowledge management is crucial for success in the dynamic area of public relations, and it

represents a dedication to promoting cooperation, conserving institutional memory, and ensuring quick access to critical ideas.

Key System Components

Frontend: The user interface allows users to access and interact with the hk.net system. It features an intuitive design that aims to improve the overall user experience.
The user interface is built with **HTML, CSS,** and **JavaScript**, assuring cross-browser compatibility and responsiveness. Additionally, **ReactJS** (an open-source JavaScript package) was utilized to create user interfaces or UI components.

Backend: The backend serves as the system's engine, orchestrating server-side logic, managing data storage, and smoothly integrating with third-party tools. This essential component guarantees that the entire system runs smoothly.
Django (Python-based web framework) was chosen for its scalability and effectiveness in managing server-side logic.

Database: At the heart of knowledge preservation, the database maintains a wide range of data, including project-related emails, customer information, and several knowledge categories. It stores organizational memory, allowing for effective retrieval and utilization.

The **PostreSQL** database management system has been chosen for its dependability and extensive data storing capabilities.

Type Model Implementation

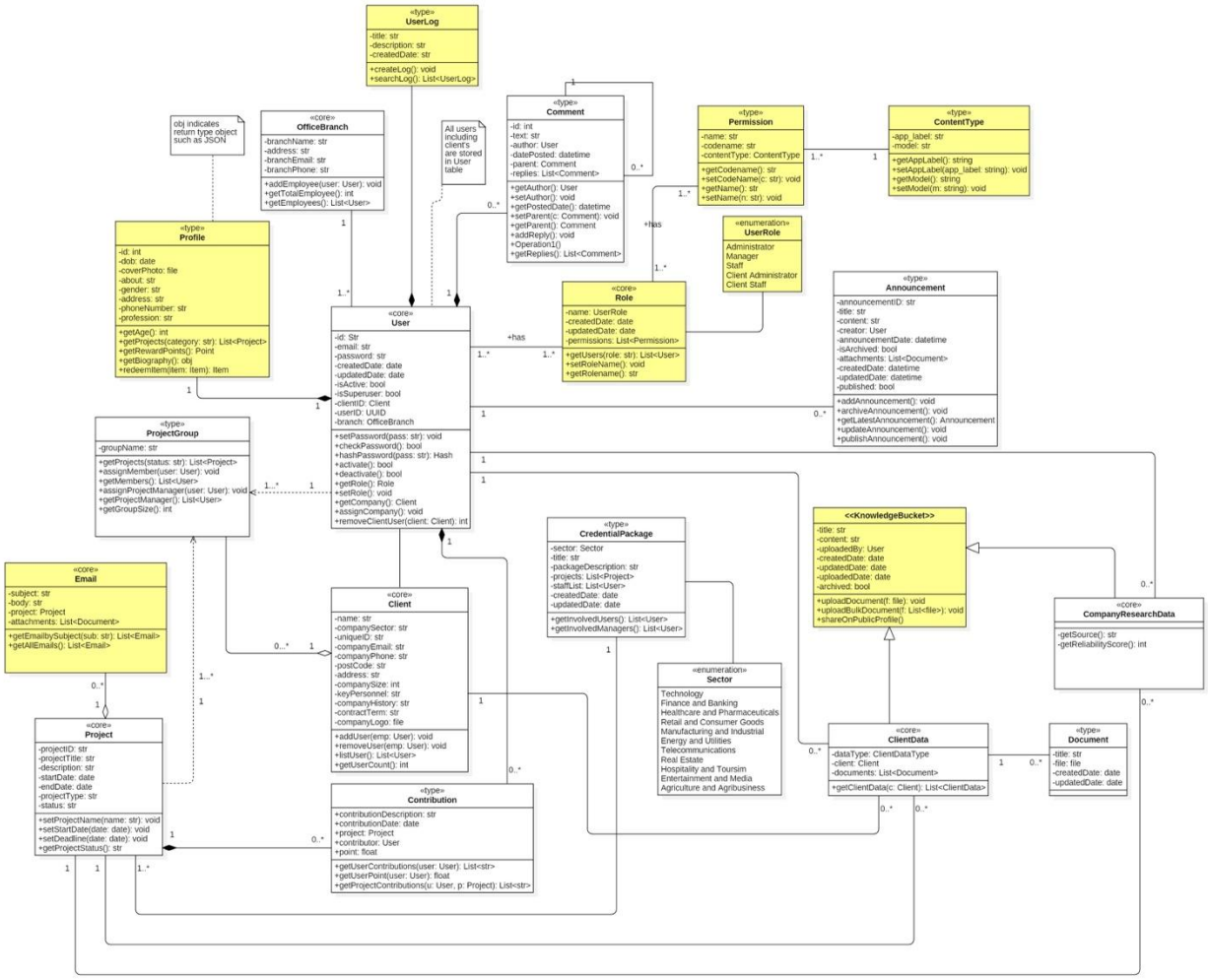


Figure 1 Type Class Model

The suggested Knowledge Management system should be a component-based system built on key type models that hold relevant user and system information. The model includes a class diagram that depicts essential type models such as User, Client, Credential Package, Project, Contribution, Client Data, Document, Company Data, and Announcement. This model comprises all the attributes that are stored in a system. It also shows the link between several models.

This report includes implementation of all type models, its attributes, and methods.

