

Collaborative Text Editor Documentation

Kenisha and Ziv

December 5, 2024

Contents

1	Introduction	3
1.1	Overview	3
1.2	Features	3
1.3	Technologies Used	3
2	User Guide	4
2.1	Getting Started	4
2.1.1	Registration	4
2.1.2	Login	4
2.2	Dashboard	4
2.3	Creating a New Document	5
2.4	Editing a Document	5
2.4.1	Real-time Collaboration	5
2.4.2	Adding Comments	5
2.4.3	Version Control	5
2.4.4	Sharing Documents	5
2.5	Downloading Documents	5
3	Technical Documentation	6
3.1	Project Structure	6
3.2	Dependencies	7
3.3	ASGI Configuration	7
3.3.1	ASGI Application	7
3.3.2	Channel Layers	7
3.4	Models	8
3.4.1	Document Model	8
3.4.2	DocumentVersion Model	8
3.4.3	Comment Model	8
3.5	Views	8
3.5.1	User Registration	8
3.5.2	Document List	9
3.5.3	Document Editing	9
3.5.4	Version Control Views	9
3.6	WebSocket Consumers	10
3.6.1	DocumentConsumer	10
3.6.2	Handling Real-time Updates	10
3.7	Front-end Integration	11
3.7.1	QuillJS Editor	11
3.7.2	WebSocket Connection	11
3.8	Templates and Static Files	11
3.8.1	Base Template	11
3.8.2	Editor Template	12

3.9	Routing	12
3.9.1	URL Configuration	12
3.9.2	WebSocket Routing	12
3.10	Forms	13
3.10.1	User Registration Form	13
3.10.2	Document Form	13
3.11	Settings	13
3.11.1	Installed Apps	13
3.11.2	Middleware	14
3.11.3	Templates	14
3.11.4	Static Files	14
3.11.5	ASGI Application	14
3.12	Authentication and Authorization	14
3.13	Deployment Considerations	14
4	Appendix	15
4.1	Running the Application	15
4.2	Requirements File	15
4.3	Future Enhancements	16

Chapter 1

Introduction

1.1 Overview

This documentation provides a comprehensive guide to the development, deployment, and usage of the **Collaborative Text Editor** project. This application allows multiple users to edit documents collaboratively in real-time, incorporating features like commenting, version control, and the ability to download documents in Word format.

1.2 Features

- **Real-time Collaborative Editing:** Multiple users can edit the same document simultaneously.
- **User Authentication and Authorization:** Secure login and registration system using Django's built-in authentication.
- **Commenting System:** Users can add comments and suggest edits on specific text ranges.
- **Version Control:** Save, restore, and delete different versions of a document.
- **Download as Word Document:** Export the document in `.docx` format.

1.3 Technologies Used

- **Django:** Web framework for building the backend.
- **Django Channels:** Extends Django to handle WebSockets for real-time communication.
- **Redis:** In-memory data structure store used as the channel layer backend.
- **QuillJS:** Rich text editor integrated into the frontend.
- **Daphne:** ASGI server used to run the application.
- **JavaScript:** For frontend interactivity and WebSocket communication.
- **HTML/CSS:** For structuring and styling the web pages.

Chapter 2

User Guide

2.1 Getting Started

2.1.1 Registration

To use the Collaborative Text Editor, you need to create an account.

1. Navigate to the registration page by clicking on `register()` on the homepage.
2. Fill in the registration form:
 - **Username**
 - **Email**
 - **Password**
 - **Confirm Password**
3. Click on `initializeAccount()` to complete the registration.

2.1.2 Login

If you already have an account:

1. Click on `login()`.
2. Enter your **Username** and **Password**.
3. Click on `authenticate()` to log in.

2.2 Dashboard

Upon logging in, you are directed to your **Workspace**. Here you can:

- View all documents you have access to.
- Create a new document by clicking `new Document()`.
- Edit or delete existing documents.

2.3 Creating a New Document

1. Click on `new Document()`.
2. Provide a **Title** for your document.
3. Optionally, add **Collaborators** by selecting users from the list.
4. Click on `createDocument()` to create the document.

