**Route53 DNS Viewer - Project Documentation**

Table of Contents:

* Project Overview
* System Architecture
* Technology Stack
* Setup Instructions
* API Documentation
* Features
* File Structure
* Development Workflow
* Troubleshooting
* Deployment

1. **Project Overview**

Route53 DNS Viewer is a modern web application that provides a user-friendly interface for managing and viewing DNS records across AWS Route 53 hosted zones. The application features a serverless architecture using AWS Lambda and API Gateway, with an Angular frontend that communicates with AWS services through RESTful APIs.

* Key Benefits
  + Serverless Architecture: No server management required
  + Multi-Zone Support: View DNS records across all hosted zones or specific zones
  + Real-time Search: Instant filtering and search capabilities
  + Secure Access: IAM-based authentication through Lambda execution roles
  + Scalable: Automatically scales with AWS infrastructure

1. **System Architecture**

* High-Level Architecture

┌─────────────────┐ ┌──────────────────┐ ┌─────────────────┐

│ Angular │ │ API Gateway │ │ Lambda │

│ Frontend │───▶│ (eu-north-1) │───▶│ Functions │

└─────────────────┘ └──────────────────┘ └─────────────────┘

│ │

▼ ▼

┌──────────────────┐ ┌─────────────────┐

│ Route 53 │ │ CloudWatch │

│ DNS Service │ │ Logs │

└──────────────────┘ └─────────────────┘

* Component Details
  + Frontend (Angular)
    - Framework: Angular 17+ with standalone components
    - UI Components: Angular Material (tables, forms, dialogs)
    - State Management: RxJS observables and services
    - Routing: Angular Router with authentication guards
  + Backend (AWS Lambda)
    - Runtime: Node.js 22.x
    - Memory: 128MB (configurable)
    - Timeout: 3 seconds
    - IAM Role: `route53dnsviewer-role-0jtkvryg`
* API Gateway
  + Endpoint: `https://j125ycvo89.execute-api.eu-north-1.amazonaws.com/prod`
  + Region: eu-north-1
  + CORS: Enabled for cross-origin requests
  + Authentication: None (public endpoints)

1. **Technology Stack**

* Frontend
  + Angular: 17+ (latest)
  + TypeScript: 5.0+
  + Angular Material: UI component library
  + RxJS: Reactive programming
  + SCSS: Advanced CSS preprocessing
* Backend
  + AWS Lambda\*\*: Serverless compute
  + AWS API Gateway\*\*: REST API management
  + AWS SDK v3\*\*: JavaScript SDK for AWS services
  + Node.js\*\*: Runtime environment
* AWS Services
  + Route 53: DNS management
  + CloudWatch: Logging and monitoring
  + IAM: Identity and access management
  + S3: File storage (additional features)

1. **Setup Instructions**

* Prerequisites
  + Node.js 18+ installed
  + AWS CLI configured with appropriate permissions
  + AWS account with Route 53 access
* Frontend Setup
  + git clone <repository-url>
  + cd route53-dns-viewer
  + npm install
  + npm start
  + npm run build
* Backend Setup (AWS Lambda)
  + Create Lambda Function
    - zip -r lambda-function.zip index.js package.json
    - aws lambda create-function \

--function-name route53-dns-viewer \

--runtime nodejs22.x \

--role arn:aws:iam::<account-I d>:role/route53dnsviewer-role \

--handler index.handler \

--zip-file fileb://lambda-function.zip

* + Create API Gateway
    - aws apigateway create-rest-api \

--name "route53-dns-api" \

--description "Route53 DNS Viewer API"

* + Configure CORS and endpoints (Use AWS Console for easier configuration)
* IAM Role Setup

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"route53:ListHostedZones",

"route53:ListResourceRecordSets",

"route53:GetHostedZone"

],

"Resource": "\*"

},

{

"Effect": "Allow",

"Action": [

"logs:CreateLogGroup",

"logs:CreateLogStream",

"logs:PutLogEvents"

],

"Resource": "arn:aws:logs:\*:\*:\*"

}

]

}

1. **API Documentation**

* Base URL :<https://j125ycvo89.execute-api.eu-north-1.amazonaws.com/prod>
* Endpoints
* List Hosted Zones
  + GET /zones
  + Response:

[

{

"id": "Z1234567890",

"name": "example.com.",

"resourceRecordSetCount": 5

}

]

* List DNS Records
  + POST /records
  + Request Body:

{

"hostedZoneId": "Z1234567890",

"prefix": "www",

"allZones": false

}

* + Response:

[

{

"name": "www.example.com.",

"type": "A",

"ttl": 300,

"values": ["192.168.1.1"],

"hostedZoneName": "example.com."