User Model

This model is responsible for storing all user information and carries important user related functionalities such as setting user role, activation/deactivation, password checker etc.

```
Surya Limbu, 2 hours ago | 1 author (Surya Limbu)
class CustomUserManager(BaseUserManager):
    def create_user(self, email, password=None, **extra_fields):
        if not email:
            raise ValueError('The Email field must be set')
        email = self.normalize_email(email)
        user = self.model(email=email, **extra_fields)
        user.set_password(password)
        user.save(using=self._db)
        return user

    def create_superuser(self, email, password=None, **extra_fields):
        extra_fields.setdefault('is_staff', True)
        extra_fields.setdefault('is_superuser', True)

        if extra_fields.get('is_staff') is not True:
            raise ValueError('Superuser must have is_staff=True.')
        if extra_fields.get('is_superuser') is not True:
            raise ValueError('Superuser must have is_superuser=True.')
```

Figure 2 User Manager Type Model Implementation

```python
class CustomUser(AbstractUser):
    username = None
    email = models.EmailField(unique=True)
    client = models.ForeignKey('client.Client',
                               on_delete=models.SET_NULL, null=True)
    branch = models.ForeignKey(OfficeBranch,
                               on_delete=models.SET_NULL, null=True)

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = []

    objects = CustomUserManager()

    def __str__(self):
        return self.email

    def get_company(self):
        self.branch.branch_name      You, 3 seconds ago • Uncommitted
```

Figure 3 User Type Model Implementation

Project Group

This model is responsible for storing project groups. Project groups is made of project and involved users in particular project. It works as a bridge entity between CustomUser and Project models.

```python
class ProjectGroup(TimestampedModel):
    group_name = models.CharField(max_length=250)
    project = models.ForeignKey(Project, on_delete=models.SET_NULL, null=True)
    user = models.ForeignKey('user.CustomUser', on_delete=models.SET_NULL, null=True)

    def __str__(self):
        return self.group_name
```

Figure 4 ProjectGroup Type Model

Client Model

The primary purpose of the Client Model is to store comprehensive information about each client. This information may include details such as the client's name, contact information, industry, account activity, budgets, and any other relevant data points.

```
Surya Limbu, 2 hours ago | 1 author (Surya Limbu)
class Client(TimestampedModel):
    TECHNOLOGY = 'Technology'
    FINANCE = 'Finance'
    HEALTHCARE = 'Healthcare'
    RETAIL = 'Retail'      Surya Limbu, 3 weeks ago • create: app setup

    # Define choices as tuples of (value, display_name)
    SECTOR_CHOICES = [
        (TECHNOLOGY, 'Technology'),
        (FINANCE, 'Finance'),
        (HEALTHCARE, 'Healthcare'),
        (RETAIL, 'Retail'),
        # Add other sectors as needed
    ]

    TERM_CHOICES = [
        ('6', '6 Months'),
        ('9', '9 Months'),
        ('12', '12 Months'),
        ('24', '24 Months'),
        ('36', '36 Months'),
        ('0', 'Unlimited'),
    ]

    name = models.CharField(max_length=255)
    sector = models.CharField(max_length=200, choices=SECTOR_CHOICES)
    unique_client_id = models.CharField(max_length=50,
                                        default=generate_unique_ID, unique=True)
    company_email = models.EmailField()
    company_phone = models.CharField(max_length=20)
    post_code = models.CharField(max_length=50)
    address1 = models.CharField(max_length=50)
    address2 = models.CharField(max_length=50, null=True, blank=True)
    company_size = models.IntegerField(null=True, blank=True)
    key_personnel = models.CharField(max_length=100, null=True, blank=True)
    company_history = models.TextField(null=True, blank=True)
    contactTerm = models.CharField(max_length=20, choices=TERM_CHOICES, default=0)
    company_logo = models.ImageField(upload_to='company_logo')

    def __str__(self):
        return self.name

    def add_user(self, user):
        user.client.id = self.id
        user.save()
        return user

    def remove_user(self, user):
        pass
```

Figure 5 Client Type Model

Office Branch

The primary purpose is to store and organize information about different office branches. This includes details such as the branch's location, contact information, office-specific protocols, and any other relevant data points.

```
Surya Limbu, 3 weeks ago | 1 author (Surya Limbu)
class OfficeBranch(TimestampedModel):      Surya Limbu, 3 we
    branch_name = models.CharField(max_length=100)
    address1 = models.CharField(max_length=200)
    address2 = models.CharField(max_length=200)
    post_code = models.CharField(max_length=40)
    country = models.CharField(max_length=30)
    branch_email = models.EmailField()
    branch_phone = models.CharField(max_length=20)

    def __str__(self):
        return self.branch_name

    def add_employee(self, user):
        pass

    def get_employee_list(self):
        pass

    def remove_employee(self):
        pass
```

Figure 6 Office Branch Type Model

Project Model

The Project Model encapsulate information about projects within the Hill & Knowlton (H&K) organization. Each project is represented as an entity within this model, capturing relevant details to facilitate efficient project management, collaboration, and knowledge sharing. The primary purpose of this model is to store Project related information. It stores data related to different branch offices such as name, address, and contact.

```python
class Project(TimestampedModel):      Surya Limbu, 3 weeks ago • create: app set
    STATUS_CHOICES = [
        ('not_started', 'Not Started'),
        ('in_progress', 'In Progress'),
        ('completed', 'Completed'),
        ('abandoned', 'Abandoned'),
        ('on_hold', 'On Hold'),
    ]
    project_id = models.CharField(max_length=50, default=generate_unique_ID,
                                  unique=True)
    project_title = models.CharField(max_length=200)
    description = models.TextField(null=True)
    start_date = models.DateField(null=True, blank=True)
    end_date = models.DateField(null=True, blank=True)
    project_type = models.CharField(max_length=50) # make it choice field
    status = models.CharField(choices=STATUS_CHOICES, max_length=20,
                              default='not_started')

    def start_project(self):
        self.start_date = timezone.now().date()
        return self.start_date

    def set_deadline(self):
        if self.start_date:
            self.end_date = timezone.now().date()
            return True
        return False
```