2.4 Editing a Document

2.4.1 Real-time Collaboration

When editing a document:

- Changes are synchronized in real-time among all collaborators.
- Active users are displayed, each with a unique cursor color.
- A rich text editor is provided for formatting and structuring your document.

2.4.2 Adding Comments

To add a comment:

1. Highlight the text you want to comment on.
2. Click on the `comment` button in the toolbar.
3. Enter your comment and optionally suggest an edit.
4. Click `Submit` to post the comment.

2.4.3 Version Control

The editor supports version control:

- Click on the `Save` button to save the current state as a new version.
- Provide a description for the version.
- Access version history via the `Version History` button.
- Preview, restore, or delete versions as needed.

2.4.4 Sharing Documents

To share a document:

1. Click on the `Share` button.
2. Add or remove collaborators.
3. Click `Save Changes` to update the collaborators.

2.5 Downloading Documents

You can download your document as a Word file:

1. Click on the `Download as Doc` button.
2. The document will be downloaded in `.docx` format.

Chapter 3

Technical Documentation

3.1 Project Structure

The project is organized as follows:

- **collaborative_text_editor/**: Root project directory containing settings and configurations.
 - `__init__.py`
 - `asgi.py`
 - `settings.py`
 - `urls.py`
 - `wsgi.py`
- **editor/**: Django app containing models, views, forms, and routing.
 - `__init__.py`
 - `admin.py`
 - `apps.py`
 - `consumers.py`
 - `forms.py`
 - `models.py`
 - `routing.py`
 - `tests.py`
 - `urls.py`
 - `views.py`
 - `migrations/`
- **templates/**: HTML templates used by the application.
- **static/**: Static files like CSS and JavaScript.
- `manage.py`: Django's command-line utility.

3.2 Dependencies

- Django \geq 3.2
- Django Channels \geq 3.0
- Channels Redis \geq 3.0
- Redis: For channel layers.
- QuillJS: Rich text editor.
- Daphne: ASGI server.
- python-docx: For exporting documents to Word format.

3.3 ASGI Configuration

3.3.1 ASGI Application

The `asgi.py` file configures the ASGI application for handling both HTTP and WebSocket protocols. It sets up Django and wraps the HTTP application with `ASGIStaticFilesHandler` to serve static files during development.

```
1 import os
2
3 os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'collaborative_text_editor.settings')
4
5 import django
6 django.setup()
7
8 from channels.auth import AuthMiddlewareStack
9 from channels.routing import ProtocolTypeRouter, URLRouter
10 from django.core.asgi import get_asgi_application
11 from django.contrib.staticfiles.handlers import ASGIStaticFilesHandler
12 import editor.routing
13
14 application = ProtocolTypeRouter({
15     "http": ASGIStaticFilesHandler(get_asgi_application()),
16     "websocket": AuthMiddlewareStack(
17         URLRouter(
18             editor.routing.websocket_urlpatterns
19         )
20     ),
21 })
```

Listing 3.1: `collaborative_text_editor/asgi.py`

3.3.2 Channel Layers

Configured in `settings.py` to use Redis as the backend for channel layers, which enables message passing between different instances of the application.

```
1 CHANNEL_LAYERS = {
2     'default': {
3         'BACKEND': 'channels_redis.core.RedisChannelLayer',
4         'CONFIG': {
5             'hosts': [('localhost', 6379)],
6             'capacity': 10000,
7             'group_expiry': 60 * 60,
8         },
9     },
10 }
```


Listing 3.2: collaborative_text_editor/settings.py

3.4 Models

3.4.1 Document Model

Represents a collaborative document with fields for title, content, collaborators, and timestamps.

```
1 class Document(models.Model):
2     title = models.CharField(max_length=255)
3     content = models.TextField(blank=True, default='') # Store content as JSON
4     collaborators = models.ManyToManyField(User, related_name='documents')
5     created_at = models.DateTimeField(auto_now_add=True)
6     updated_at = models.DateTimeField(auto_now=True)
```