}

]

* Delete DNS Record
  + DELETE /records/{id}
  + Parameters:
    - id: DNS record identifier

1. **Features**

* Core Features
  + Hosted Zone Management
    - List all hosted zones
    - View zone details and record counts
  + DNS Record Operations
    - List DNS records by zone
    - Search records by prefix
    - Multi-zone record search
    - Delete DNS records
  + Advanced Search
    - Prefix-based filtering
    - Cross-zone search capabilities
    - Real-time search results
  + User Interface
    - * Responsive design
      * Material Design components
      * Dynamic table columns
      * Export functionality
* Multi-Zone Search Feature
  + - Toggle Option: Checkbox to enable/disable multi-zone search
    - Zone Information: Displays hosted zone name for each record
    - Aggregated Results: Combines records from all zones
    - Performance: Optimized for large numbers of zones

1. **File Structure**

route53-dns-viewer/

├── src/

│ ├── app/

│ │ ├── components/

│ │ │ ├── dns-viewer/ # Main DNS viewer component

│ │ │ └── login/ # Authentication component

│ │ ├── guards/

│ │ │ └── auth.guard.ts # Route protection

│ │ ├── models/

│ │ │ └── dns-record.interface.ts # Data models

│ │ ├── services/

│ │ │ ├── auth.service.ts # Authentication logic

│ │ │ └── aws-route53.service.ts # AWS API integration

│ │ ├── app.component.ts # Root component

│ │ ├── app.config.ts # App configuration

│ │ └── app.routes.ts # Routing configuration

│ ├── main.ts # Application entry point

│ └── styles.scss # Global styles

├── angular.json # Angular CLI configuration

├── package.json # Dependencies and scripts

└── README.md # Project overview

1. **Development Workflow**

* Feature Development
  + git checkout -b feature/multi-zone-search
  + npm start
  + git add .
  + git commit -m "Add multi-zone search functionality"
  + git push origin feature/multi-zone-search
* Testing
  + npm test
  + npm run e2e
  + npm run lint
* Building
  + bash
  + npm run build
  + npm run build --prod
  + npm run build --stats-json

1. **Troubleshooting**

* Common Issues
  + Lambda Function Errors
    - Error: `require is not defined`
    - Solution: Use ES modules (`import`) instead of CommonJS (`require`)
    - Error: `AccessDenied` for Route 53
    - Solution: Check IAM role permissions and ensure Route 53 access
  + Frontend Issues
    - Error: `HttpClient not provided`
    - Solution: Ensure `provideHttpClient()` is in app.config.ts
    - Error: `MatCheckboxModule not found`
    - Solution: Import `MatCheckboxModule` in component imports
  + CORS Issues
    - Error: CORS policy violation
    - Solution: Verify API Gateway CORS configuration
* Debug Steps
  + Check CloudWatch logs for Lambda errors
  + Verify API Gateway endpoint configuration
  + Test API endpoints with Postman or curl
  + Check browser console for frontend errors
  + Verify IAM role permissions

1. **Deployment**

* Frontend Deployment
  + - npm run build
    - aws s3 sync dist/ s3://your-bucket-name --delete
* Backend Deployment
  + - aws lambda update-function-code \

--function-name route53-dns-viewer \

--zip-file fileb://lambda-function.zip

* + - Update API Gateway (if needed)
* Environment Configuration
  + - // src/environments/environment.ts

export const environment = {

production: false,

apiUrl: 'https://j125ycvo89.execute-api.eu-north-1.amazonaws.com/prod'

};

* + - // src/environments/environment.prod.ts

export const environment = {

production: true,

apiUrl: 'https://j125ycvo89.execute-api.eu-north-1.amazonaws.com/prod'

};

1. **Security Considerations**

* IAM Best Practices
  + Use least privilege principle
  + Regularly rotate access keys
  + Monitor CloudTrail logs
  + Use IAM roles instead of access keys
* API Security
  + Implement rate limiting
  + Add authentication if needed
  + Validate input parameters
  + Log all API calls
* Frontend Security
  + Sanitize user inputs
  + Implement proper error handling
  + Use HTTPS in production
  + Regular dependency updates

1. **Performance Optimization**

* Lambda Optimization
  + - Increase memory allocation for CPU-intensive operations
    - Use connection pooling for database connections
    - Implement caching strategies
    - Optimize cold start times
* Frontend Optimization
  + - Lazy load components
    - Implement virtual scrolling for large datasets
    - Use OnPush change detection strategy
    - Optimize bundle size

1. **Monitoring and Logging**

* CloudWatch Metrics
  + Lambda invocation count and duration
  + API Gateway request count and latency
  + Error rates and 4xx/5xx responses
* Application Logs
  + User actions and search queries
  + API response times
  + Error details and stack traces
* Alerts
  + High error rates
  + Lambda timeout occurrences
  + API Gateway throttling

1. **Future Enhancements**

* Planned Features
  + Bulk Operations: Delete multiple records at once
  + Record Templates: Predefined record configurations
  + Health Check Integration: Route 53 health check status
  + Multi-Account Support: Cross-account DNS management
  + Audit Trail: Track all DNS changes
* Technical Improvements
  + GraphQL API: More efficient data fetching
  + Real-time Updates: WebSocket integration
  + Offline Support: Service worker implementation
  + Mobile App: React Native or Flutter version