### Overview

Creating a robust, enterprise-level web application to display and manage regulatory data involves several components:

- **Backend**: Django for handling data models, business logic, and API endpoints.
- **Frontend**: React with Redux for building a dynamic user interface and managing application state.
- **User Authentication**: Secure login and session management.
- **UI/UX**: A professional landing page similar to Salesforce, featuring a feed of recent regulation summaries.
- **Workflow Management**: Tools for reviewing, approving, or rejecting incoming regulatory notifications.
- **Data Integration**: Fetching data from a web API and summarizing it using OpenAI.

---

### Backend Implementation (Django)

1. **Setup Django Project**

```
pip install django djangorestframework
django-admin startproject regulatory_app
cd regulatory_app
python manage.py startapp regulations
```

2. **Configure Database**

   - Use PostgreSQL for enterprise-level reliability.
   - Update `settings.py`:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'regulatory_db',
        'USER': 'your_db_user',
        'PASSWORD': 'your_db_password',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

3. **Install Required Packages**

```
pip install djangorestframework simplejwt django-cors-headers openai
psycopg2-binary
```

4. **Set Up User Authentication**

   - Use Django REST Framework's JWT Authentication.

```
# settings.py
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (

'rest_framework_simplejwt.authentication.JWTAuthentication',
    ),
}
```

   - Add authentication endpoints.

```
# urls.py
from django.urls import path
from rest_framework_simplejwt.views import (
    TokenObtainPairView,
    TokenRefreshView,
)

urlpatterns = [
    path('api/token/', TokenObtainPairView.as_view(),
name='token_obtain_pair'),
    path('api/token/refresh/', TokenRefreshView.as_view(),
name='token_refresh'),
]
```

5. **Define Models**

   - Create a `Regulation` model.

```
# regulations/models.py
from django.db import models

class Regulation(models.Model):
    doc_id = models.CharField(max_length=100, unique=True)
    date_posted = models.DateTimeField()
    title = models.CharField(max_length=255)
    agency = models.CharField(max_length=255)
    document_type = models.CharField(max_length=50)
    ai_summary = models.TextField()
    document_summary = models.TextField()
    status = models.CharField(
        max_length=20,
        choices=[('new', 'New'), ('approved', 'Approved'),
('rejected', 'Rejected')],
        default='new'
```

```
        )
        created_at = models.DateTimeField(auto_now_add=True)

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

- Run migrations.

```
python manage.py makemigrations
python manage.py migrate
```

6. **Create Serializers**

```
# regulations/serializers.py
from rest_framework import serializers
from .models import Regulation

class RegulationSerializer(serializers.ModelSerializer):
    class Meta:
        model = Regulation
        fields = '__all__'
```

7. **Develop Views**

```
# regulations/views.py
from rest_framework import viewsets, permissions
from .models import Regulation
from .serializers import RegulationSerializer
from django.utils import timezone
from datetime import timedelta

class RegulationViewSet(viewsets.ModelViewSet):
    queryset = Regulation.objects.all()
    serializer_class = RegulationSerializer
    permission_classes = [permissions.IsAuthenticated]

    def get_queryset(self):
        if self.action == 'list':
            last_24_hours = timezone.now() - timedelta(days=1)
            return
Regulation.objects.filter(created_at__gte=last_24_hours)
        return super().get_queryset()
```

8. **Set Up URLs**

```
# regulations/urls.py
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from .views import RegulationViewSet

router = DefaultRouter()
router.register(r'regulations', RegulationViewSet)

urlpatterns = [
    path('', include(router.urls)),
]
```

- Include these URLs in the main `urls.py`.

```
# regulatory_app/urls.py
urlpatterns = [
    path('api/', include('regulations.urls')),
    # ... other paths
]
```

9. **Integrate OpenAI for Summarization**

- Set up OpenAI API key securely (e.g., using environment variables).

```
# settings.py
import os
OPENAI_API_KEY = os.getenv('OPENAI_API_KEY')
```

- Create a management command to fetch and summarize regulations.

```
# regulations/management/commands/fetch_regulations.py
from django.core.management.base import BaseCommand
import requests
import openai
from regulations.models import Regulation

class Command(BaseCommand):
    help = 'Fetch and summarize regulations'

    def handle(self, *args, **options):
        openai.api_key = settings.OPENAI_API_KEY
        response =
requests.get('https://api.example.com/regulations')
        data = response.json()

        for item in data:
```

```
            summary = openai.Completion.create(
                engine='text-davinci-003',
                prompt=f"Summarize the following
regulation:\n{item['document_summary']}",
                max_tokens=150
            ).choices[0].text.strip()

            Regulation.objects.update_or_create(
                doc_id=item['doc_id'],
                defaults={
                    'date_posted': item['date_posted'],
                    'title': item['title'],
                    'agency': item['agency'],
                    'document_type': item['document_type'],
                    'ai_summary': summary,
                    'document_summary': item['document_summary'],
                }
            )
```

- Schedule this command using a cron job or a task scheduler like Celery.

10. **Implement Workflow Actions**

    - Add API endpoints for approving or rejecting regulations.