Figure 7 Project Type Model implementation

Contribution Model

This model is designed for handling contributions within a knowledge management system. It includes fields for a short description, contribution date, associated projects, contributing user, and a numerical value or points associated with the contribution. The model utilizes foreign key relationships with the Projects and custom User.

```
Surya Limbu, 3 weeks ago | 1 author (Surya Limbu)
class Contribution(TimestampedModel):
    description = models.CharField(max_length=255, null=True, blank=True)
    contribution_date = models.DateField(default=timezone.now)
    project = models.ForeignKey(Project, on_delete=models.SET_NULL, null=True)
    contributor = models.ForeignKey('user.CustomUser', on_delete=models.SET_NULL, null=True)
    contribution_point = models.FloatField()

    def __str__(self):
        if self.description:
            return self.description
        return self.project.title

    def get_user_contribution(self, user):
        pass

    def get_user_points(self, user):
        pass

    def get_project_contributions(self, project):
        pass
```

Figure 8 Contribution Type Model

Credential Package Model

Document Model

This model stores all files data and has a reference in related tables. It will create a separate folder named *information*_files. Since, the files gets stored separately, it only stores references to that path.

```
Surya Limbu, 3 weeks ago | 1 author (Surya Limbu)
class Document(TimestampedModel):
    title = models.CharField(max_length=200, null=True, blank=True)
    file = models.FileField(upload_to='information_files')
    company_research_data = models.ForeignKey(CompanyResearchData,
                                              on_delete=models.SET_NULL,
                                              null=True)
    client_data = models.ForeignKey(ClientData,
                                    on_delete=models.SET_NULL,
                                    null=True)

    def __str__(self):
        return self.title
```

Figure 9 Document Type Model

Announcement Model

This model Is designed for managing announcements within a system. It includes fields for the title, content, creator, announcement date, and flags for archiving and publication status. The model provides methods for archiving announcements, retrieving the latest announcement, and publishing announcements.

```python
class Announcement(TimestampedModel):
    title = models.CharField(max_length=255)
    content = models.TextField()
    creator = models.ForeignKey(CustomUser, on_delete=models.SET_NULL,
                                null=True)
    announcement_date = models.DateTimeField()
    is_archive = models.BooleanField(default=False)
    published = models.BooleanField(default=False)

    def archive_announcement(self):
        self.is_archive = True
        return True

    def get_latest(self, count=5):
        return Announcement.objects.all().order_by('-announcement_date')[:count]

    def publish(self, announcement=None):
        if announcement:
            announcement.published = True
            announcement.save()
            return announcement.id
        self.published = True
        return self.id
```

Figure 10 Announcement Type Model

Comment Model

Functional Requirements and UI Implementation

This section depicts the process of translating the functional requirements of a system into actual working functionalities or features withing the application.

User Authentication and Authorization

This is the process of verifying the identity of a user, typically during login, it ensures that the person trying to access a system or application is indeed the person they claim to be.

Authentication process has been handled using JSON Web Tokens. It's a common approach in modern web applications to secure access to resources.

```
You, 3 days ago | 1 author (You)
class LoginView(APIView):

    def post(self, request):
        email = request.data.get('email')
        password = request.data.get('password')

        user = authenticate(
            username=email, password=password)
        if user:
            refresh = RefreshToken.for_user(user)

            return JsonResponse(
                {
                'message': 'Successful',
                'status': 1,
                'data': [{
                    'refresh':str(refresh),
                    'access':str(refresh.access_token)
                    }
                ]}
            )
        return JsonResponse(
            {
                'message': 'Invalid Credentials',      You, 3
                'status': 0,
                'data': []
            })
```

Figure 11 Backend Authentication Implementation

```
export default function SignIn() {

  const [user, setUser] = useState("")
  const auth = useAuth()
      You, now • Uncommitted changes
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');

  const handleLogin = async (e) => {
    e.preventDefault();

    try {
      const response = await fetch(`${PROJECT_API}/login`, {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({ username, password }),
      });

      if (!response.ok) {
        throw new Error('Login failed');
      }

      const data = await response.json();
      const token = data.token;
      setToken(token);
    } catch (error) {
      console.error('Login failed', error);
    }
  };
```

Figure 12 Authentication Component Front

The **Login.js** component is a React component that handles user authentication in a web application. It uses the fetch API to send a POST request to a server endpoint for user login. The user provides a username and password through a form, and upon successful authentication, a JWT token is extracted from the server response. The **setToken** function is then called to update the application state with the obtained token. Error handling is implemented to catch and log any errors that may occur during the login process. The component is designed to be integrated into a broader React application for user authentication.

