Full Stack Developer Task

Overview

Landing page Visitors to the app should be able to view a livestream video played from an RTSP URL (use RTSP me or RTSP stream to create a temporary stream from a video over internet). The video should be embedded on the landing page, and users can click play to start watching the livestream.

App Playing Livestream

The app should be able to play a livestream video from a given RTSP URL. Users should have basic controls such as play, pause, and volume adjustment.

1.Overlay Options: Users should have the option to add custom overlays (such as logos and text) on top of the livestream. These overlays can be positioned and resized as needed.

2.CRUD API for Overlays and Settings:

- Create: Users should be able to create and save custom overlay settings, including position, size, and content.
- Read: Users should be able to retrieve their saved overlay settings.
- Update: Users should be able to modify existing overlay settings.
- Delete: Users should be able to delete saved overlay settings.

Tech Stack

- Python (Flask Preferred)
- MongoDB
- React
- Video Streaming compatible with RTSP

Working of HLS Streaming + Overlay Manager Project

-What does the project do?

- Displays an **HLS livestream video** in a browser.
- Lets you create **text or image overlays** on top of the video.
- You can **add, edit, delete, drag, and resize** these overlays.
- Overlays are saved in a **MongoDB database**.
- Images can be uploaded to the backend and displayed as overlays.
- The video and overlays are fully interactive and persist between sessions.

-Project Structure

1. **Backend:** Flask API + MongoDB
2. **Frontend:** React app with HLS.js + overlay UI
3. **Uploads folder:** stores uploaded images for overlays.

-Step 1: Backend (Flask + MongoDB)

- Hosts REST APIs for managing overlays (Create, Read, Update, Delete).
- Receives image uploads, stores files, and returns public URLs.
- Connects to MongoDB to persist overlay data.
- Serves uploaded images for frontend to use.
- Runs on http://127.0.0.1:5000 by default.

Key libraries:

- Flask — Python web framework
- Flask-PyMongo — to interact with MongoDB
- Flask-CORS — to enable cross-origin requests from frontend
- Werkzeug — for safely saving uploaded files
- bson ObjectId for MongoDB document ids

MongoDB:

- Stores overlays with fields: type, content, x, y, width, height, createdAt
- Each overlay has a unique _id.

Backend Setup

1. Create virtual environment:
2. python -m venv venv
3. source venv/bin/activate # Linux/Mac
4. venv\Scripts\activate   # Windows
5. Install dependencies:
6. pip install flask flask-pymongo flask-cors werkzeug
7. Start MongoDB locally (ensure it is running).
8. Run backend:
9. python app.py
    - Listens on http://127.0.0.1:5000

Backend APIs

Endpoint	**Method**	**Description**
/api/overlays	GET	List all overlays
/api/overlays	POST	Create a new overlay
/api/overlays/<id>	PUT	Update overlay with given id
/api/overlays/<id>	DELETE	Delete overlay with given id
/api/upload	POST	Upload an image and return its URL
/uploads/<filename>	GET	Serve uploaded image files

Backend Workflow

1. When you **create** or **update** an overlay, frontend sends data (type, content, x, y, width, height) as JSON to the backend.
2. Backend saves the overlay in MongoDB.
3. When frontend loads or refreshes, it **fetches all overlays** from backend to render on video.
4. When you upload an image, frontend sends the file to /api/upload.
5. Backend saves the image in uploads/ folder and returns a URL.
6. Frontend uses that URL to show the image overlay.

Step 2: Frontend (React + HLS.js + react-rnd)

- Renders the HLS livestream video.
- Fetches overlays from backend and draws them as draggable, resizable elements over video.
- Provides a form to create or edit overlays.
- Allows uploading images as overlays.
- Controls video screen size.
- Sends overlay data to backend to save changes.

Key frontend libraries:

- **React** — UI framework.
- **HLS.js** — play HLS streams in browsers without native support.
- **axios** — for HTTP requests to backend.
- **react-rnd** — for draggable and resizable overlays.

Frontend setup

1. Create React app : npx create-react-app@latest frontend
2. cd frontend

3. Install dependencies: npm install axios hls.js react-rnd
4. Replace src/App.jsx content with the provided React code.
5. Start frontend: npm start
6. Visit http://localhost:3000 to see the app.

Frontend Workflow

1. User enters HLS stream URL.
2. HLS.js loads the video and plays in <video> element.
3. On page load, frontend fetches overlays from backend (GET /api/overlays).
4. Overlays are drawn as draggable, resizable boxes using react-rnd.
5. User can add a new overlay via form:
   - Select "text" or "image".
   - For images, upload a file → frontend uploads to backend and gets a URL.
   - Specify position and size.
   - Click **Create** → sends POST to backend to save overlay.
6. User can click an existing overlay’s **Edit** button to load it into the form, modify and **Update** it.
7. User can **Delete** overlays.
8. Moving or resizing overlays updates their position/size on screen.
9. On submit of create/update, frontend tells backend to save changes.

Visual layers:

- Video is the base layer.
- Overlays (text or images) are absolutely positioned divs/images on top.
- React-rnd lets user drag/resize overlays.
- Changes to overlays trigger API calls to save to backend.

-How video streaming works

- You provide an HLS URL (.m3u8 playlist).
- HLS.js loads this URL, parses the playlist, downloads video segments, and feeds them to the browser's video player.
- Your video plays live or on-demand depending on the stream.
- You can resize the video display with the screen size selector.
- Overlays are drawn on top using absolute positioning.

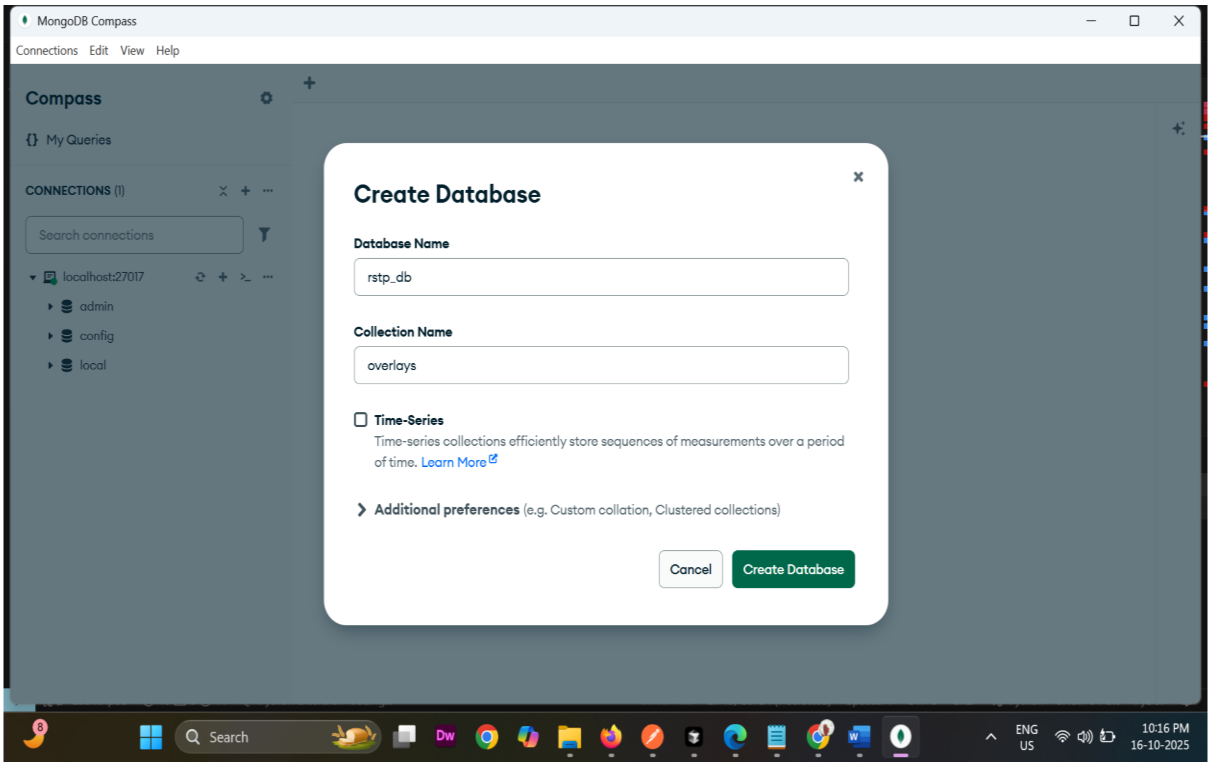
-How overlays works

- Overlays are stored in MongoDB.
- Each overlay has:
  - type: "text" or "image"
  - content: text string or image URL
  - x, y: position relative to video container
  - width, height: size of overlay box
- Frontend renders overlays using react-rnd inside the video container.
- User interactions (drag/resize) update the overlay properties locally and can be saved to backend.

-Image Uploading Flow

1. User chooses an image file in the overlay form.
2. Frontend sends the file to /api/upload via multipart form data.
3. Backend saves file in /uploads folder.
4. Backend responds with URL to access the image.
5. Frontend sets overlay content to this image URL and displays it.