Listing 3.3: editor/models.py

3.4.2 DocumentVersion Model

Stores versions of a document for version control, including content, description, timestamps, and the user who saved the version.

```
1 class DocumentVersion(models.Model):
2     document = models.ForeignKey(Document, on_delete=models.CASCADE, related_name='versions')
3     content = models.TextField()
4     description = models.CharField(max_length=255, blank=True)
5     created_at = models.DateTimeField(auto_now_add=True)
6     user = models.ForeignKey(User, on_delete=models.SET_NULL, null=True)
```

3.4.3 Comment Model

Represents comments made on specific text ranges within a document. Supports suggesting edits and tracking whether the comment has been resolved or the edit has been applied.

```
1 class Comment(models.Model):
2     document = models.ForeignKey(Document, on_delete=models.CASCADE, related_name='comments')
3     user = models.ForeignKey(User, on_delete=models.CASCADE)
4     range = models.JSONField()
5     content = models.TextField()
6     suggested_edit = models.TextField(null=True, blank=True)
7     created_at = models.DateTimeField(auto_now_add=True)
8     resolved = models.BooleanField(default=False)
9     edit_applied = models.BooleanField(default=False)
```

3.5 Views

3.5.1 User Registration

Handles user registration using a custom RegisterForm.

```
1 def register(request):
2     if request.method == 'POST':
3         form = RegisterForm(request.POST)
4         if form.is_valid():
5             user = form.save()
6             login(request, user)
7             return redirect('document_list')
8     else:
```

```

9     form = RegisterForm()
10     return render(request, 'register.html', {'form': form})

```

Listing 3.4: editor/views.py

3.5.2 Document List

Displays the list of documents the user has access to, filtering by the user's collaborations.

```

1 @login_required
2 def document_list(request):
3     documents = Document.objects.filter(collaborators=request.user)
4     return render(request, 'document_list.html', {'documents': documents})

```

3.5.3 Document Editing

Handles document editing, including updating the title via AJAX, updating collaborators, and rendering the editor template.

```

1 @csrf_exempt
2 @login_required
3 def document_edit(request, pk):
4     document = get_object_or_404(Document, pk=pk, collaborators=request.user)
5     if request.method == 'POST':
6         if request.content_type == 'application/json':
7             data = json.loads(request.body)
8             new_title = data.get('title')
9             if new_title:
10                 document.title = new_title
11                 document.save()
12                 return JsonResponse({'status': 'success'})
13             else:
14                 return HttpResponseBadRequest('Invalid title')
15         elif 'form_type' in request.POST and request.POST.get('form_type') == 'collaborators_form':
16             collaborators_form = CollaboratorsForm(request.POST, instance=document)
17             if collaborators_form.is_valid():
18                 collaborators_form.save()
19                 return redirect('document_edit', pk=document.pk)
20             else:
21                 print(collaborators_form.errors)
22         else:
23             return HttpResponseBadRequest('Invalid form submission')
24     else:
25         collaborators_form = CollaboratorsForm(instance=document)
26     return render(request, 'editor.html', {'document': document, 'collaborators_form': collaborators_form})

```

3.5.4 Version Control Views

Includes saving, listing, restoring, and deleting document versions.

```

1 @csrf_exempt
2 @login_required
3 def save_version(request, pk):
4     if request.method == 'POST':
5         document = get_object_or_404(Document, pk=pk, collaborators=request.user)
6         data = json.loads(request.body)
7         content = data.get('content')
8         description = data.get('description', '')
9         if content:
10             DocumentVersion.objects.create(
11                 document=document,

```

```

12         content=json.dumps(content),
13         description=description,
14         user=request.user
15     )
16     return JsonResponse({'status': 'success'})
17 else:
18     return HttpResponseBadRequest('No content provided.')
19 else:
20     return HttpResponseBadRequest('Invalid request method.')