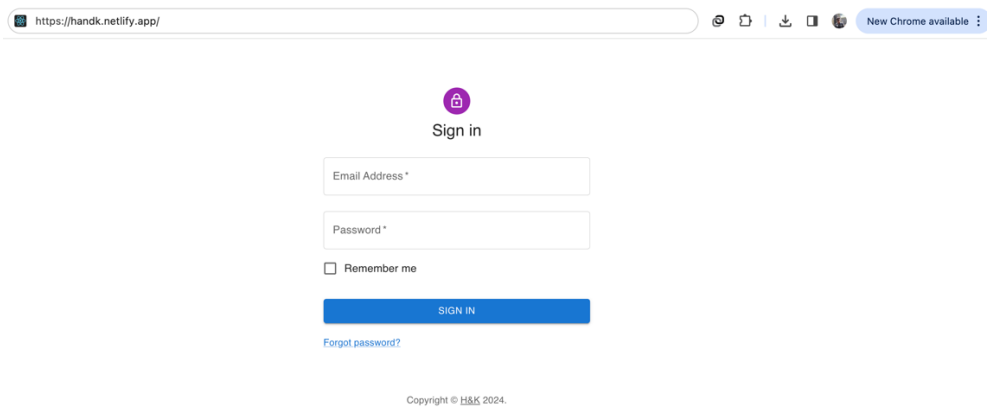


Figure 13 Authentication User Interface

Client Onboarding

The **ClientCreationForm** React component manages the creation or update of client information, incorporating state management, form handling, file uploads, and Material-UI components within a dashboard layout.

```
Surya Limbu, 5 hours ago | 1 author (Surya Limbu)
import {React, useEffect, useState} from 'react';
import { useParams, useNavigate } from 'react-router-dom';
import { TextField, Button, Select, MenuItem, FormControl, InputLabel, Paper, Typography, Grid } from '@mui/mat
import Dashboard from '../../components/dashboard/Dashboard';
import { PROJECT_API } from '../../components/apis/api';
import { CLIENT_API } from '../../components/apis/api';

const ClientCreationForm = () => {

  const navigate = useNavigate();
  const {id} = useParams();

  const [formData, setFormData] = useState({
    name: '',
    sector: '',
    company_email: '',
    company_phone: '',
    post_code: '',
    address1: '',
    address2: '',
    company_size: '',
    key_personnel: '',
    company_history: '',
    contactTerm: '',
    company_logo: null,
  });
```

Figure 14 Client Onboarding 1

```
useEffect(() => {
  if (id){
    console.log("Getting project details....")
      const fetchProjectData = async () => {
          try{
              const projectUrl = PROJECT_API + `${id}/`
              const projectResponse = await fetch(
                  projectUrl,
                  {
                      method: 'GET',
                      headers: {
                          'Accept': 'application/json',
                          'Content-Type': 'application/json'
                      }
                  });
              if (!projectResponse.ok){
                  const responseData = await projectResponse.json();
                  console.error('Error', responseData);
              }
              else {
                  const projectData = await projectResponse.json();
                  console.log("form data:: ", projectData)
                  setFormData(projectData);
              }

          } catch (error){
              console.error('Error: ', error)
          }
      };

      fetchProjectData();
  }
}, [id])
```

Figure 15 Client Onboarding 2

```
  const handleChange = (e) => {
    const {name, value} = e.target;
    setFormData({ ...formData, [name]: value});
  }

  const handleFileChange = (e) => {
    console.log("File change triggered...")
    const file = e.target.files[0];
    console.log("File::: ", file)
    setFormData({ ...formData, company_logo: file});
  }

  // Function to handle form submission
  const handleSubmit = async (event) => {
    event.preventDefault();
    console.log('form data: ', formData);

    try{

        const formDataForFile = new FormData();
        Object.entries(formData).forEach(([key, value]) => {
            formDataForFile.append(key, value)
        })
        // formDataForFile.append('company_logo', formData.company_logo);

        // const formDataForFields = { ...FormData };
        // delete formDataForFields.company_logo;
        const apiUrl = CLIENT_API;
        const response = await fetch(apiUrl,
        {
```

Figure 16 Client Onboarding 3

```
            method: 'POST',
            redirect: 'follow',
            headers: {
                'Accept': 'application/json',
                // 'Content-Type': 'multipart/form-data'
            },
            // body: JSON.stringify(formData)
            body: formDataForFile,
        });

        if (!response.ok){
            const responseData = await response.json();
            console.log("error::: ", responseData);
            // throw new Error('Failed to create project');
        }

        const responseData = await response.json();
        console.log('response: ', responseData);
        console.log('Project created successfully');
        navigate('/clients');
    }
    catch(error){
        console.error('Error creating project:', error);
    }

};

return (
    <Dashboard>
    <Paper elevation={3} style={{ padding: 20, margin: 20, borderRadius: 10 }}>
      <Typography variant="h6" gutterBottom style={{ color: '#3B71CA' }}>
        Add Client
      </Typography>
```

Figure 17 Client Onboarding 4

```
126         <form onSubmit={handleSubmit} encType="multipart/form-data">
127         <Grid container spacing={3}>
128           {/* Add other form fields based on your model */}
129           <Grid item xs={12} sm={6}>
130             <TextField
131               label="Name"
132               fullWidth
133               required
134               name="name"
135               value={formData.name}
136               onChange={handleChange}
137             />
138           </Grid>
139           <Grid item xs={12} sm={6}>
140             <FormControl fullWidth>
141               <InputLabel>Sector</InputLabel>
142               <Select
143                 label="Sector"
144                 name="sector"
145                 value={formData.sector}
146                 onChange={handleChange}
147                 required
148               >
149                   <MenuItem value="Technology">Technology</MenuItem>
150                   <MenuItem value="Finance">Finance</MenuItem>
151                   <MenuItem value="Healthcare">Healthcare</MenuItem>
152                   <MenuItem value="Retail">Retail</MenuItem>
153                 {/* {SECTOR_CHOICES.map((sector) => (
154                   <MenuItem key={sector} value={sector}>
155                     {sector}
156                   </MenuItem>
157                 ))} */}
158               </Select>
```

Figure 18 Client Onboarding 5

```
269           </Grid>
270           <Grid item xs={12}>
271             <Button type="submit" variant="contained" color="primary">
272               Submit
273             </Button>
274           </Grid>
275         </Grid>
276       </form>
277       </Paper>
278       </Dashboard>
279     );
280   };
281
282   export default ClientCreationForm;
283
284
285
286
```

Figure 19 Client onboarding 6

The provided code represents a Client Onboarding Component, responsible for displaying a list of clients retrieved from the client **list API**. This component features a form field designed to capture all client attributes for storage in our database through the **create API**. Utilizing the **useState** React hook facilitates efficient management of the form data state. The implementation incorporates a toggle mechanism between create and update functionalities, contingent on the presence of the parameter variable named ID. The Navigate feature is employed to redirect the user upon successful form submission, enhancing the overall user experience and navigation flow within the application.