1.Create Database using MongoDB



Create Database

Database Name: rstp_db

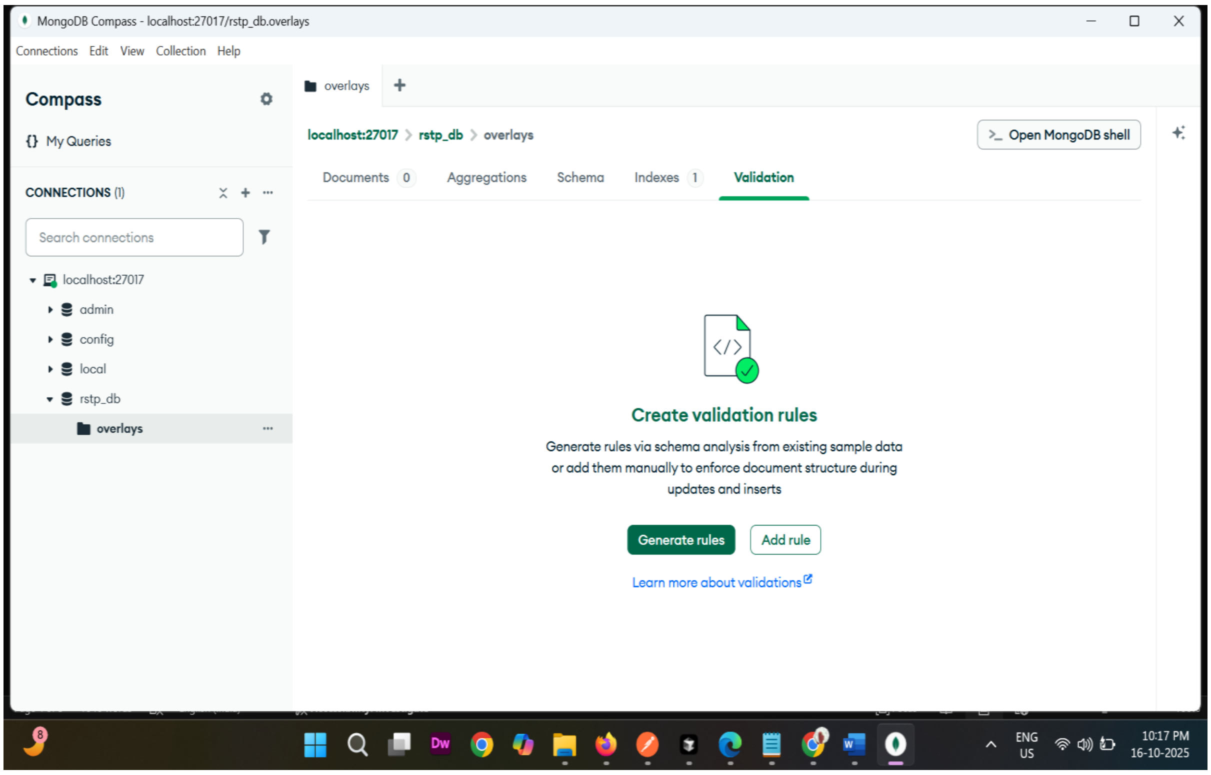
Collection Name: overlays

☐ Time-Series
Time-series collections efficiently store sequences of measurements over a period of time. Learn More

Additional preferences (e.g. Custom collation, Clustered collections)

Cancel | Create Database

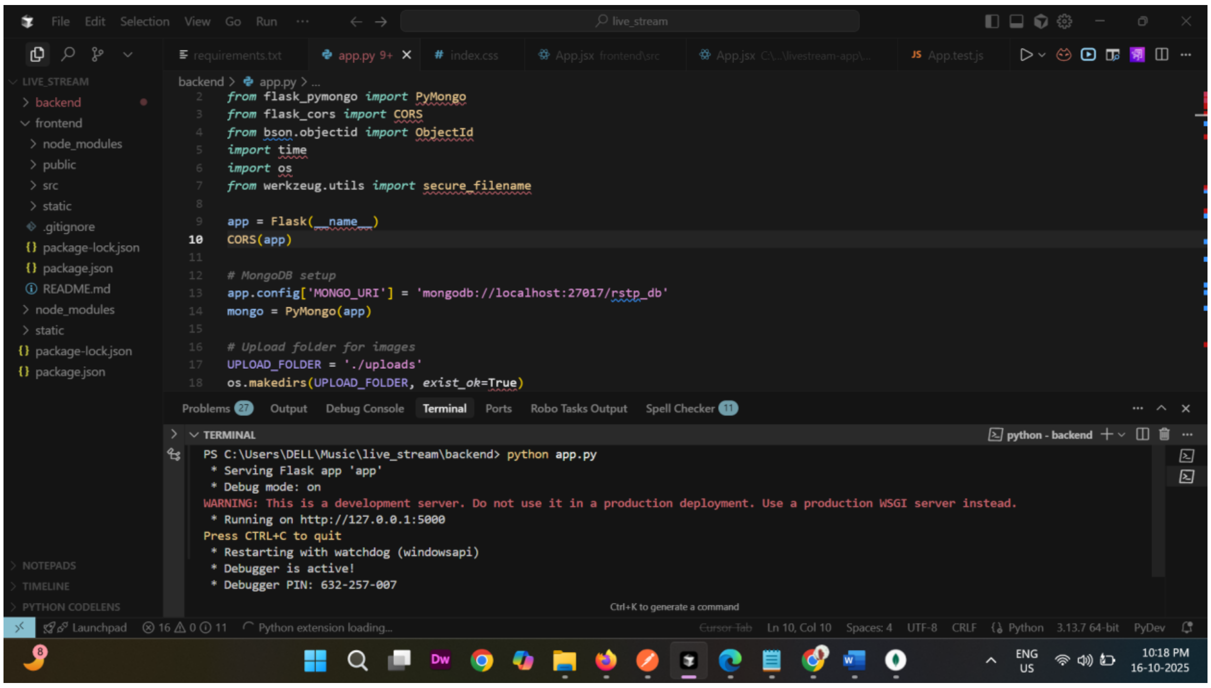
2.Database with rstp_db created with no entries



Create validation rules

Generate rules via schema analysis from existing sample data or add them manually to enforce document structure during updates and inserts

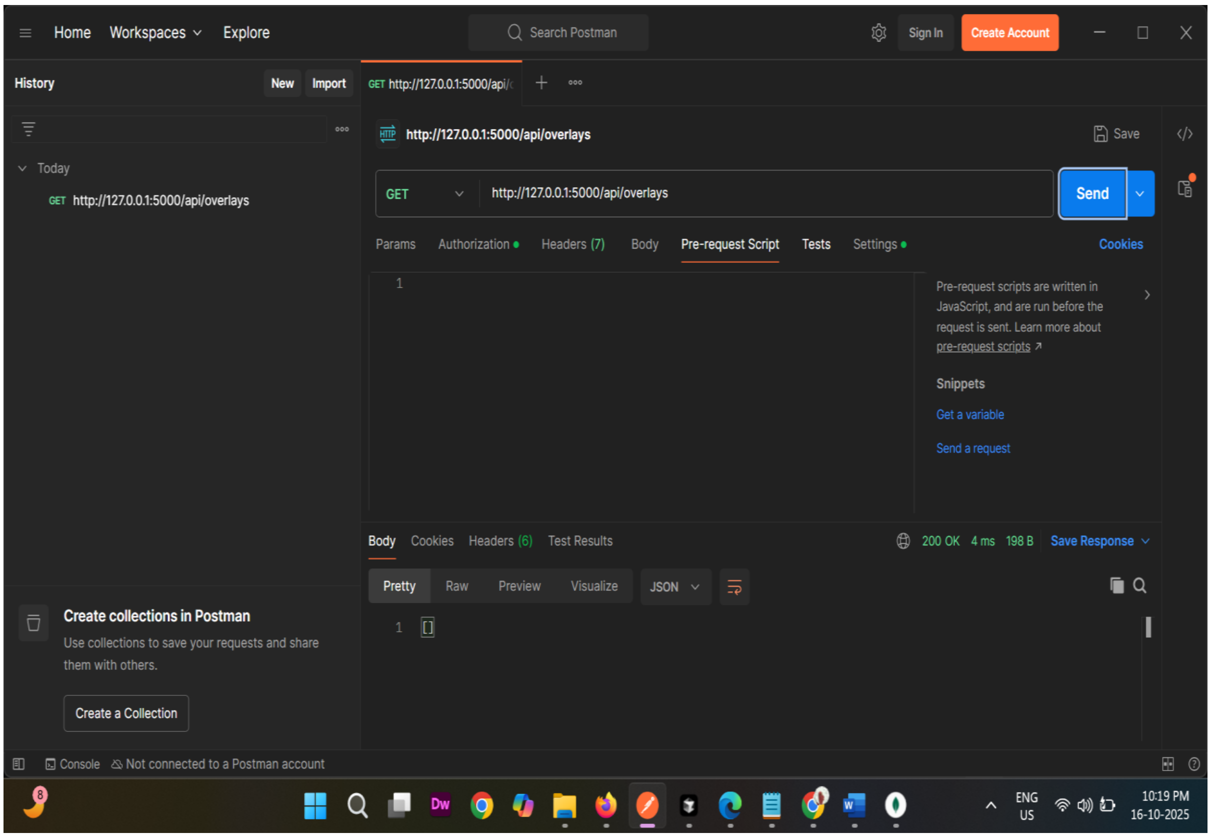
3.Run the backend App.py



```
from flask_pymongo import PyMongo
from flask_cors import CORS
from bson.objectid import ObjectId
import time
import os
from werkzeug.utils import secure_filename

app = Flask(__name__)
CORS(app)

# MongoDB setup
app.config['MONGO_URI'] = 'mongodb://localhost:27017/rstp_db'
mongo = PyMongo(app)

# Upload folder for images
UPLOAD_FOLDER = './uploads'
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
```

```
PS C:\Users\DELL\Music\live_stream\backend> python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 632-257-007
```