```

3.6 WebSocket Consumers

3.6.1 DocumentConsumer

Manages real-time document editing, handling WebSocket connections, broadcasting changes, and managing active users.

```

1 class DocumentConsumer(AsyncWebsocketConsumer):
2     active_users_dict = {}
3
4     async def connect(self):
5         self.document_id = self.scope['url_route']['kwargs']['document_id']
6         self.room_group_name = f'document_{self.document_id}'
7         self.user = self.scope["user"]
8         self.color = await self.get_user_color(self.user.id)
9
10        if self.user.is_anonymous:
11            await self.close()
12        else:
13            has_permission = await self.check_permission()
14            if not has_permission:
15                await self.close()
16            else:
17                await self.add_active_user()
18                await self.channel_layer.group_add(
19                    self.room_group_name,
20                    self.channel_name
21                )
22                await self.accept()
23                document = await self.get_document()
24                content = json.loads(document.content) if document.content else {'ops': []}
25                await self.send(text_data=json.dumps({
26                    'type': 'init',
27                    'content': content
28                }))
29                await self.broadcast_active_users()

```

Listing 3.5: editor/consumers.py

3.6.2 Handling Real-time Updates

Updates to the document are broadcast to all connected clients, ensuring synchronization across sessions.

```

1 async def receive(self, text_data):
2     data = json.loads(text_data)
3     message_type = data.get('type')
4
5     if message_type == 'delta':
6         delta_ops = data.get('delta')
7         if delta_ops:
8             await self.update_document_content(delta_ops)
9             await self.channel_layer.group_send(
10                 self.room_group_name,
11                 {

```

```
12         'type': 'broadcast_delta',
13         'delta': delta_ops,
14         'sender_channel_name': self.channel_name
15     })
```

3.7 Front-end Integration

3.7.1 QuillJS Editor

The QuillJS editor is initialized with real-time collaboration features, including cursor synchronization and commenting functionality.

```
1 Quill.register('modules/cursors', window.QuillCursors);
2
3 const quill = new Quill('#editor-container', {
4     theme: 'snow',
5     modules: {
6         cursors: true,
7         toolbar: {
8             container: [
9                 // Toolbar options
10            ],
11            handlers: {
12                'comment': function() {
13                    // Comment handler
14                }
15            }
16        }
17    }
18 });
```

Listing 3.6: Initializing Quill Editor

3.7.2 WebSocket Connection

Establishes a WebSocket connection for real-time collaboration, handling messages for content updates, cursor positions, and comments.

```
1 const wsScheme = window.location.protocol === "https:" ? "wss" : "ws";
2 const documentId = "{ document.pk }";
3 const socketUrl = `${wsScheme}://${window.location.host}/ws/document/${documentId}/`;
4 const socket = new WebSocket(socketUrl);
5
6 socket.onmessage = function(e) {
7     const data = JSON.parse(e.data);
8     if (data.type === 'delta') {
9         // Apply delta to the editor
10    }
11};
```

Listing 3.7: WebSocket Connection

3.8 Templates and Static Files

3.8.1 Base Template

Defines the base HTML structure, including the header, footer, and main content blocks.

```
1 {% load static %}
2 <!DOCTYPE html>
3 <html lang="en">
```

```
4 <head>
5   <meta charset="UTF-8">
6   <title>CollabDocs</title>
7   <link rel="stylesheet" href="{% static 'css/styles.css' %}">
8     {% block extra_head %}{% endblock %}
9 </head>
10 <body>
11   {% block content %}{% endblock %}
12   {% block extra_script %}{% endblock %}
13 </body>
14 </html>
```

Listing 3.8: templates/base.html

3.8.2 Editor Template

The template for the document editor page, including the QuillJS editor and modals for version control and sharing.

```
1 {% extends 'base.html' %}
2
3 {% block extra_head %}
4 <link href="https://cdn.quilljs.com/1.3.6/quill.snow.css" rel="stylesheet">
5 {% endblock %}
6
7 {% block content %}
8 <div id="editor-container"></div>
9 <!-- Additional modals and components -->
10 {% endblock %}
11
12 {% block extra_script %}
13 <script src="https://cdn.quilljs.com/1.3.6/quill.js"></script>
14 <script>
15 // JavaScript code to initialize Quill and WebSocket
16 </script>
17 {% endblock %}
```

Listing 3.9: templates/editor.html

3.9 Routing

3.9.1 URL Configuration

Defines URL patterns for views, including authentication, document operations, and version control.

```
1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5     path('register/', views.register, name='register'),
6     path('', views.document_list, name='document_list'),
7     path('document/new/', views.document_create, name='document_create'),
8     path('document/<int:pk>/edit/', views.document_edit, name='document_edit'),
9     # Additional URL patterns
10 ]
```