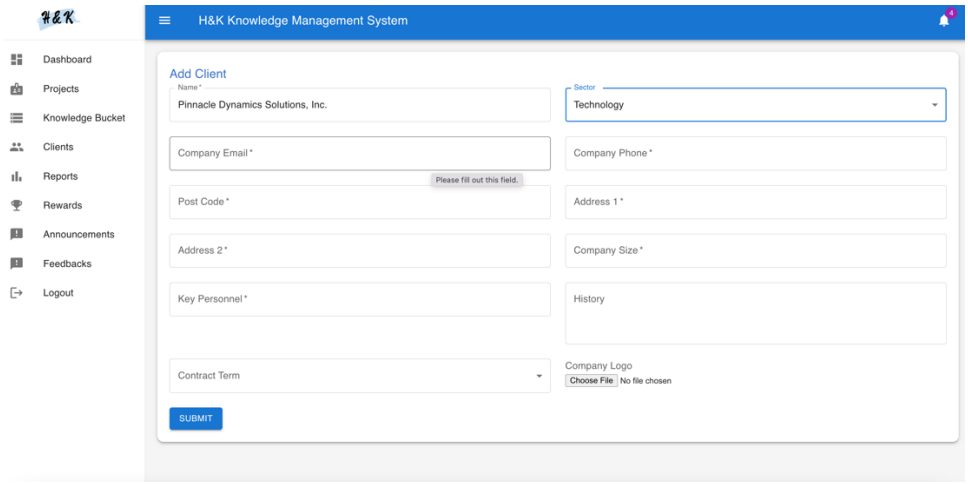


Figure 20 Client onboarding UI

Information Exchange

```
18    const handlePageChange = (page) => {
19        // Add your logic to fetch and display data for the selected page
20        setCurrentPage(page);
21      };
22
23    return (
24      <Dashboard>
25        {/* <SearchComponent /> */}
26        <Grid container spacing={1} style={{margin:2}}>
27        {items.map((item, index) => (
28            <Grid key={index} item xs={12} md={6} lg={3}>
29            <Paper style={{ padding: 16, display: 'flex', flexDirection: 'column', alignItems: 'center' }}
30                component={Link} to={`/bucket-detail/${item}`}>
31                <FolderIcon style={{ fontSize: 50, marginBottom: 10 }} />
32                {`${item}`}
33            </Paper>
34            </Grid>
35        ))}
36        </Grid>      Surya Limbu, 5 hours ago • final UI
37        <CustomPagination
38        currentPage={currentPage}
39        totalPages={totalPages}
40        onPageChange={handlePageChange}
41      />
42    </Dashboard>
43    );
44  };
45
46  export default KnowledgeBucket;
47
```

Figure 21 Information Exchange Implementation

The provided code represents an information listing page, where user uploads information in many forms such as, files or folders. Those items are listed and can be accessed accordingly. The given code is an **Information Component**, it is iterating all the available items from Information list API and showing them in a grid.

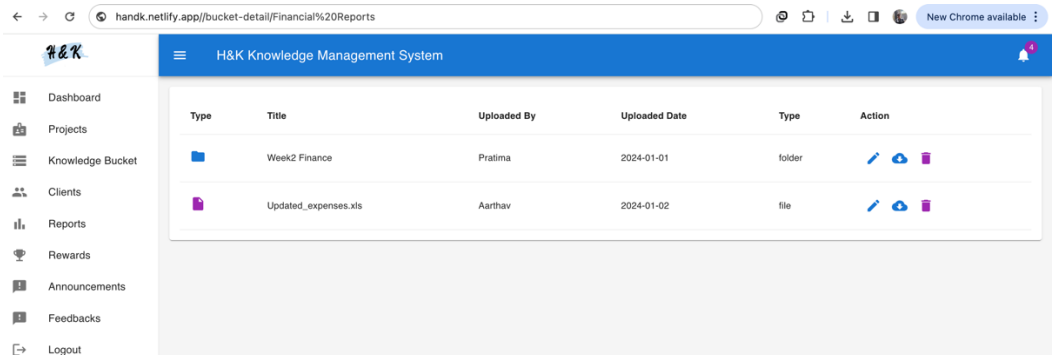


Figure 22 Information Exchange UI

Knowledge Categorization

```
18    const handlePageChange = (page) => {
19       // Add your logic to fetch and display data for the selected page
20       setCurrentPage(page);
21     };
22
23   return (
24     <Dashboard>
25       {/* <SearchComponent /> */}
26       <Grid container spacing={1} style={{margin:2}}>
27       {items.map((item, index) => (
28           <Grid key={index} item xs={12} md={6} lg={3}>
29           <Paper style={{ padding: 16, display: 'flex', flexDirection: 'column', alignItems: 'center' }}
30               component={Link} to={`/bucket-detail/${item}`}>
31               <FolderIcon style={{ fontSize: 50, marginBottom: 10 }} />
32               {`${item}`}
33           </Paper>
34           </Grid>
35       ))}
36       </Grid>      Surya Limbu, 5 hours ago • final UI
37       <CustomPagination
38       currentPage={currentPage}
39       totalPages={totalPages}
40       onPageChange={handlePageChange}
41     />
42     </Dashboard>
43   );
44 };
45
46 export default KnowledgeBucket;
47
```