4.Checking with GET API with Postman no entries found

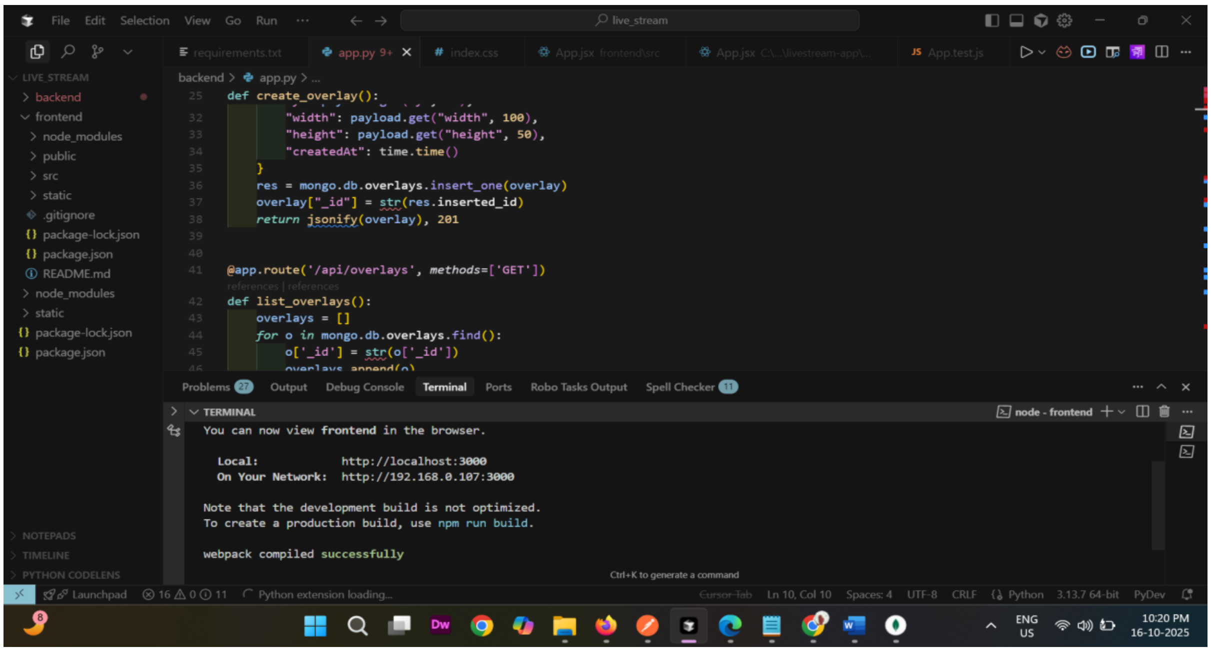


```
GET http://127.0.0.1:5000/api/overlays
```

