**Resume Projects Flow**

**Project: Secure Chatbot Hosting on AWS (Thomson Reuters)**

When someone opens the website with your chatbot integrated, here's a step-by-step breakdown of what happens behind the scenes:

**🌐 1. Website Loads in Browser**

* The user visits the main website (e.g., [www.thomsonreuters.com](https://www.thomsonreuters.com)).
* The HTML page includes a <script> tag or iframe that loads the **chatbot widget**.
* This widget is a **React.js frontend**, hosted on **Amazon S3** and delivered via **CloudFront CDN** for fast loading.

**💬 2. Chatbot Widget Appears (Bottom-Right Corner)**

* The widget loads asynchronously and appears at the bottom-right of the screen.
* It initializes with a welcome message or prompt (e.g., “Hi! How can I help you today?”).
* The UI is interactive and styled to match the website’s branding.

**🔄 3. User Sends a Message**

* The user types a message into the chatbot.
* The frontend sends this message to the backend via **HTTPS API call** (e.g., POST /chat/message).

**🧠 4. Backend Processes the Message**

* The request hits **Nginx** on the EC2 instance, which forwards it to the **Node.js backend**.
* The backend:
  + Validates the request (e.g., JWT token).
  + Sends the message to an NLP engine (like Dialogflow or custom logic).
  + Gets a response (e.g., “Here’s your document…”).

**📤 5. Response Sent Back to Frontend**

* The backend sends the response back to the chatbot widget.
* The widget displays the reply in the chat window.

**🔐 6. Security & Monitoring**

* All communication is encrypted via **HTTPS**.
* **JWT tokens** ensure only authenticated users can access certain features.
* **CloudWatch** logs and monitors backend performance.
* **Snyk** and **SonarQube** ensure secure and clean code during CI/CD.

**🔁 7. CI/CD Keeps Everything Updated**

* Any code changes pushed to GitHub trigger **GitHub Actions**:
  + Frontend updates are deployed to S3.
  + Backend updates are deployed to EC2.
  + Security scans and code quality checks run automatically.

**Frontend**

**When someone opens the website, and your chatbot frontend is hosted on Amazon S3, here’s exactly how it becomes visible and functional on the site:**

**🌐 Step-by-Step Flow: How the Chatbot Frontend Is Seen from S3**

**✅ 1. Chatbot Widget Hosted on S3**

* **You build your chatbot frontend (e.g., React.js) and run npm run build.**
* **The build output (HTML, CSS, JS files) is uploaded to an S3 bucket configured for static website hosting.**
* **Example URL:** [**https://chatbot-widget.s3.amazonaws.com/index.html**](https://chatbot-widget.s3.amazonaws.com/index.html)

**✅ 2. Delivered via CloudFront (CDN)**

* **You attach a CloudFront distribution to the S3 bucket.**
* **This gives you a fast, secure, and globally distributed URL like:**

**✅ 3. Embedded into Main Website**

* **On the main website (e.g.,** [**www.thomsonreuters.com**](https://www.thomsonreuters.com)**), you include the chatbot widget using:  
  or**

**✅ 4. Widget Loads in Browser**

* **When the user opens the website:** 
  + **The browser fetches the chatbot widget from CloudFront, which pulls it from S3.**
  + **The widget appears at the bottom-right corner of the page.**
  + **It initializes and connects to the backend API hosted on EC2.**

**✅ 5. Secure Communication**

* **All communication between the widget and backend is done over HTTPS.**
* **You can use JWT tokens to authenticate users if needed.**

**🛡️ Security & Performance Best Practices**

* **S3 Bucket Policy: Make sure it's public only for static files, not sensitive data.**
* **CloudFront: Use HTTPS and enable caching.**
* **CORS: Configure CORS on S3 and backend to allow cross-origin requests.**
* **Versioning: Use versioned URLs for JS files to avoid caching issues.**

[**https://d123abc.cloudfront.net/chatbot.js**](https://d123abc.cloudfront.net/chatbot.js)

**Backend + JWT**

Here’s a **complete end-to-end explanation** of your chatbot system, including **JWT token generation, validation, and how everything works together**:

**🔄 Full Chatbot Workflow (Frontend + Backend + JWT)**

**🧩 1. User Opens the Website**

* The main website loads in the browser.
* A chatbot widget appears at the bottom-right corner.
* This widget is a **React.js frontend**, hosted on **Amazon S3** and delivered via **CloudFront**.

**🔐 2. User Logs In or Starts a Session**

* The chatbot asks the user to log in or authenticate.
* The frontend sends a login request to the backend:

POST /auth/login  
{  
  "email": "user@example.com",  
  "password": "securepassword"  
}

**🧠 3. Backend Authenticates and Generates JWT**

* The **Node.js backend** (running on EC2) verifies the credentials.
* If valid, it generates a **JWT token** using a secret key:

const jwt = require('jsonwebtoken');  
  
const token = jwt.sign(  
  { userId: user.id, email: user.email },  
  process.env.JWT\_SECRET,  
  { expiresIn: '1h' }  
);

* The token is sent back to the frontend:

{  
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6..."  
}

**📦 4. Frontend Stores the JWT**

* The frontend stores the token in memory or localStorage.
* For every future API call (e.g., sending a message), it includes the token:

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6...