Figure 23 Knowledge Categorization Implementation

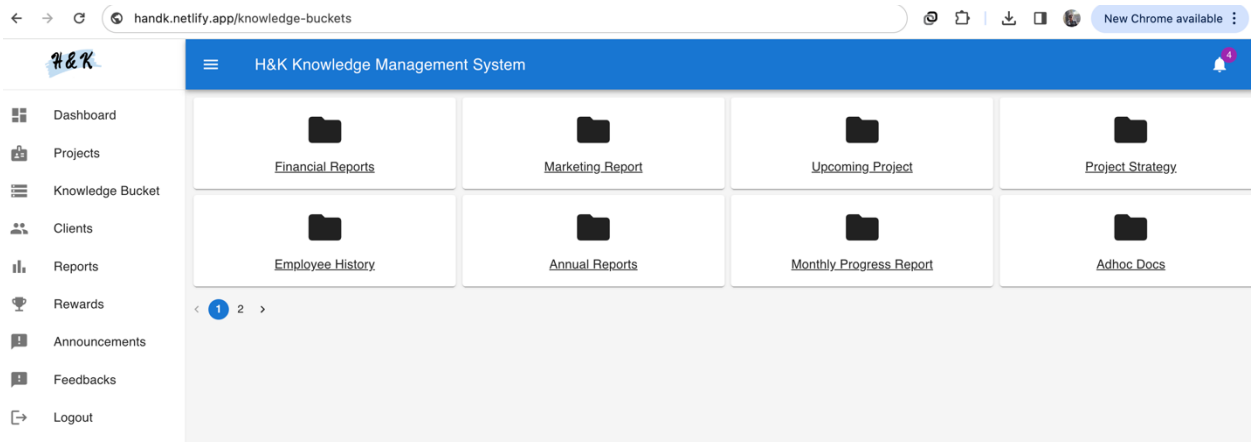
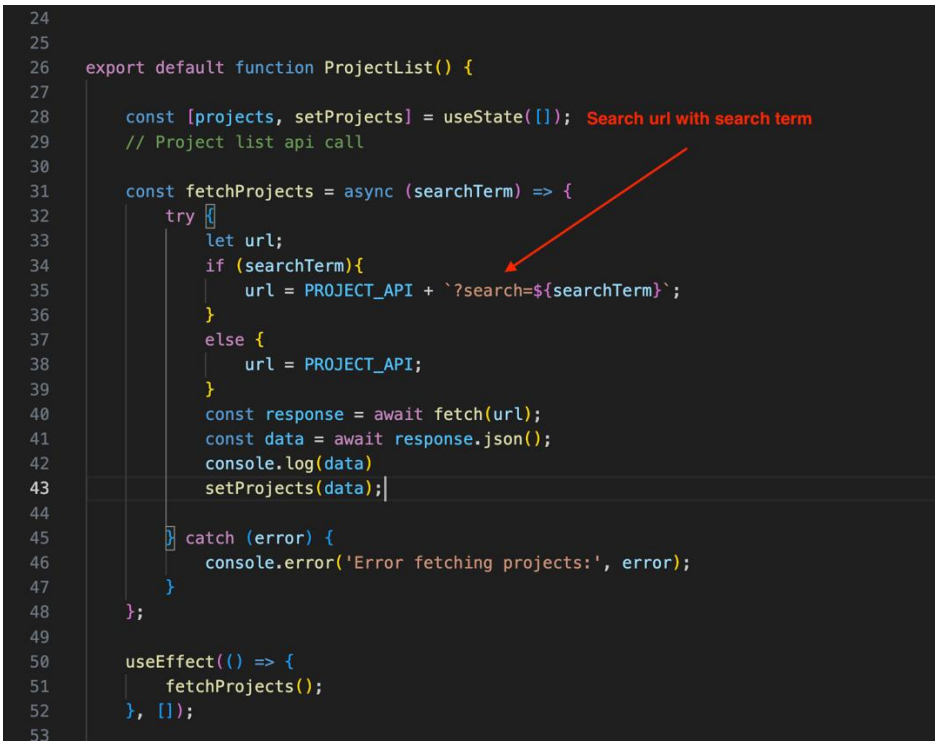


Figure 24 Knowledge Categorization UI

Search and Retrieval




```
export default function ProjectList() {

    const [projects, setProjects] = useState([]);
    // Project list api call

    const fetchProjects = async (searchTerm) => {
        try {
            let url;
            if (searchTerm){
                url = PROJECT_API + `?search=${searchTerm}`;
            }
            else {
                url = PROJECT_API;
            }
            const response = await fetch(url);
            const data = await response.json();
            console.log(data)
            setProjects(data);

        } catch (error) {
            console.error('Error fetching projects:', error);
        }
    };

    useEffect(() => {
        fetchProjects();
    }, []);
```

Figure 25 Search Implementation

The provided code constitutes a Search Component that enhances user interaction with projects, utilizing the **useState** hook for efficient state management. The component includes a **fetchProject** function, accepting **searchTerm** as a parameter, which, in turn, passes the parameter to the search API with a suffix of search. This results in the listing of project items related to the search term.

Moreover, the component incorporates the **useEffect** hook to ensure the **fetchProject** function is invoked only during the initial page load. This mechanism optimizes the performance and functionality of the Search Component by executing the search operation when the user first accesses the page.

Reward Management and Redemption

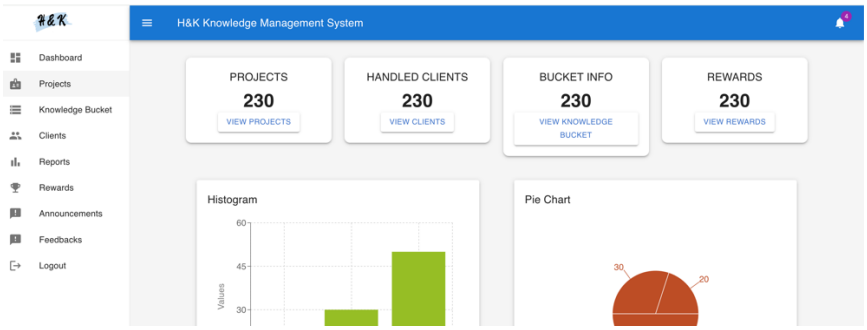


Figure 26 Reward Count UI

Announcement Management

```
21
22  const AnnouncementDisplayPage = () => {
23    return (
24      <Dashboard>
25        <div>
26          {announcements.map((announcement, index) => (
27            <Announcement key={index} {...announcement} />
28          ))}
29        </div>
30        {/* <Announcement /> */}
31      </Dashboard>
32    );
33  };
34
35  export default AnnouncementDisplayPage;
36
```