200 OK 4 ms 198 B

```
[]
```

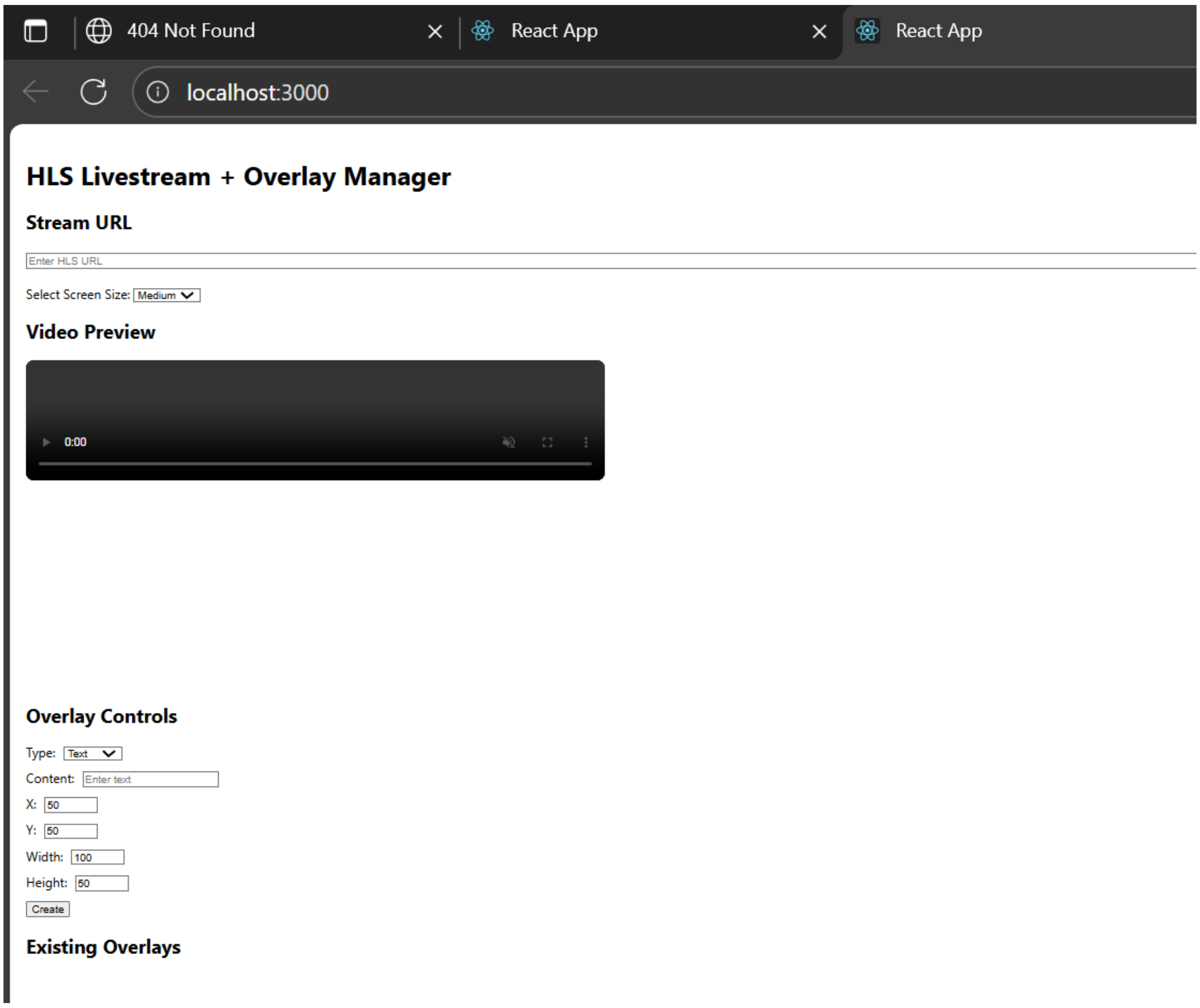
5.Starting the frontend React-app server



```
def create_overlay():
        "width": payload.get("width", 100),
        "height": payload.get("height", 50),
        "createdAt": time.time()
    }
    res = mongo.db.overlays.insert_one(overlay)
    overlay["_id"] = str(res.inserted_id)
    return jsonify(overlay), 201


@app.route('/api/overlays', methods=['GET'])
def list_overlays():
    overlays = []
    for o in mongo.db.overlays.find():
        o['_id'] = str(o['_id'])
        overlays.append(o)
```

```
You can now view frontend in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.0.107:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

6.React App works on localhost:3000



HLS Livestream + Overlay Manager

Stream URL

Enter HLS URL

Select Screen Size: Medium

Video Preview

Overlay Controls

Type: Text

Content: Enter text

X: 50

Y: 50

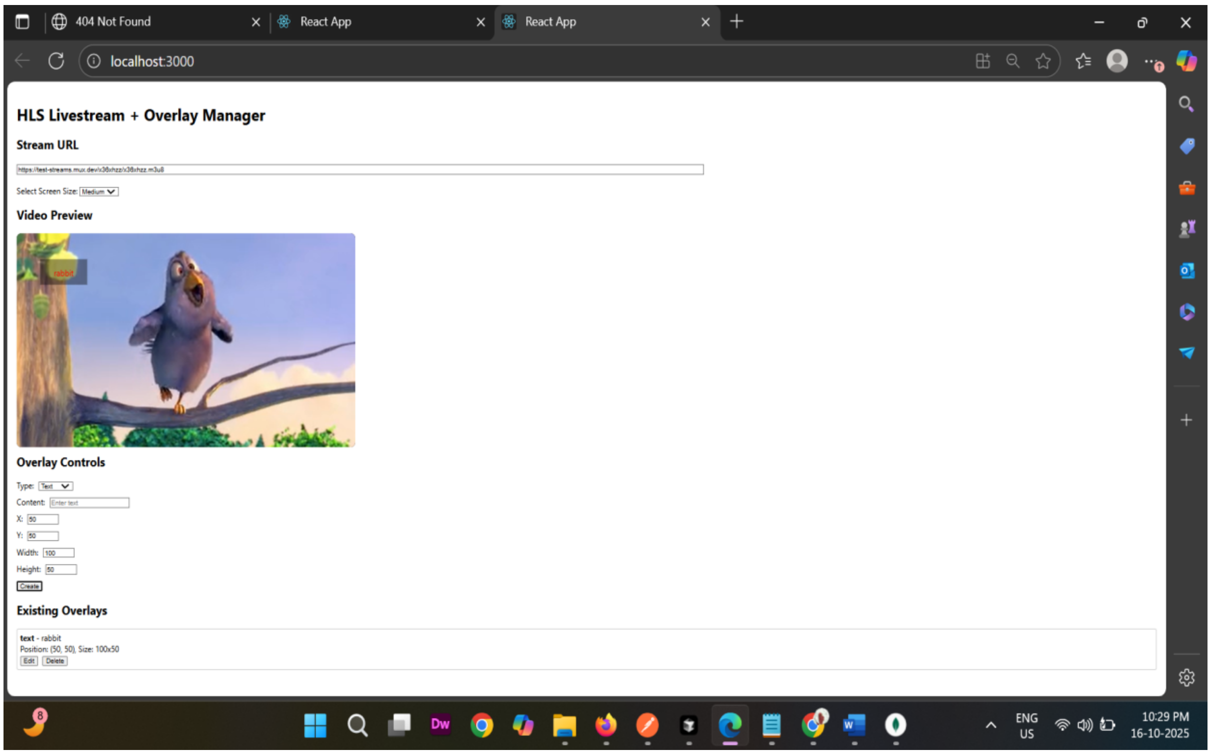
Width: 100

Height: 50

Create

Existing Overlays

7.I have entered Url and created overlay



HLS Livestream + Overlay Manager

Stream URL

https://test-streams.mux.dev/x36xhzz/x36xhzz.m3u8

Select Screen Size: Medium

Video Preview

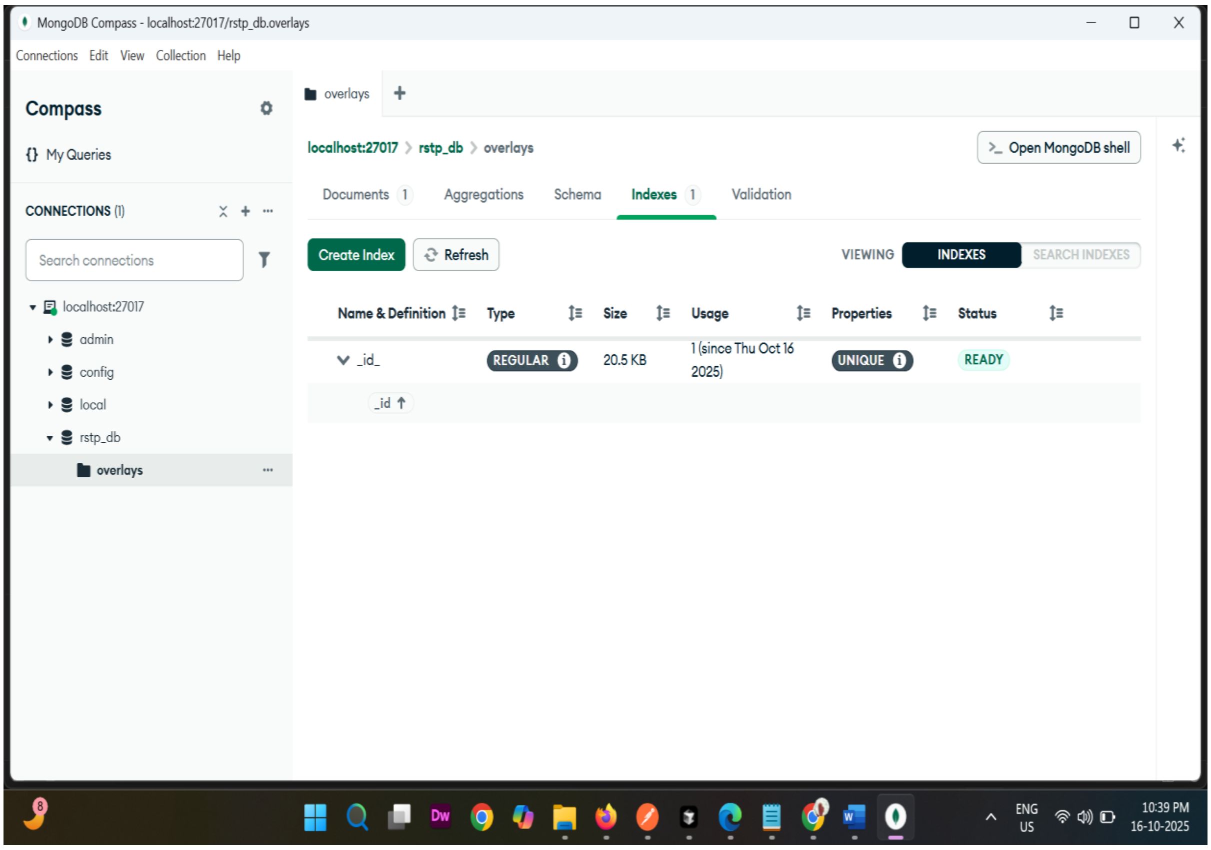
Overlay Controls

Type: Text
Content: Enter text
X: 50
Y: 50
Width: 100
Height: 50
Create

Existing Overlays

text - rabbit
Position: (50, 50), Size: 100x50
Edit Delete

9.One index created in database



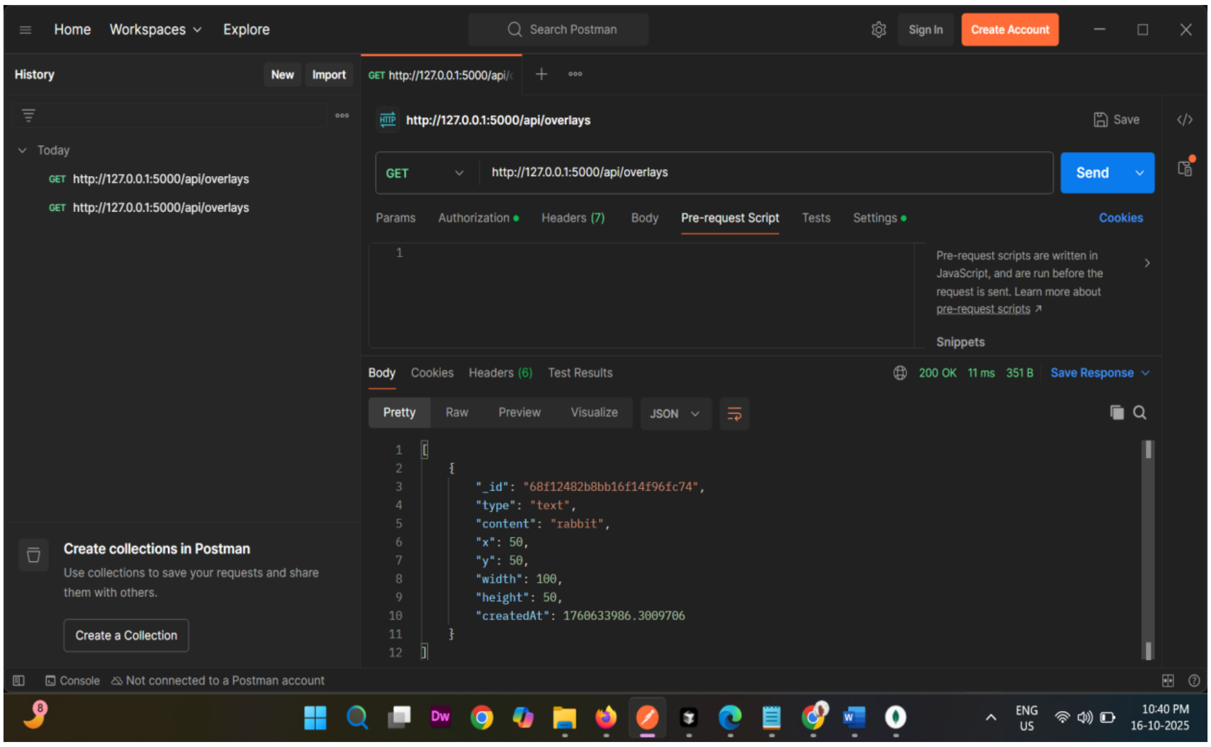
MongoDB Compass - localhost:27017/rstp_db.overlays

localhost:27017 > rstp_db > overlays

Documents 1 | Aggregations | Schema | Indexes 1 | Validation

Name & Definition	Type	Size	Usage	Properties	Status
_id_	REGULAR	20.5 KB	1 (since Thu Oct 16 2025)	UNIQUE	READY
_id ↑					

10.Now I have tested GET request from postman



http://127.0.0.1:5000/api/overlays

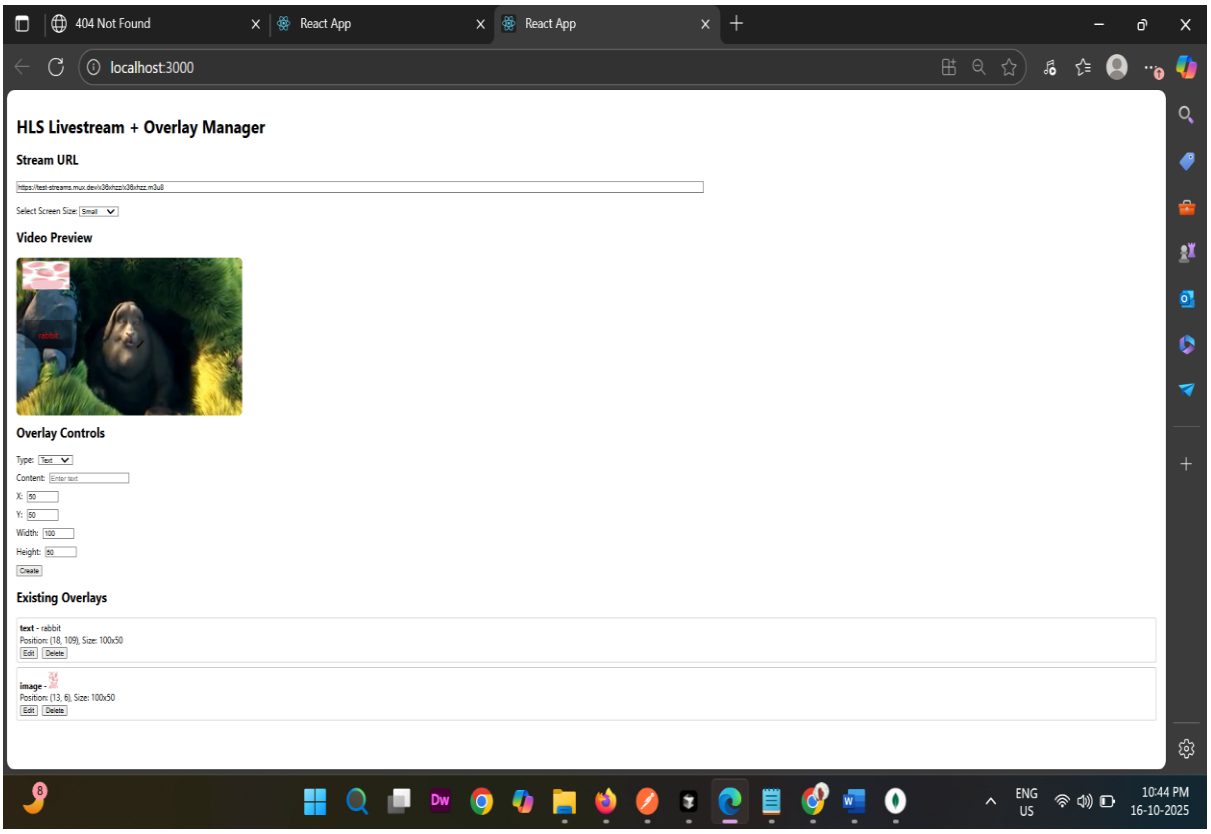
GET http://127.0.0.1:5000/api/overlays

200 OK 11 ms 351 B