```
# regulations/views.py
from rest_framework.decorators import action
from rest_framework.response import Response

class RegulationViewSet(viewsets.ModelViewSet):
    # ... existing code

    @action(detail=True, methods=['post'])
    def approve(self, request, pk=None):
        regulation = self.get_object()
        regulation.status = 'approved'
        regulation.save()
        return Response({'status': 'approved'})

    @action(detail=True, methods=['post'])
    def reject(self, request, pk=None):
        regulation = self.get_object()
        regulation.status = 'rejected'
        regulation.save()
        return Response({'status': 'rejected'})
```

---

**Frontend Implementation (React with Redux)**

1. **Initialize React Project**

```
npx create-react-app regulatory_frontend
cd regulatory_frontend
```

2. **Install Dependencies**

```
npm install redux react-redux redux-thunk axios react-router-dom
@mui/material @mui/icons @emotion/react @emotion/styled
```

3. **Set Up Redux Store**

```
// src/store.js
import { createStore, applyMiddleware, combineReducers } from "redux";
import thunk from "redux-thunk";
import { regulationsReducer } from "./reducers/regulationsReducer";
import { authReducer } from "./reducers/authReducer";

const rootReducer = combineReducers({
  regulations: regulationsReducer,
  auth: authReducer,
});

const store = createStore(rootReducer, applyMiddleware(thunk));

export default store;
```

4. **Implement Authentication**

   - Create login page component.
   - Use Axios to send login credentials to the backend and store JWT tokens.

```
// src/actions/authActions.js
import axios from "axios";

export const login = (username, password) => async (dispatch) =>
{
  try {
    const response = await axios.post("/api/token/", {
      username,
      password,
    });
    localStorage.setItem("token", response.data.access);
    dispatch({ type: "LOGIN_SUCCESS", payload:
response.data.access });
  } catch (error) {
    dispatch({ type: "LOGIN_FAIL", payload: error });
```

```
    }
  };
```

5. **Create Protected Routes**

```
// src/ProtectedRoute.js
import React from "react";
import { Route, Redirect } from "react-router-dom";

const ProtectedRoute = ({ component: Component, ...rest }) => {
  const isAuthenticated = !!localStorage.getItem("token");
  return (
    <Route
      {...rest}
      render={(props) =>
        isAuthenticated ? <Component {...props} /> : <Redirect
to="/login" />
      }
    />
  );
};

export default ProtectedRoute;
```

6. **Build the Landing Page**

   - Use Material-UI components for a professional look.
   - Include a feed of regulation summaries added within the past 24 hours.

```
// src/components/Dashboard.js
import React, { useEffect } from "react";
import { useDispatch, useSelector } from "react-redux";
import { fetchRegulations } from "../actions/regulationActions";
import RegulationCard from "./RegulationCard";
import { Grid } from "@mui/material";

const Dashboard = () => {
  const dispatch = useDispatch();
  const regulations = useSelector((state) =>
state.regulations.items);

  useEffect(() => {
    dispatch(fetchRegulations());
  }, [dispatch]);

  return (
    <Grid container spacing={2}>
      {regulations.map((regulation) => (
        <Grid item xs={12} md={6} key={regulation.id}>
```

```
            <RegulationCard regulation={regulation} />
          </Grid>
        ))}
      </Grid>
    );
  };

  export default Dashboard;
```

7. **Design Regulation Cards**

```
// src/components/RegulationCard.js
import React from "react";
import { Card, CardContent, Typography, Button } from "@mui/material";

const RegulationCard = ({ regulation }) => (
  <Card>
    <CardContent>
      <Typography variant="h5">{regulation.title}</Typography>
      <Typography color="textSecondary">{regulation.agency}
</Typography>
      <Typography variant="body2">{regulation.ai_summary}</Typography>
      <Button variant="contained" color="primary">
        View Details
      </Button>
    </CardContent>
  </Card>
);

export default RegulationCard;
```

8. **Implement Review Workflow**

   - Add buttons for approving or rejecting regulations.
   - Dispatch actions to update the regulation status.