Figure 27 Announcement Listing Implementation

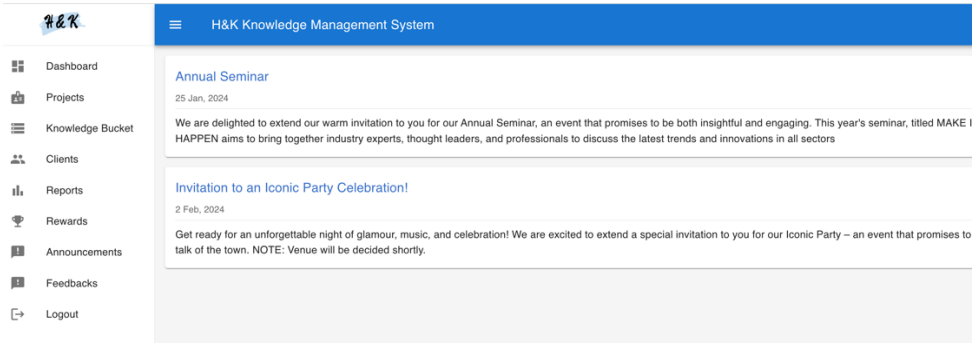


Figure 28 Announcement Listing UI

Non-Functional Requirements

Performance

Overview

The performance of the hk.net Knowledge Management System stands as a cornerstone of its design, ensuring that users worldwide, regardless of their geographical locations and varying data speeds, encounter an optimal and consistent user experience. Performance benchmarks have been meticulously established to guarantee that the system delivers unparalleled responsiveness with minimal latency.

Global Accessibility

Recognizing the diverse global footprint of Hill & Knowlton (H&K) with offices spanning numerous countries, the system is architected to transcend geographical boundaries. Whether accessed from high-speed metropolitan locations or areas with lower bandwidth availability, the design prioritizes a uniform and efficient performance.

Responsiveness Metrics

Performance benchmarks encompass a spectrum of responsiveness metrics, including but not limited to page load times, data retrieval speed, and real-time collaboration responsiveness within collaborative workspaces. These metrics are continuously monitored and fine-tuned to meet or exceed predefined performance standards.

Scalability

The system's architecture is engineered to scale seamlessly, accommodating a growing user base and an expanding volume of project-related data. Scalability ensures that performance remains robust even as usage patterns evolve, and the system undergoes increased demands over time.

Security

Security Protocols

Security is paramount in safeguarding the sensitive project information and client data entrusted to the hk.net Knowledge Management System. The implementation adheres to multi-layered security protocols, employing a defense-in-depth strategy to fortify the system against potential threats and vulnerabilities.

User Authentication

User authentication is a fundamental security measure, ensuring that only authorized individuals gain access to the system. Robust authentication mechanisms, including secure password policies and multi-factor authentication where applicable, are implemented to validate user identities.

Access Control

Access control mechanisms are in place to delineate and enforce user permissions. Role-based access control (RBAC) ensures that each user, based on their role within H&K, is granted appropriate access privileges. Fine-grained access controls enable administrators to tailor permissions with precision.

Deployment

Deployment in software development means getting a software ready for use by people, involving tasks like sharing the code, setting up configurations, and preparing servers so that the application can be used by the intended audience.

Back End Deployment

The deployment process for the Django application involved utilizing the PythonAnywhere server, a platform that provides hosting services for Python applications. This entailed

configuring and setting up the Django application on the PythonAnywhere server, ensuring that all necessary files, dependencies, and configurations were properly in place. Once deployed, the Django application became accessible and operational through the PythonAnywhere server, allowing users to interact with the application online.

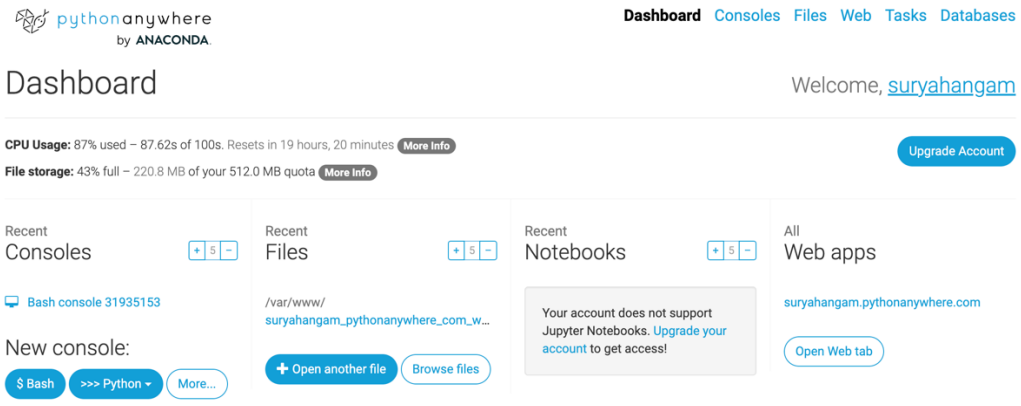


Figure 29 Back End Server Info 1

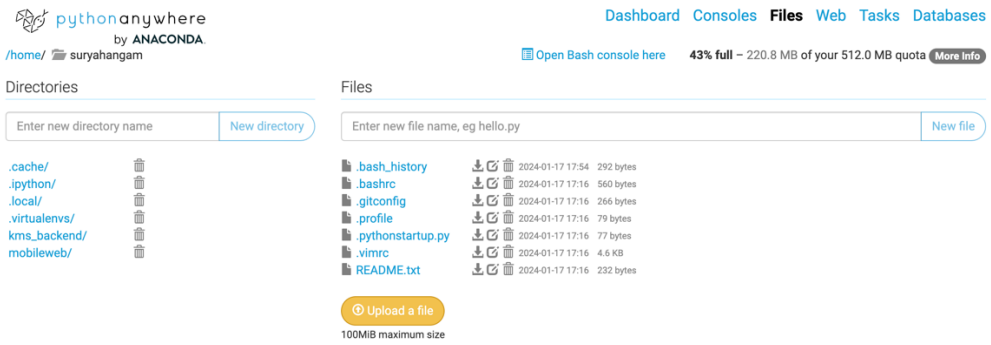


Figure 30 Back End Server Info 2

Front End Deployment

The deployment process for the front-end React components involved leveraging Netlify, a cloud platform designed for deploying and hosting web applications. This encompassed configuring