Listing 3.10: editor/urls.py

3.9.2 WebSocket Routing

Configures WebSocket routes for real-time communication using Django Channels.

```
1 from django.urls import re_path
2 from . import consumers
3
4 websocket_urlpatterns = [
5     re_path(r'^ws/document/(?P<document_id>\d+)/$', consumers.DocumentConsumer.as_asgi()),
6 ]
```

Listing 3.11: editor/routing.py

3.10 Forms

3.10.1 User Registration Form

Extends Django's `UserCreationForm` to include an email field.

```
1 class RegisterForm(UserCreationForm):
2     email = forms.EmailField(required=True)
3
4     class Meta:
5         model = User
6         fields = ['username', 'email', 'password1', 'password2']
```

Listing 3.12: editor/forms.py

3.10.2 Document Form

Used for creating new documents, allowing the selection of collaborators.

```
1 class DocumentForm(forms.ModelForm):
2     collaborators = forms.ModelMultipleChoiceField(
3         queryset=User.objects.all(),
4         widget=forms.CheckboxSelectMultiple,
5         required=False,
6         label="Share with"
7     )
8
9     class Meta:
10         model = Document
11         fields = ['title', 'collaborators']
```

3.11 Settings

Key settings in `settings.py`:

3.11.1 Installed Apps

Includes necessary Django apps and third-party apps like Channels.

```
1 INSTALLED_APPS = [
2     'channels',          # For WebSocket support
3     'django.contrib.admin',
4     'django.contrib.auth',
5     # ...
6     'editor',           # Your app
7 ]
```

3.11.2 Middleware

Standard Django middleware components.

```
1 MIDDLEWARE = [  
2     'django.middleware.security.SecurityMiddleware',  
3     'django.contrib.sessions.middleware.SessionMiddleware',  
4     # ...  
5 ]
```

3.11.3 Templates

Configure template directories and context processors.

```
1 TEMPLATES = [  
2     {  
3         'BACKEND': 'django.template.backends.django.DjangoTemplates',  
4         'DIRS': [os.path.join(BASE_DIR, 'templates')],  
5         # ...  
6     },  
7 ]
```

3.11.4 Static Files

Configure static files directories.

```
1 STATIC_URL = '/static/'  
2 STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]
```

3.11.5 ASGI Application

Specify the ASGI application for Channels.

```
1 ASGI_APPLICATION = 'collaborative_text_editor.asgi.application'
```

3.12 Authentication and Authorization

- Uses Django's built-in authentication system.
- Views are protected using `@login_required`.
- Document access is restricted to collaborators via query filters.
- The `CollaboratorsForm` allows updating collaborators for a document.

3.13 Deployment Considerations

- Ensure Redis is running for channel layers.
- Use an ASGI server like **Daphne** for deployment.
- Configure `ALLOWED_HOSTS` and security settings in `settings.py`.
- Collect static files using `python manage.py collectstatic` for production.
- For running the server, use the following command:

```
1 daphne -b 0.0.0.0 -p 8000 collaborative_text_editor.asgi:application  
2
```

Chapter 4

Appendix

4.1 Running the Application

1. Install dependencies using pip:

```
1 pip install -r requirements.txt
2
```

2. Start the Redis server:

```
1 redis-server
2
```

3. Apply database migrations:

```
1 python manage.py migrate
2
```

4. Create a superuser (optional):

```
1 python manage.py createsuperuser
2
```

5. Collect static files (for production):

```
1 python manage.py collectstatic
2
```

6. Run the application using Daphne:

```
1 daphne -b 0.0.0.0 -p 8000 collaborative_text_editor.asgi:application
2
```

7. Access the application at `http://localhost:8000`

4.2 Requirements File

An example `requirements.txt`:

```
1 Django>=3.2
2 channels>=3.0
3 channels_redis>=3.0
4 redis>=3.5
5 python-docx>=0.8
```


4.3 Future Enhancements

- Implement more granular permissions for collaborators (e.g., read-only access).
- Enhance the commenting system with email notifications.
- Optimize performance for large documents by implementing pagination or virtual scrolling.
- Add support for real-time editing of images and other media types.
- Integrate user avatars and profiles for a more personalized experience.