```
[
    {
        "_id": "68f12482b8bb16f14f96fc74",
        "type": "text",
        "content": "rabbit",
        "x": 50,
        "y": 50,
        "width": 100,
        "height": 50,
        "createdAt": 1760633986.3009706
    }
]
```

Home
Workspaces
Explore
Search Postman
Sign In
Create Account
History
New
Import
Today
GET http://127.0.0.1:5000/api/overlays
GET http://127.0.0.1:5000/api/overlays
Save
Send
Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings
Cookies
Pre-request scripts are written in JavaScript, and are run before the request is sent. Learn more about pre-request scripts
Snippets
Body
Cookies
Headers (6)
Test Results
Save Response
Pretty
Raw
Preview
Visualize
JSON
Create collections in Postman
Use collections to save your requests and share them with others.
Create a Collection
Console
Not connected to a Postman account
ENG US
10:40 PM
16-10-2025

11.I have inserted logo in livestream



HLS Livestream + Overlay Manager

Stream URL

https://test-streams.mux.dev/x36xhzz/x36xhzz.m3u8

Select Screen Size: Small

Video Preview

Overlay Controls

Type: Text

Content: Enter text

X: 50

Y: 50

Width: 100

Height: 50

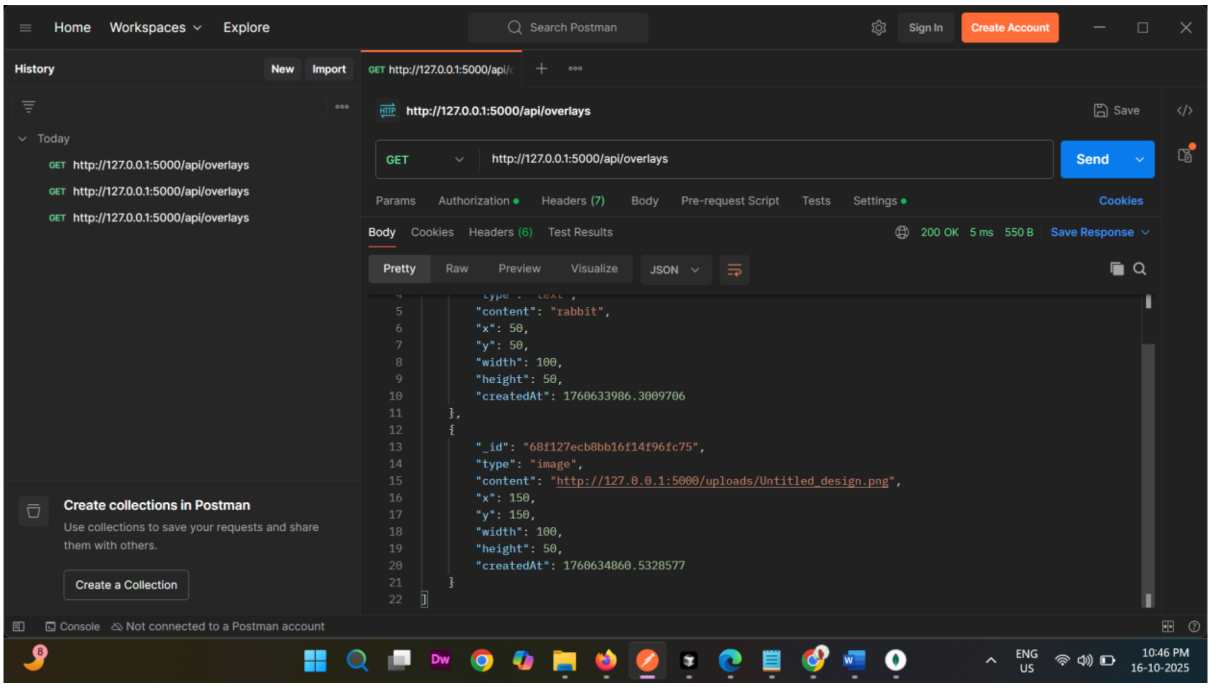
Create

Existing Overlays

text - rabbit
Position: (18, 109), Size: 100x50
Edit Delete

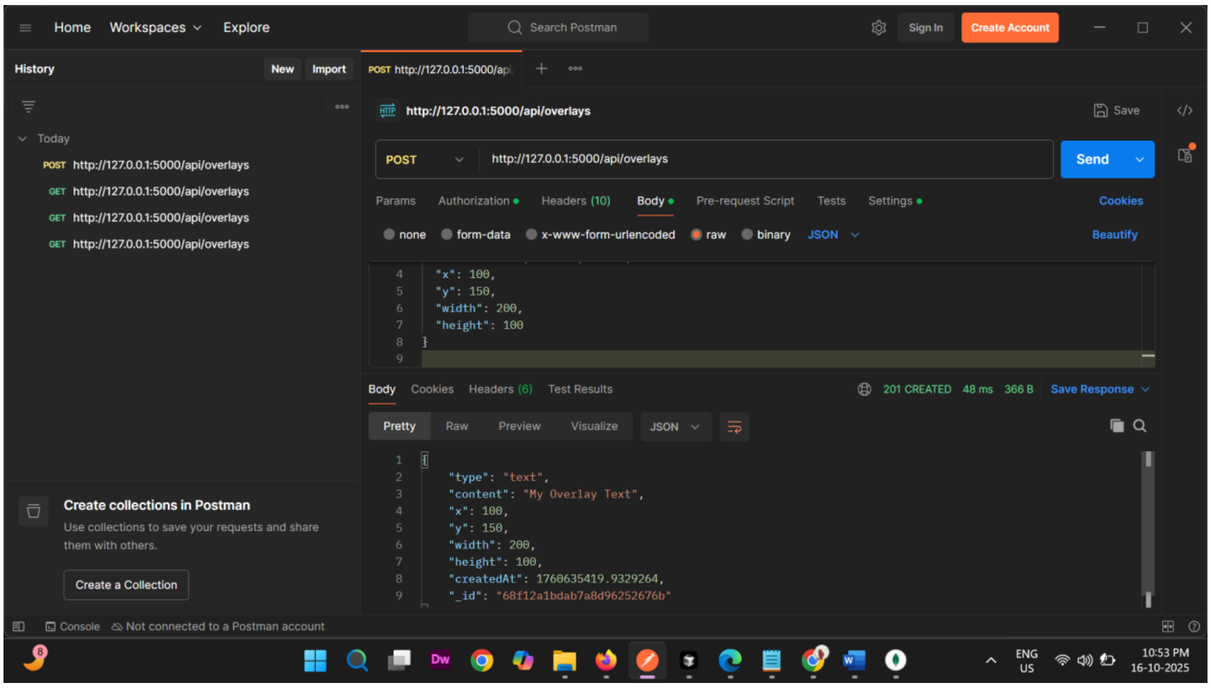
image -
Position: (13, 6), Size: 100x50
Edit Delete

12.I have retrieve using GET so with image I have created logo it is inserted and recorded in database