and setting up the React components on the Netlify platform, where they were transformed into a production-ready build. The deployment on Netlify ensured that the front-end components became publicly accessible on the internet, allowing users to interact with the web application seamlessly. Netlify provides features such as continuous integration and automatic deployments, simplifying the process of updating and maintaining the deployed React components.

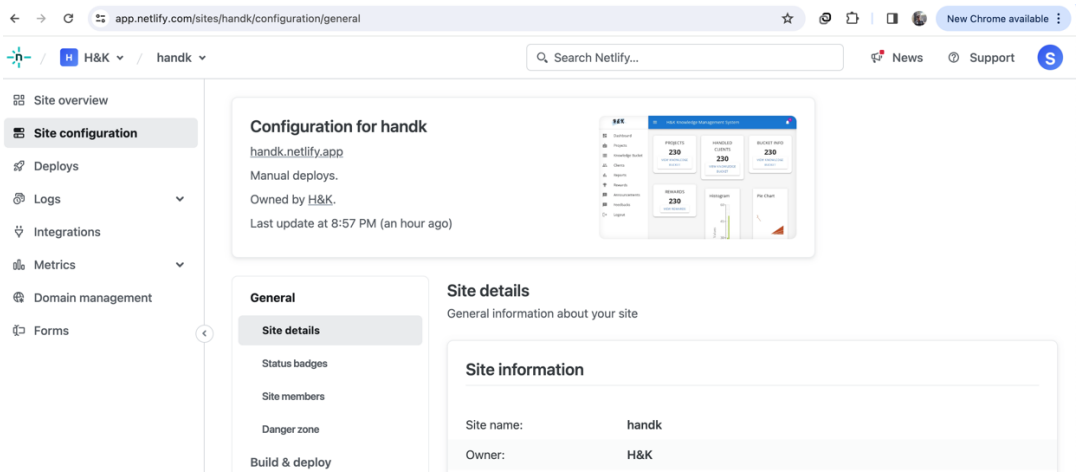


Figure 31 Front End Deployment Server Info

Conclusion

In conclusion, the deployment of the Django application on the PythonAnywhere server and the front-end React components on Netlify has successfully made both the back end and front-end of the web application accessible to users, providing a seamless and reliable online experience. This deployment ensures the efficient functioning of the application, allowing users to interact with its features and content with ease.

References

1. Williamson, E. and Lengstorf, J. (2023) *A step-by-step guide: Deploying on netlify*, *Netlify*. Available at: https://www.netlify.com/blog/2016/09/29/a-step-by-step-guide-deploying-on-netlify/ (Accessed: 17 January 2024).
2. PythonAnywere (2016) *Deploying an existing Django Project on Pythonanywhere*, *PythonAnywhere help*. Available at: https://help.pythonanywhere.com/pages/DeployExistingDjangoProject/ (Accessed: 17 January 2024).
3. 1. Cheesman, John., & Daniels, John. (2001). UML Components, A Simple Process for Specifying Component-Based Software. Pearson Education Corporation Sales Division.
4. 2. Evans, E. (2003). Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesely.
5. Morris, J. (2022) *How to create Django Models*, *DigitalOcean*. Available at: https://www.digitalocean.com/community/tutorials/how-to-create-django-models (Accessed: 17 January 2024).

Appendices

Postman Reference

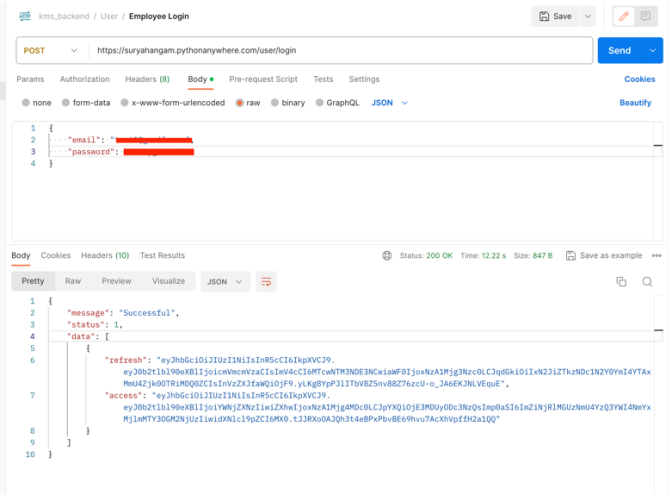


Figure 32 Postman API Testing

Backend Dependencies

```
Surya Limbu, 6 hours ago | 1 author (Surya Limbu)
asgiref==3.7.2
astroid==3.0.2
black==23.12.1
click==8.1.7
dill==0.3.7
Django==5.0
django-cors-headers==4.3.1
djangorestframework==3.14.0
djangorestframework-simplejwt==5.3.1
isort==5.13.2
mccabe==0.7.0
mypy-extensions==1.0.0
packaging==23.2
pathspec==0.12.1
Pillow==10.1.0
platformdirs==4.1.0
PyJWT==2.8.0
pylint==3.0.3
pytz==2023.3.post1
sqlparse==0.4.4
tomlkit==0.12.3
```

Figure 33 Django Dependency List