**Programming and Application**

**By Devi Sri Charan**

[Securing and documenting the APIs 2](#_Toc2049586889)

[Integrating Amazon API Gateway 3](#_Toc1061484057)

[Implementing security mechanisms for the API using API keys 3](#_Toc530309319)

[1. Generating API Keys 3](#_Toc392868914)

[2. Middleware for API Key Validation: 3](#_Toc1379195640)

[3. Applying Middleware to Routes 3](#_Toc956810005)

[Using Swagger to document the API endpoints, detailing parameters, responses, and error handling. 4](#_Toc693535017)

[1. Setting Up Swagger Configuration 4](#_Toc2043280140)

[2. Adding Swagger Annotations to the API 5](#_Toc1104892008)

# Securing and documenting the APIs

## Integrating Amazon API Gateway

## Implementing security mechanisms for the API using API keys

To secure your claims management system with API keys, I used middleware to check for the presence and validity of an API key in each request. Here’s how I implemented this:

### Generating API Keys

I manually generated API keys and stored them securely, in environment variables.

`.env file`

API\_KEY1="035b90011e5843db"

API\_KEY2="1951fdcb661f2129"

### Middleware for API Key Validation:

Created middleware to check for the presence of a valid API key in incoming requests.

`apiKeyMiddleware.js`

const dotenv = require("dotenv")

dotenv.config()

const validAPIKeys = [process.env.API\_KEY1, process.env.API\_KEY2]

const apiKeyMiddleware = (req,res,next) => {

const apiKey = req.header('x-api-key');

if (!apiKey || !validAPIKeys.includes(apiKey)){

return res.status(403).json({message: "Forbidden - Invalid API Key"})

}

next();

}

module.exports = apiKeyMiddleware

### Applying Middleware to Routes

Used the middleware in all routes to ensure only requests with valid API keys are processed.

const apiKeyMiddleware = require('./middleware/apiKeyMiddleware');

app.use(apiKeyMiddleware);

## Using Swagger to document the API endpoints, detailing parameters, responses, and error handling.

To document the API using Swagger, I used the `swagger-jsdoc` and `swagger-ui-express` packages. Swagger provides a user-friendly interface for interacting with and testing the API endpoints.

### Setting Up Swagger Configuration

Created a file named swaggerConfig.js to define the Swagger options:

const swaggerJSDoc = require('swagger-jsdoc')

const swaggerUi = require('swagger-ui-express')

const swaggerDefinition = {

openapi: '3.0.0',

info: {

title: 'Claims Management System',

version: '1.0.0',

description: 'This collection contains all the necessary API endpoints for managing policyholders, policies, and claims within a Claims Management System. Each endpoint supports various operations including creation, retrieval, updating, and deletion of records. This API is designed to facilitate the administration of insurance claims and related data, making it easier for users to handle insurance-related transactions.',

},

components: {

securitySchemes: {

ApiKeyAuth: {

type: 'apiKey',

in: 'header',

name: 'x-api-key',

description: 'API key for authorization',

},

},

},

security: [

{

ApiKeyAuth: [],

},

],

servers: [

{

url: '<http://localhost:3000>',

description: 'Development server',

},

],

}

const options = {

swaggerDefinition,

apis: ['./app.js'],

};

const swaggerSpec = swaggerJSDoc(options);

module.exports = (app) => {

app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerSpec));

}

### Adding Swagger Annotations to the API

const express = require('express');

const { body, validationResult } = require('express-validator');

const bodyParser = require('body-parser');

const mongoose = require("mongoose");

const cors = require("cors");

const dotenv = require("dotenv");

const apiKeyMiddleware = require('./middleware/apiKeyMiddleware');

const setupSwagger = require('./swaggerConfig');

dotenv.config();

mongoose

.connect(process.env.MONGODB\_URL)

.then(() => console.log("Connected to MongoDB"))

.catch((err) => console.log('Could not connect to MongoDB', err));

const app = express();

app.use(bodyParser.json());

app.use(cors());

app.use(express.json());

app.use(express.urlencoded({ extended: true }));

const Policyholder = require("./models/Policyholder");

const Policy = require("./models/Policy");

const Claim = require("./models/Claim");

setupSwagger(app);

app.get('/', (req, res) => {

res.send('Welcome to Claims Management System API');

});

// Apply the API key middleware to all routes

app.use(apiKeyMiddleware);

/\*\*

\* @swagger

\* components:

\* schemas:

\* Policyholder:

\* type: object

\* required:

\* - policyholder\_id

\* - name

\* - date\_of\_birth

\* - address

\* - phone

\* properties:

\* policyholder\_id:

\* type: string

\* description: The unique ID of the policyholder

\* name:

\* type: string

\* description: The name of the policyholder

\* date\_of\_birth:

\* type: string

\* format: date

\* description: The date of birth of the policyholder

\* address:

\* type: string

\* description: The address of the policyholder

\* phone:

\* type: string

\* description: The phone number of the policyholder

\* example:

\* policyholder\_id: "1"

\* name: "John Doe"

\* date\_of\_birth: "1980-01-01"

\* address: "123 Main St"

\* phone: "555-555-5555"

\*/

/\*\*

\* @swagger

\* /policyholders:

\* post:

\* summary: Create a new policyholder

\* tags: [Policyholders]

\* requestBody:

\* required: true

\* content:

\* application/json:

\* schema:

\* $ref: '#/components/schemas/Policyholder'

\* responses:

\* 201:

\* description: The policyholder was successfully created

\* content:

\* application/json:

\* schema:

\* $ref: '#/components/schemas/Policyholder'

\* 400:

\* description: Bad request

\*/

app.post('/policyholders', [

body('policyholder\_id').notEmpty().withMessage('Policyholder ID is required'),

body('name').notEmpty().withMessage('Name is required'),

body('date\_of\_birth').isDate().withMessage('Date of Birth must be a valid date'),

body('address').notEmpty().withMessage('Address is required'),

body('phone').notEmpty().withMessage('Phone number is required')

], async (req, res) => {

const errors = validationResult(req);

if (!errors.isEmpty()) {

return res.status(400).json({ errors: errors.array() });

}

try {

const policyholder = new Policyholder(req.body);

await policyholder.save();

res.status(201).send(policyholder);

} catch (error) {

res.status(400).send(error.message);

}

});

/\*\*

\* @swagger

\* /policyholders/{policyholder\_id}:

\* get:

\* summary: Retrieve a specific policyholder

\* tags: [Policyholders]

\* parameters:

\* - in: path

\* name: policyholder\_id

\* required: true

\* schema:

\* type: string

\* description: The ID of the policyholder to retrieve

\* responses:

\* 200:

\* description: A policyholder object

\* content:

\* application/json:

\* schema:

\* $ref: '#/components/schemas/Policyholder'

\* 404:

\* description: Policyholder not found

\*/

app.get('/policyholders/:policyholder\_id', async (req, res) => {

try {

const policyholder = await Policyholder.findOne({ policyholder\_id: req.params.policyholder\_id });

if (!policyholder) return res.status(404).send('Policyholder not found');

res.send(policyholder);

} catch (error) {

res.status(400).send(error.message);

}

});

/\*\*

\* @swagger

\* /policyholders/{policyholder\_id}:

\* put:

\* summary: Update a policyholder

\* tags: [Policyholders]

\* parameters:

\* - in: path

\* name: policyholder\_id

\* required: true

\* schema:

\* type: string

\* description: The ID of the policyholder to update

\* requestBody:

\* required: true

\* content:

\* application/json:

\* schema:

\* $ref: '#/components/schemas/Policyholder'

\* responses:

\* 200:

\* description: The updated policyholder

\* content:

\* application/json:

\* schema:

\* $ref: '#/components/schemas/Policyholder'

\* 404:

\* description: Policyholder not found

\*/

app.put('/policyholders/:policyholder\_id', async (req, res) => {

try {

const policyholder = await Policyholder.findOneAndUpdate({ policyholder\_id: req.params.policyholder\_id }, req.body, { new: true, runValidators: true });

if (!policyholder) return res.status(404).send('Policyholder not found');

res.send(policyholder);

} catch (error) {

res.status(400).send(error.message);

}

});

/\*\*

\* @swagger

\* /policyholders/{policyholder\_id}:

\* delete:

\* summary: Delete a policyholder

\* tags: [Policyholders]

\* parameters:

\* - in: path

\* name: policyholder\_id

\* required: true

\* schema:

\* type: string

\* description: The ID of the policyholder to delete

\* responses:

\* 200:

\* description: The policyholder was successfully deleted

\* content:

\* application/json:

\* schema:

\* type: object

\* properties:

\* message:

\* type: string

\* 404:

\* description: Policyholder not found

\*/

app.delete('/policyholders/:policyholder\_id', async (req, res) => {

try {

const policyholder = await Policyholder.findOneAndDelete({ policyholder\_id: req.params.policyholder\_id });

if (!policyholder) return res.status(404).send('Policyholder not found');

res.send({ message: 'Policyholder deleted successfully' });

} catch (error) {

res.status(400).send(error.message);

}

});

