HOW TO DEPLOY APPLICATION ON AWS SERVER

Step 1: Deploy the React Frontend

1. Build the React App

Prepare the production build of your React app:

```
npm run build
```

This generates a `build` directory with the optimized static files.

2. Create an S3 Bucket

- Go to the **S3** service in the AWS Management Console.
- Click **Create Bucket** and configure the following:
  - **Bucket Name**: Choose a unique name (e.g., `accessibility-checker-frontend`).
  - **Region**: Select your preferred AWS region.
  - **Block Public Access**: Uncheck "Block all public access" (you'll configure access settings later).
  - Click **Create Bucket**.

3. Enable Static Website Hosting

- Open your bucket and go to the **Properties** tab.
- Scroll down to **Static Website Hosting** and click **Edit**:
  - Select **Enable**.
  - Enter `index.html` for the **Index Document**.
  - Leave **Error Document** as `index.html`.
  - Save changes.

4. Upload the Build Files

- Go to the **Objects** tab in your bucket.
- Click **Upload → Add files**.
- Select all files from the `build` directory.
- Click **Upload**.

5. Set Bucket Permissions

- Go to the **Permissions** tab and scroll to the **Bucket Policy**.
- Add this policy to make the files publicly accessible:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::accessibility-checker-frontend/*"
    }
  ]
}
```

Replace `accessibility-checker-frontend` with your bucket name.

6. Test the Frontend

Access the URL provided in the **Static Website Hosting** section of the bucket properties.

Step 2: Deploy the Node.js Backend

1. Prepare the Backend

Ensure your backend code is working locally, using `multer.memoryStorage()` for uploads and handling analysis in memory or with S3.

2. Package the Backend

- Create a `zip` archive of your backend files, including dependencies (`node_modules`).
- Example:

```
zip -r backend.zip server.js utils/analyzer.js node_modules
```

3. Create a Lambda Function

- Go to the **Lambda** service in the AWS Management Console.
- Click **Create Function** and configure:
  - **Function Name**: e.g., `accessibility-checker-backend`.
  - **Runtime**: Node.js 18.x.
  - **Execution Role**: Create or select a role with S3 and Lambda permissions.
- Click **Create Function**.

4. Upload the Code

- In the Lambda function dashboard, go to the **Code** tab.
- Click **Upload from → .zip file →** Upload your `backend.zip`.

5. Test the Lambda Function

- Click **Test** and provide a sample event (e.g., a simulated file upload).

- Verify the function processes requests correctly.

Step 3: Configure API Gateway

1. Create an API Gateway

- Go to the **API Gateway** service.
- Click **Create API → HTTP API → Build**.
- Add a new route:
  - **Resource Path**: `/upload`.
  - **HTTP Method**: POST.

2. Integrate with Lambda

- For the `/upload` route, click **Edit Integration** and select your Lambda function (`accessibility-checker-backend`).

3. Enable CORS

- Go to the API Gateway **CORS Settings** and allow requests from your frontend origin (`http://<your-s3-bucket-url>`).

4. Deploy the API

- Click **Deploy API**.
- Note the **API Endpoint URL** (e.g., `https://abc123.execute-api.us-east-1.amazonaws.com/upload`).

Step 4: Integrate Frontend with Backend

1. Update Frontend API URL

In your React app, update the API endpoint in the relevant component (e.g., `FileUpload.js`):

```
const API_URL = "https://abc123.execute-api.us-east-
1.amazonaws.com/upload";
```

2. Redeploy the Frontend

Rebuild the React app and re-upload it to the S3 bucket.

Step 5: Optional - Add CloudFront for Frontend

1. Create a CloudFront Distribution

- Go to the **CloudFront** service.
- Click **Create Distribution** and configure:
  - **Origin Domain Name**: Your S3 bucket URL.
  - **Viewer Protocol Policy**: Redirect HTTP to HTTPS.

2. Deploy and Test

- CloudFront will provide a public URL for your frontend. Use this for accessing your app globally with reduced latency.