```
  "type": "text",
      "content": "rabbit",
      "x": 50,
      "y": 50,
      "width": 100,
      "height": 50,
      "createdAt": 1760633986.3009706
    },
    {
      "_id": "68f127ecb8bb16f14f96fc75",
      "type": "image",
      "content": "http://127.0.0.1:5000/uploads/Untitled_design.png",
      "x": 150,
      "y": 150,
      "width": 100,
      "height": 50,
      "createdAt": 1760634860.5328577
    }
]
```

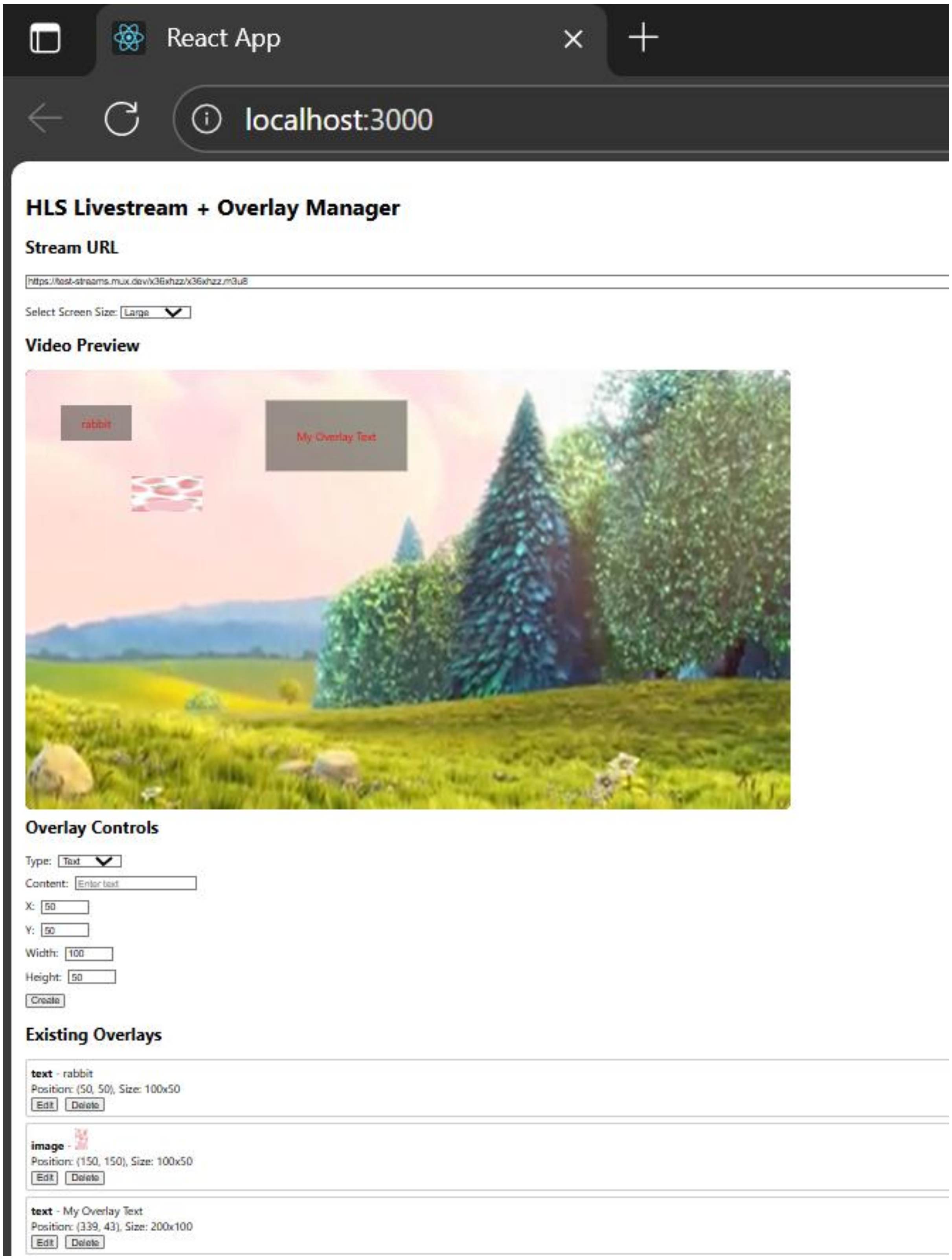
13.I have Created one more document using postman with POST api