**💬 5. User Sends a Message**

* The chatbot sends a message to the backend:

POST /chat/message  
Headers: Authorization: Bearer &lt;JWT&gt;  
Body: { message: "Hello!" }

**🔐 6. Backend Validates JWT**

* The request reaches **Nginx** on EC2, which forwards it to the Node.js backend.
* A **middleware** in Node.js checks the JWT:

const jwt = require('jsonwebtoken');  
  
function authenticateToken(req, res, next) {  
  const authHeader = req.headers['authorization'];  
  const token = authHeader &amp;&amp; authHeader.split(' ')[1];  
  
  if (!token) return res.sendStatus(401);  
  
  jwt.verify(token, process.env.JWT\_SECRET, (err, user) =&gt; {  
    if (err) return res.sendStatus(403);  
    req.user = user;  
    next();  
  });  
}

* This middleware is applied to protected routes like /chat/message.

**🧠 7. Backend Processes the Message**

* If the token is valid:
  + The backend processes the message.
  + It may use **Dialogflow**, custom NLP, or business logic.
  + Optionally stores the chat in a database.

**📤 8. Response Sent Back to Frontend**

* The backend sends a reply (e.g., “Hi! How can I help you?”).
* The chatbot widget displays the response in the chat window.

**🔁 9. CI/CD Keeps Everything Updated**

* **GitHub Actions** automates deployment:
  + Frontend → S3
  + Backend → EC2
  + Runs **Snyk** and **SonarQube** for security and code quality.

**📊 10. Monitoring & Security**

* **CloudWatch** tracks EC2 performance and logs.
* **HTTPS** ensures secure communication.
* **JWT** secures user sessions.
* **IAM roles** restrict access to AWS resources.

Would you like me to generate a **diagram** showing this flow visually or help you write the **Node.js backend code** for login, JWT generation, and message handling?

**🧠 Backend Code Structure Overview**

Your backend will usually be organized like this:

Plain Text

chatbot-backend/  
├── controllers/  
│   └── chatController.js         # Handles chat message logic  
├── middleware/  
│   └── authMiddleware.js         # JWT validation logic  
├── routes/  
│   └── chatRoutes.js             # API route definitions  
├── services/  
│   └── nlpService.js             # Dialogflow or custom NLP logic  
├── utils/  
│   └── tokenUtils.js             # JWT generation functions  
├── .env                          # Environment variables (JWT secret, etc.)  
├── app.js                        # Main Express app setup  
├── package.json                  # Dependencies and scripts  
├── ecosystem.config.js           # PM2 config (optional)  
└── README.md                     # Project documentation

**🔍 Key Components Explained**

**1. app.js – Main Server Setup**

Plain Text

const express = require('express');  
const app = express();  
const chatRoutes = require('./routes/chatRoutes');  
  
app.use(express.json());  
app.use('/chat', chatRoutes);  
  
const PORT = process.env.PORT || 3000;  
app.listen(PORT, () =&gt; console.log(`Server running on port ${PORT}`));

**2. routes/chatRoutes.js – API Routes**

Plain Text

const express = require('express');  
const router = express.Router();  
const { handleMessage } = require('../controllers/chatController');  
const authenticateToken = require('../middleware/authMiddleware');  
  
router.post('/message', authenticateToken, handleMessage);  
  
module.exports = router;

**3. middleware/authMiddleware.js – JWT Validation**

Plain Text

const jwt = require('jsonwebtoken');  
  
function authenticateToken(req, res, next) {  
  const authHeader = req.headers['authorization'];  
  const token = authHeader &amp;&amp; authHeader.split(' ')[1];  
  
  if (!token) return res.status(401).json({ error: 'Token missing' });  
  
  jwt.verify(token, process.env.JWT\_SECRET, (err, user) =&gt; {  
    if (err) return res.status(403).json({ error: 'Token invalid' });  
    req.user = user;  
    next();  
  });  
}  
  
module.exports = authenticateToken;

**4. controllers/chatController.js – Message Handling**

Plain Text

const { processMessage } = require('../services/nlpService');  
  
exports.handleMessage = async (req, res) =&gt; {  
  const userMessage = req.body.message;  
  const userInfo = req.user;  
  
  const botReply = await processMessage(userMessage, userInfo);  
  res.json({ reply: botReply });  
};

**5. services/nlpService.js – NLP Integration**

Plain Text

exports.processMessage = async (message, user) =&gt; {  
  // Call Dialogflow or custom logic  
  return `Hello ${user.email}, you said: ${message}`;  
};

**6. utils/tokenUtils.js – JWT Generation**

Plain Text

const jwt = require('jsonwebtoken');  
  
exports.generateToken = (user) =&gt; {  
  return jwt.sign(  
    { id: user.id, email: user.email },  
    process.env.JWT\_SECRET,  
    { expiresIn: '1h' }  
  );  
};

**7. .env – Environment Variables**

Plain Text

JWT\_SECRET=your\_super\_secret\_key

**8. ecosystem.config.js – PM2 Config (Optional)**

Plain Text

module.exports = {  
  apps: [{  
    name: 'chatbot-backend',  
    script: 'app.js',  
    env: {  
      NODE\_ENV: 'production',  
      JWT\_SECRET: 'your\_super\_secret\_key'  
    }  
  }]  
};

**9. CI/CD with GitHub Actions**

Your GitHub Actions workflow will:

* Pull latest code
* Install dependencies
* Restart the backend service on EC2