```
// src/components/RegulationDetail.js
import React from "react";
import { useDispatch } from "react-redux";
import {
  approveRegulation,
  rejectRegulation,
} from "../actions/regulationActions";
import { Button } from "@mui/material";

const RegulationDetail = ({ regulation }) => {
  const dispatch = useDispatch();

  const handleApprove = () => {
```

```
    dispatch(approveRegulation(regulation.id));
  };

  const handleReject = () => {
    dispatch(rejectRegulation(regulation.id));
  };

  return (
    <div>
      {/* Display regulation details */}
      <Button variant="contained" color="primary" onClick=
{handleApprove}>
        Approve
      </Button>
      <Button variant="contained" color="secondary" onClick=
{handleReject}>
        Reject
      </Button>
    </div>
  );
};

export default RegulationDetail;
```

9. **Set Up API Interactions**

   - Configure Axios to include the JWT token in headers.

```
// src/api.js
import axios from "axios";

const api = axios.create({
  baseURL: "http://localhost:8000",
});

api.interceptors.request.use((config) => {
  const token = localStorage.getItem("token");
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

export default api;
```

   - Create action creators for fetching and updating regulations.

```
// src/actions/regulationActions.js
import api from "../api";
```

```
export const fetchRegulations = () => async (dispatch) => {
  const response = await api.get("/api/regulations/");
  dispatch({ type: "FETCH_REGULATIONS_SUCCESS", payload:
response.data });
};

export const approveRegulation = (id) => async (dispatch) => {
  await api.post(`/api/regulations/${id}/approve/`);
  dispatch({ type: "APPROVE_REGULATION", payload: id });
};

export const rejectRegulation = (id) => async (dispatch) => {
  await api.post(`/api/regulations/${id}/reject/`);
  dispatch({ type: "REJECT_REGULATION", payload: id });
};
```

10. **Apply Styling and UX Enhancements**
    - Use Material-UI themes to match Salesforce's aesthetic.
    - Ensure responsive design for various devices.

---

**Additional Features**

- **Search and Filter**: Implement functionality to search regulations by keywords or filter by agency or date.
- **Notifications**: Add real-time notifications for new regulations using WebSockets or libraries like Socket.IO.
- **User Roles**: Define roles (e.g., Admin, Reviewer) with different permissions.

---

**Deployment Considerations**

- **Dockerization**
  - Create Dockerfiles for both backend and frontend.
  - Use Docker Compose to manage multi-container applications.
- **CI/CD Pipeline**
  - Set up automated testing and deployment using tools like Jenkins or GitHub Actions.
- **Cloud Hosting**
  - Consider deploying on platforms like AWS (with services like ECS, RDS) or Heroku.
- **Security**
  - Use HTTPS in production.
  - Implement rate limiting and input validation to prevent abuse.
  - Regularly update dependencies to patch security vulnerabilities.

---

**Sample Data Display**

Upon logging in, users will see a feed similar to:

---

**Doc ID**: CFPB-2024-0001-0023

**Date Posted**: 2024-10-04T04:00:00Z

**Title**: Debt Collection Practices (Regulation F): Deceptive and Unfair Collection of Medical Debt

**Agency**: Consumer Financial Protection Bureau

**Document Type**: Rule

**AI Summary**:

> The CFPB reminds debt collectors to comply with the FDCPA and Regulation F when collecting medical debts. Unlawful practices include collecting amounts already paid, violating federal or state laws, exceeding legal limits, collecting for services not received, misrepresenting obligations, and collecting unsubstantiated bills. Debt collectors should verify debts before collection to prevent deceptive practices.

**Document Summary**:

> The CFPB is issuing this advisory opinion to remind debt collectors of their obligation to comply with the FDCPA and Regulation F's prohibitions on false, deceptive, or misleading representations in medical debt collection.

---

**Workflow Actions**:

- **Approve**: Import the regulation into the inventory for implementation tracking.
- **Reject**: Dismiss the regulation from the feed.

---

**Testing**

- **Unit Tests**: Write tests for both frontend and backend components.
- **Integration Tests**: Ensure that the frontend and backend communicate correctly.
- **User Acceptance Testing**: Validate the workflow with end-users to gather feedback.

---

**Conclusion**

By following this guide, you'll develop an enterprise-grade web application that meets your requirements. The combination of Django and React provides a scalable and maintainable solution, while the inclusion of AI summarization adds significant value to the user experience.

---

**Next Steps**

- Begin setting up the development environment.
- Start with user authentication to secure your application from the outset.

- Implement the data models and API endpoints.
- Develop the frontend components, starting with the login page and dashboard.
- Integrate the OpenAI API for summarization.
- Test each component thoroughly before moving to the next.

---

Feel free to reach out if you need further assistance or clarification on any of these steps!