```
  "x": 100,
  "y": 150,
  "width": 200,
  "height": 100
}
```

```
{
    "type": "text",
    "content": "My Overlay Text",
    "x": 100,
    "y": 150,
    "width": 200,
    "height": 100,
    "createdAt": 1760635419.9329264,
    "_id": "68f12a1bdab7a8d96252676b"
```

14.One more row data created display in livestream



HLS Livestream + Overlay Manager

Stream URL

https://test-streams.mux.dev/x36xhzz/x36xhzz.m3u8

Select Screen Size: Large

Video Preview

Overlay Controls

Type: Text
Content: Enter text
X: 50
Y: 50
Width: 100
Height: 50
Create

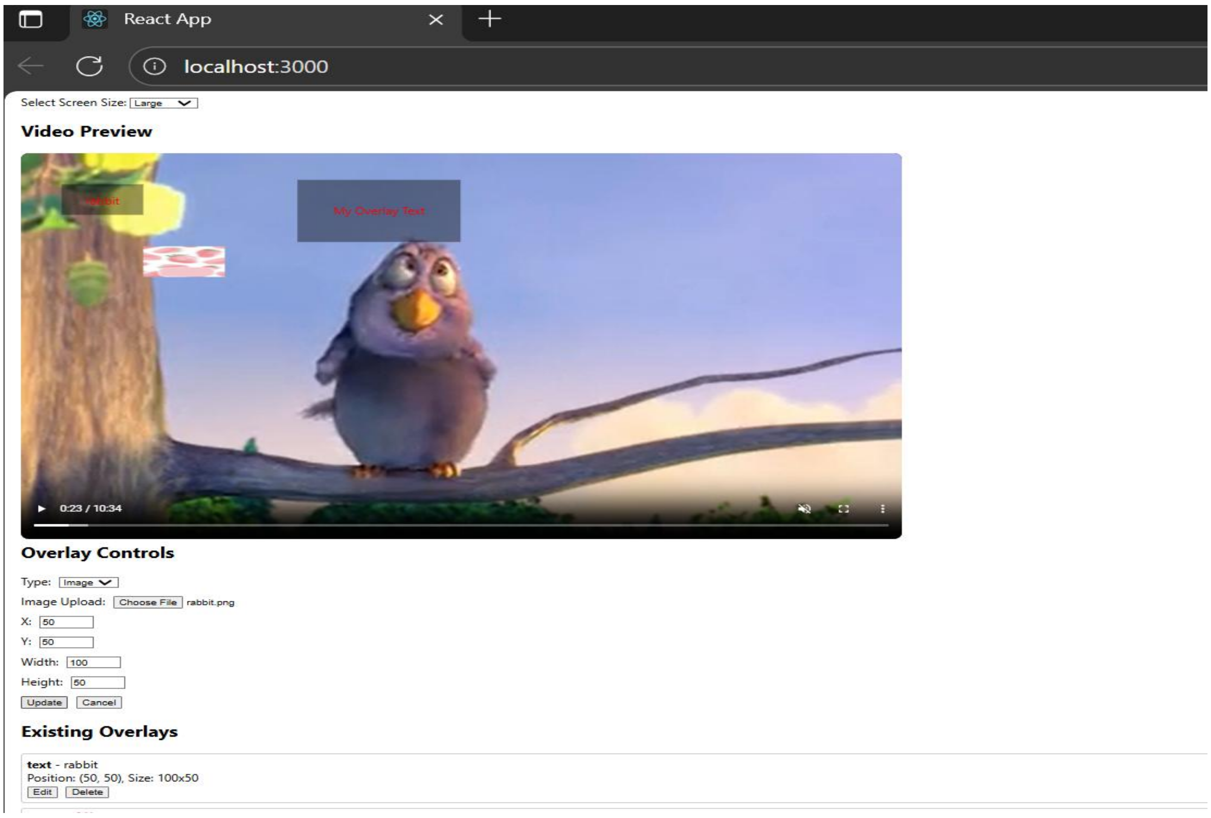
Existing Overlays

text - rabbit
Position: (50, 50), Size: 100x50
Edit Delete

image -
Position: (150, 150), Size: 100x50
Edit Delete

text - My Overlay Text
Position: (339, 43), Size: 200x100
Edit Delete

15.Editing the text-rabbit with replacing image -rabbit.png



Select Screen Size: Large

Video Preview

Overlay Controls

Type: Image

Image Upload: Choose File rabbit.png

X: 50

Y: 50

Width: 100

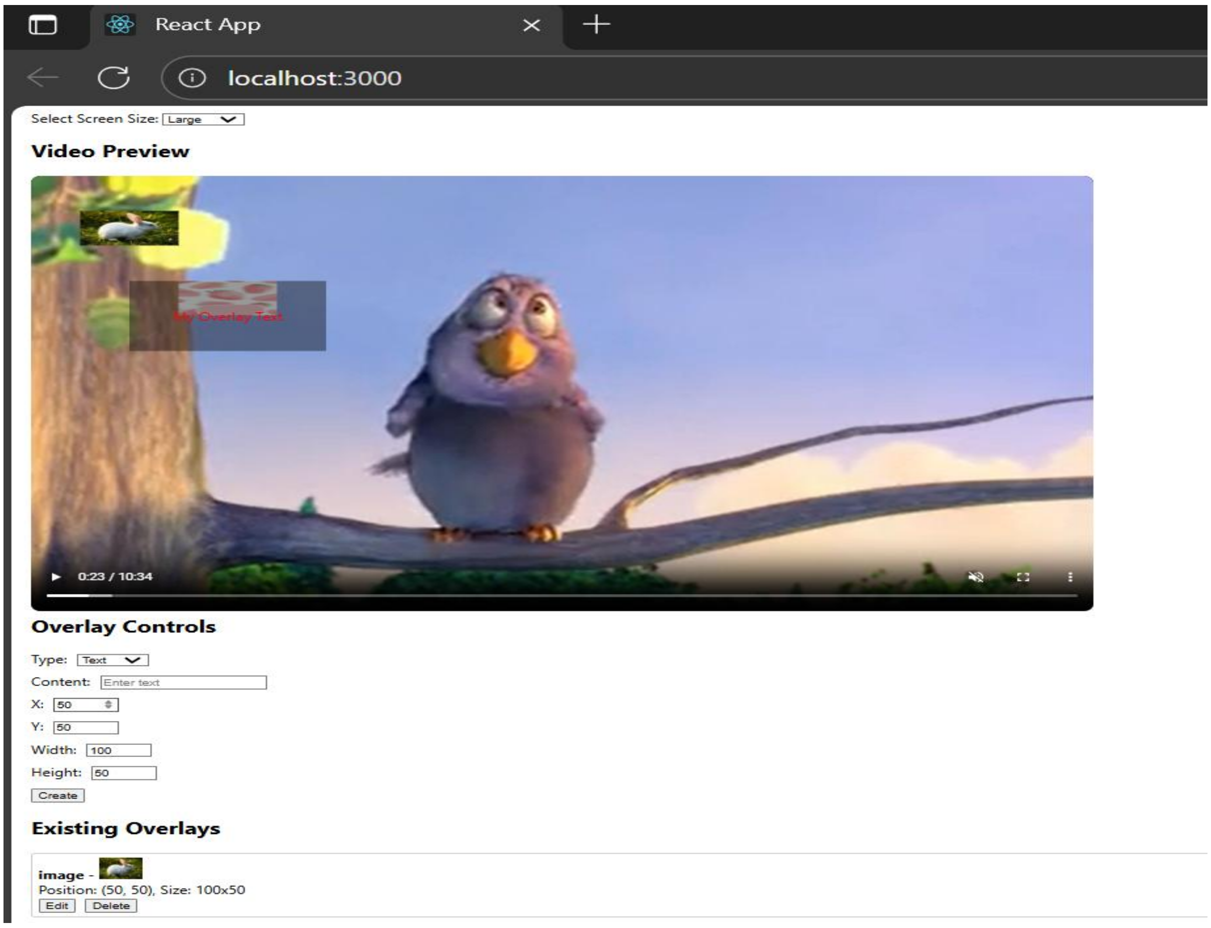
Height: 50

Update Cancel

Existing Overlays

text - rabbit
Position: (50, 50), Size: 100x50
Edit Delete

16.Update the rabbit.png image ,text replace with image



Select Screen Size: Large

Video Preview

Overlay Controls

Type: Text

Content: Enter text

X: 50

Y: 50

Width: 100

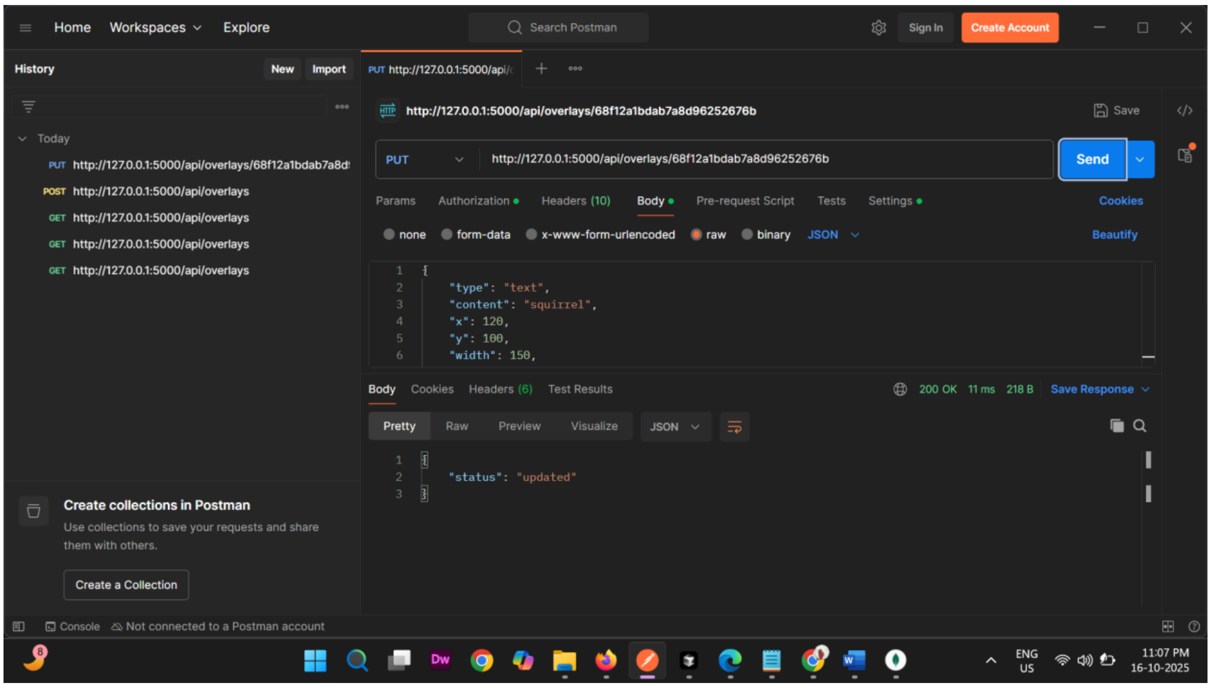
Height: 50

Create

Existing Overlays

image -
Position: (50, 50), Size: 100x50
Edit Delete

16.Using PUT api modifying the overlay details



PUT http://127.0.0.1:5000/api/overlays/68f12a1bdab7a8d96252676b

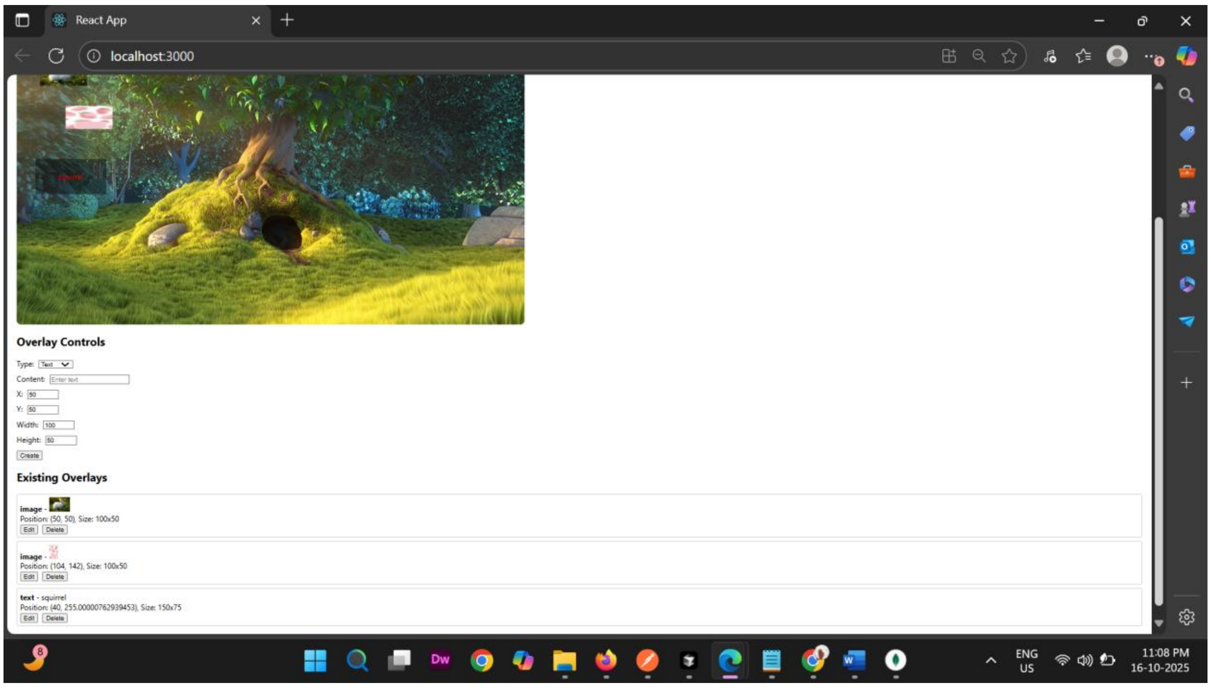
Body (raw, JSON):

```
{
    "type": "text",
    "content": "squirrel",
    "x": 120,
    "y": 100,
    "width": 150,
```

Response: 200 OK 11 ms 218 B