/\*\*

\* @swagger

\* components:

\* schemas:

\* Policy:

\* type: object

\* required:

\* - policy\_id

\* - policyholder\_id

\* - start\_date

\* - end\_date

\* - premium

\* properties:

\* policy\_id:

\* type: string

\* description: The unique ID of the policy

\* policyholder\_id:

\* type: string

\* description: The ID of the policyholder

\* start\_date:

\* type: string

\* format: date

\* description: The start date of the policy

\* end\_date:

\* type: string

\* format: date

\* description: The end date of the policy

\* premium:

\* type: number

\* description: The premium amount of the policy

\* example:

\* policy\_id: "p12345"

\* policyholder\_id: "ph12345"

\* start\_date: "2023-01-01"

\* end\_date: "2024-01-01"

\* premium: 1000.0

\*/

/\*\*

\* @swagger

\* /policies:

\* post:

\* summary: Create a new policy

\* tags: [Policies]

\* requestBody:

\* required: true

\* content:

\* application/json:

\* schema:

\* $ref: '#/components/schemas/Policy'

\* responses:

\* 201:

\* description: The policy was successfully created

\* content:

\* application/json:

\* schema:

\* $ref: '#/components/schemas/Policy'

\* 400:

\* description: Bad request

\*/

app.post('/policies', [

body('policy\_id').notEmpty().withMessage('Policy ID is required'),

body('policyholder\_id').notEmpty().withMessage('Policyholder ID is required'),

body('start\_date').isDate().withMessage('Start Date must be a valid date'),

body('end\_date').isDate().withMessage('End Date must be a valid date'),

body('premium').isNumeric().withMessage('Premium must be a number')

], async (req, res) => {

const errors = validationResult(req);

if (!errors.isEmpty()) {

return res.status(400).json({ errors: errors.array() });

}

try {

const policy = new Policy(req.body);

await policy.save();

res.status(201).send(policy);

} catch (error) {

res.status(400).send(error.message);

}

});

/\*\*

\* @swagger

\* /policies/{policy\_id}:

\* get:

\* summary: Retrieve a specific policy

\* tags: [Policies]

\* parameters:

\* - in: path

\* name: policy\_id

\* required: true

\* schema:

\* type: string

\* description: The ID of the policy to retrieve

\* responses:

\* 200:

\* description: A policy object

\* content:

\* application/json:

\* schema:

\* $ref: '#/components/schemas/Policy'

\* 404:

\* description: Policy not found

\*/

app.get('/policies/:policy\_id', async (req, res) => {

try {

const policy = await Policy.findOne({ policy\_id: req.params.policy\_id });

if (!policy) return res.status(404).send('Policy not found');

res.send(policy);

} catch (error) {

res.status(400).send(error.message);

}

});

/\*\*

\* @swagger

\* /policies/{policy\_id}:

\* put:

\* summary: Update a policy

\* tags: [Policies]

\* parameters:

\* - in: path

\* name: policy\_id

\* required: true

\* schema:

\* type: string

\* description: The ID of the policy to update

\* requestBody:

\* required: true

\* content:

\* application/json:

\* schema:

\* $ref: '#/components/schemas/Policy'

\* responses:

\* 200:

\* description: The updated policy

\* content:

\* application/json:

\* schema:

\* $ref: '#/components/schemas/Policy'

\* 404:

\* description: Policy not found

\*/

app.put('/policies/:policy\_id', async (req, res) => {

try {

const policy = await Policy.findOneAndUpdate({ policy\_id: req.params.policy\_id }, req.body, { new: true, runValidators: true });

if (!policy) return res.status(404).send('Policy not found');

res.send(policy);

} catch (error) {

res.status(400).send(error.message);

}

});

/\*\*

\* @swagger

\* /policies/{policy\_id}:

\* delete:

\* summary: Delete a policy

\* tags: [Policies]

\* parameters:

\* - in: path

\* name: policy\_id

\* required: true

\* schema:

\* type: string

\* description: The ID of the policy to delete

\* responses:

\* 200:

\* description: The policy was successfully deleted

\* content:

\* application/json:

\* schema:

\* type: object

\* properties:

\* message:

\* type: string

\* 404:

\* description: Policy not found

\*/

app.delete('/policies/:policy\_id', async (req, res) => {

try {

const policy = await Policy.findOneAndDelete({ policy\_id: req.params.policy\_id });

if (!policy) return res.status(404).send('Policy not found');

res.send({ message: 'Policy deleted successfully' });

} catch (error) {

res.status(400).send(error.message);

}

});

/\*\*

\* @swagger

\* components:

\* schemas:

\* Claim:

\* type: object

\* required:

\* - claim\_id

\* - policy\_id

\* - date\_of\_claim

\* - claim\_amount

\* - status

\* properties:

\* claim\_id:

\* type: string

\* description: The unique ID of the claim

\* policy\_id:

\* type: string

\* description: The ID of the policy

\* date\_of\_claim:

\* type: string

\* format: date

\* description: The date of the claim

\* claim\_amount:

\* type: number

\* description: The amount of the claim

\* status:

\* type: string

\* description: The status of the claim

\* example:

\* claim\_id: "c12345"

\* policy\_id: "p12345"

\* date\_of\_claim: "2023-06-01"

\* claim\_amount: 5000.0

\* status: "Pending"

\*/

/\*\*

\* @swagger

\* /claims:

\* post:

\* summary: Create a new claim

\* tags: [Claims]

\* requestBody:

\* required: true

\* content:

\* application/json:

\* schema:

\* $ref: '#/components/schemas/Claim'

\* responses:

\* 201:

\* description: The claim was successfully created

\* content:

\* application/json:

\* schema:

\* $ref: '#/components/schemas/Claim'

\* 400:

\* description: Bad request

\*/

app.post('/claims', [

body('claim\_id').notEmpty().withMessage('Claim ID is required'),

body('policy\_id').notEmpty().withMessage('Policy ID is required'),

body('date\_of\_claim').isDate().withMessage('Date of Claim must be a valid date'),

body('claim\_amount').isNumeric().withMessage('Claim Amount must be a number'),

body('status').notEmpty().withMessage('Status is required')

], async (req, res) => {

const errors = validationResult(req);

if (!errors.isEmpty()) {

return res.status(400).json({ errors: errors.array() });

}

try {

const claim = new Claim(req.body);

await claim.save();

res.status(201).send(claim);

} catch (error) {

res.status(400).send(error.message);

}

});

/\*\*

\* @swagger

\* /claims/{claim\_id}:

\* get:

\* summary: Retrieve a specific claim

\* tags: [Claims]

\* parameters:

\* - in: path

\* name: claim\_id

\* required: true

\* schema:

\* type: string

\* description: The ID of the claim to retrieve

\* responses:

\* 200:

\* description: A claim object

\* content:

\* application/json:

\* schema:

\* $ref: '#/components/schemas/Claim'

\* 404:

\* description: Claim not found

\*/

app.get('/claims/:claim\_id', async (req, res) => {

try {

const claim = await Claim.findOne({ claim\_id: req.params.claim\_id });

if (!claim) return res.status(404).send('Claim not found');

res.send(claim);

} catch (error) {

res.status(400).send(error.message);

}

});

/\*\*

\* @swagger

\* /claims/{claim\_id}:

\* put:

\* summary: Update a claim

\* tags: [Claims]

\* parameters:

\* - in: path

\* name: claim\_id

\* required: true

\* schema:

\* type: string

\* description: The ID of the claim to update

\* requestBody:

\* required: true

\* content:

\* application/json:

\* schema:

\* $ref: '#/components/schemas/Claim'

\* responses:

\* 200:

\* description: The updated claim

\* content:

\* application/json:

\* schema:

\* $ref: '#/components/schemas/Claim'

\* 404:

\* description: Claim not found

\*/

app.put('/claims/:claim\_id', async (req, res) => {

try {

const claim = await Claim.findOneAndUpdate({ claim\_id: req.params.claim\_id }, req.body, { new: true, runValidators: true });

if (!claim) return res.status(404).send('Claim not found');

res.send(claim);

} catch (error) {

res.status(400).send(error.message);

}

});

/\*\*

\* @swagger

\* /claims/{claim\_id}:

\* delete:

\* summary: Delete a claim

\* tags: [Claims]

\* parameters:

\* - in: path

\* name: claim\_id

\* required: true

\* schema:

\* type: string

\* description: The ID of the claim to delete

\* responses:

\* 200:

\* description: The claim was successfully deleted

\* content:

\* application/json:

\* schema:

\* type: object

\* properties:

\* message:

\* type: string

\* 404:

\* description: Claim not found

\*/

app.delete('/claims/:claim\_id', async (req, res) => {

try {

const claim = await Claim.findOneAndDelete({ claim\_id: req.params.claim\_id });

if (!claim) return res.status(404).send('Claim not found');

res.send({ message: 'Claim deleted successfully' });

} catch (error) {

res.status(400).send(error.message);

}

});

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {

console.log(`Server is running on port ${PORT}`);

});

We must navigate to http://localhost:3000/api-docs in the browser to see the Swagger UI with API documentation.