```
{
    "status": "updated"
}
```

17.Modify changes reflect in video stream



Overlay Controls

Type: Text
Content: Enter text
X: 50
Y: 50
Width: 100
Height: 50
Create

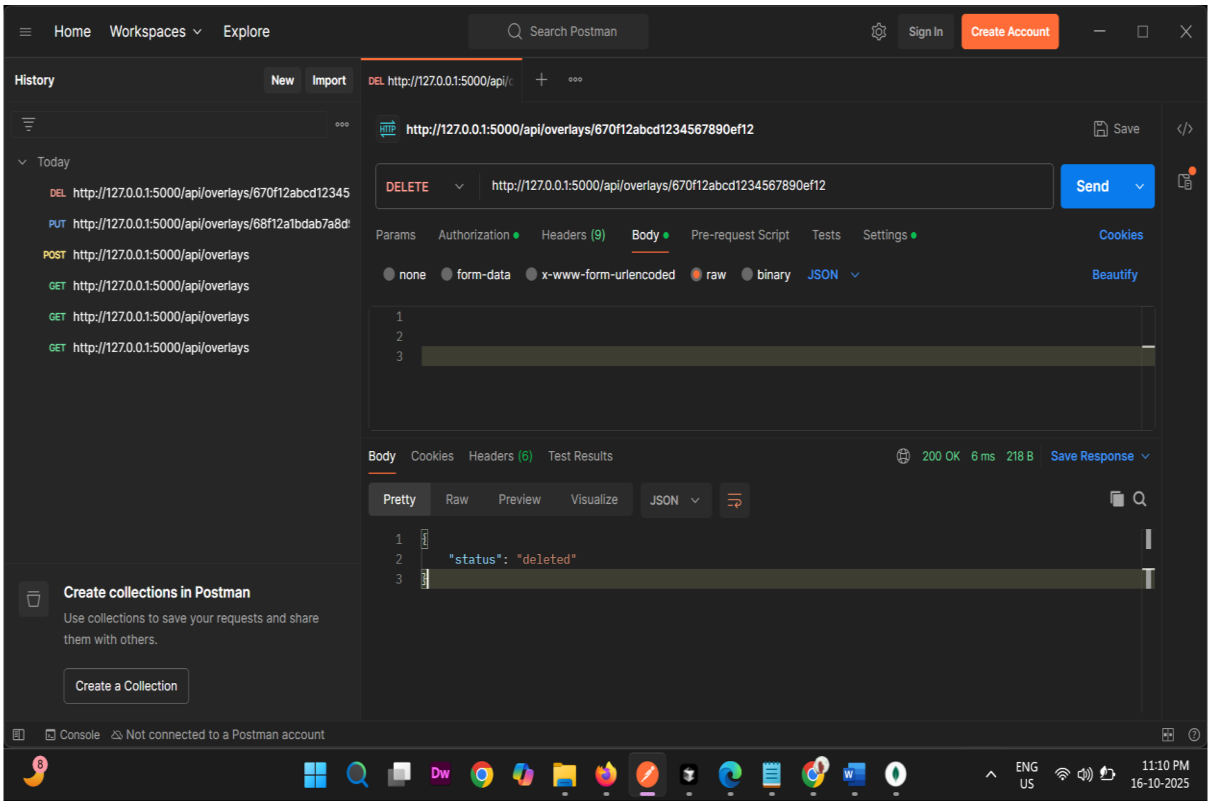
Existing Overlays

image -
Position: (50, 50), Size: 100x50
Edit Delete

image -
Position: (104, 142), Size: 100x50
Edit Delete

text - squirrel
Position: (40, 255.00000762939453), Size: 150x75
Edit Delete

18.Now detele the last row created with DELETE API from postman



History

Today

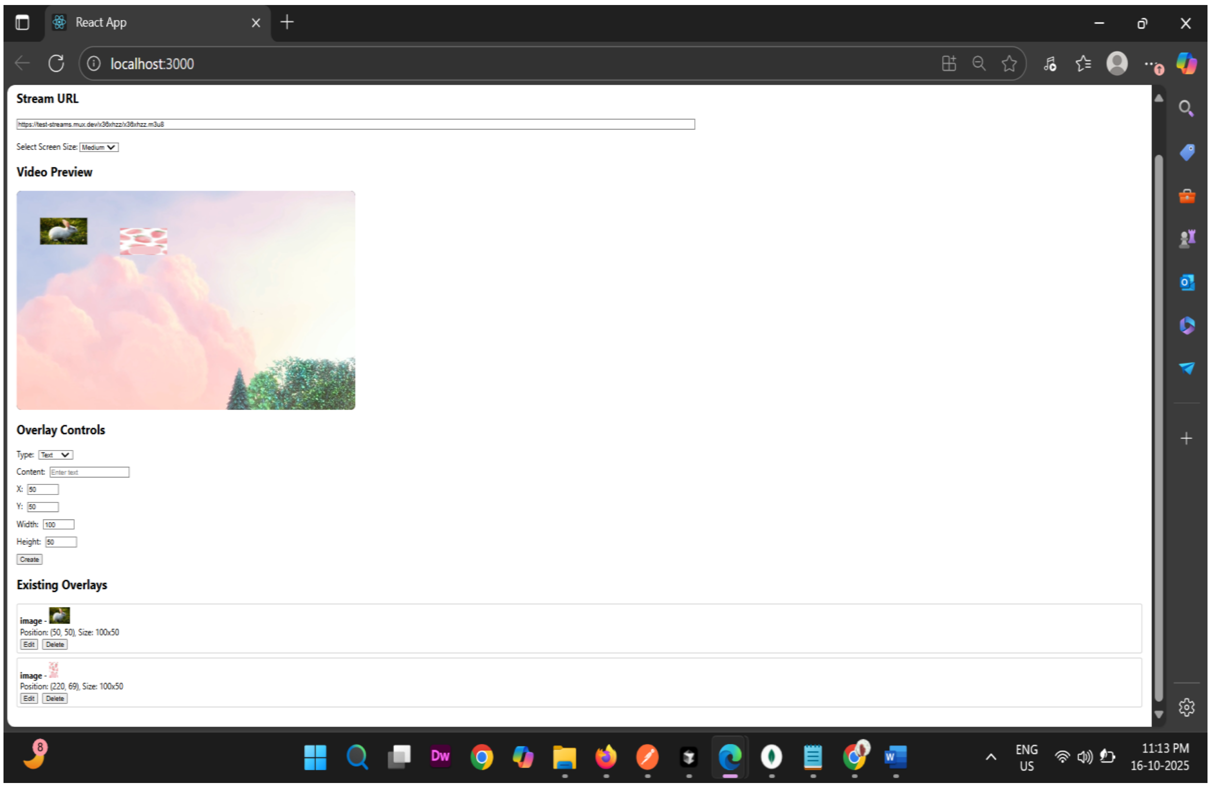
- DEL http://127.0.0.1:5000/api/overlays/670f12abcd12345
- PUT http://127.0.0.1:5000/api/overlays/68f12a1bdab7a8d
- POST http://127.0.0.1:5000/api/overlays
- GET http://127.0.0.1:5000/api/overlays
- GET http://127.0.0.1:5000/api/overlays
- GET http://127.0.0.1:5000/api/overlays

DELETE http://127.0.0.1:5000/api/overlays/670f12abcd1234567890ef12

200 OK 6 ms 218 B

```
{
  "status": "deleted"
}
```

19.Now the text with squirrel deleted



Stream URL

https://test-streams.mux.dev/x36xhzz/x36xhzz.m3u8

Select Screen Size: Medium

Video Preview

Overlay Controls

Type: Text

Content: Enter text

X: 50

Y: 50

Width: 100

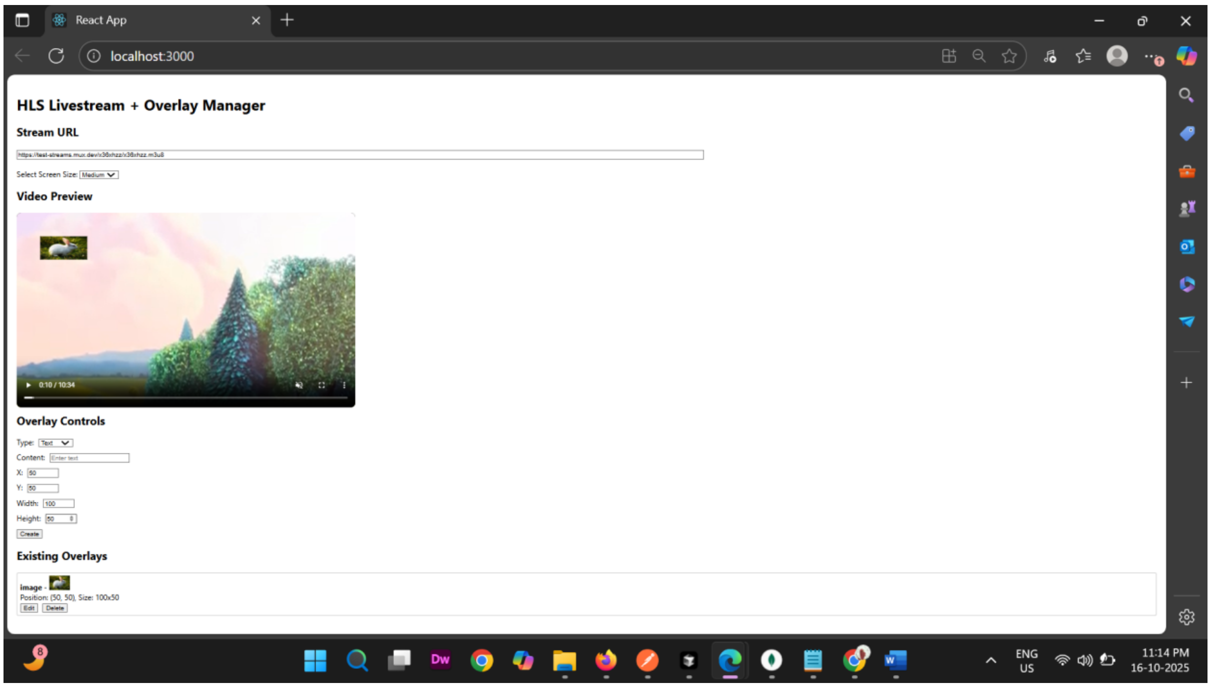
Height: 50

Existing Overlays

image -
Position: (50, 50), Size: 100x50

image -
Position: (220, 69), Size: 100x50

20.Also we can delete by clicking detele button



HLS Livestream + Overlay Manager

Stream URL

https://test-streams.mux.dev/x36xhzz/x36xhzz.m3u8

Select Screen Size: Medium

Video Preview

Overlay Controls

Type: Text

Content: Enter text

X: 50

Y: 50

Width: 100

Height: 50

Create

Existing Overlays

image -

Position: (50, 50), Size: 100x50

Edit Delete

Conclusion

1. Livestream with Overlays:You built a React app that plays a livestream video (HLS format), with draggable/resizable text or image overlays rendered on top.
2. CRUD Functionality via Flask + MongoDB:The backend (Flask) provides Create, Read, Update, and Delete APIs for managing overlay data stored in MongoDB.
3. Frontend Interaction:Users can add overlays, position them, and either create or update them directly on the video using a form and buttons.
4. API Testing with Postman:You can test your backend by sending POST, GET, PUT, and DELETE requests in Postman with appropriate JSON payloads and overlay _id.
5. Real-time Visual Feedback:Overlay changes (create/update/delete) reflect immediately on the video preview, making the app interactive and user-friendly.