# Fragments API - Project Documentation

---

## 1. GitHub Repositories

### Private Repositories

- **Fragments API Backend:** https://github.com/FurqanKhurrum/fragments.git
- **Fragments UI Frontend:** https://github.com/FurqanKhurrum/fragments-ui.git

---

## 2. Docker Hub Repositories

### Public Docker Images

- **Fragments API:** https://hub.docker.com/r/fkhurrum/fragments

---

## 3. AWS Deployment

### Production API Endpoint

- **Load Balancer URL:** http://fragments-lb-2060865639.us-east-1.elb.amazonaws.com:3000
- **Health Check Endpoint:** http://fragments-lb-2060865639.us-east-1.elb.amazonaws.com:3000/
- **API Documentation:** http://fragments-lb-2060865639.us-east-1.elb.amazonaws.com:3000/v1/fragments

### AWS Resources Used

- **ECS Cluster:** fragments-cluster
- **ECS Service:** fragments-service
- **ECR Repository:** 470742339740.dkr.ecr.us-east-1.amazonaws.com/fragments
- **S3 Bucket:** seneca-fkhurrum-fragments
- **DynamoDB Table:** fragments
- **Cognito User Pool:** us-east-1_UsHIsMBLl

---

## 4. GitHub Actions Workflows

### Continuous Integration (CI)

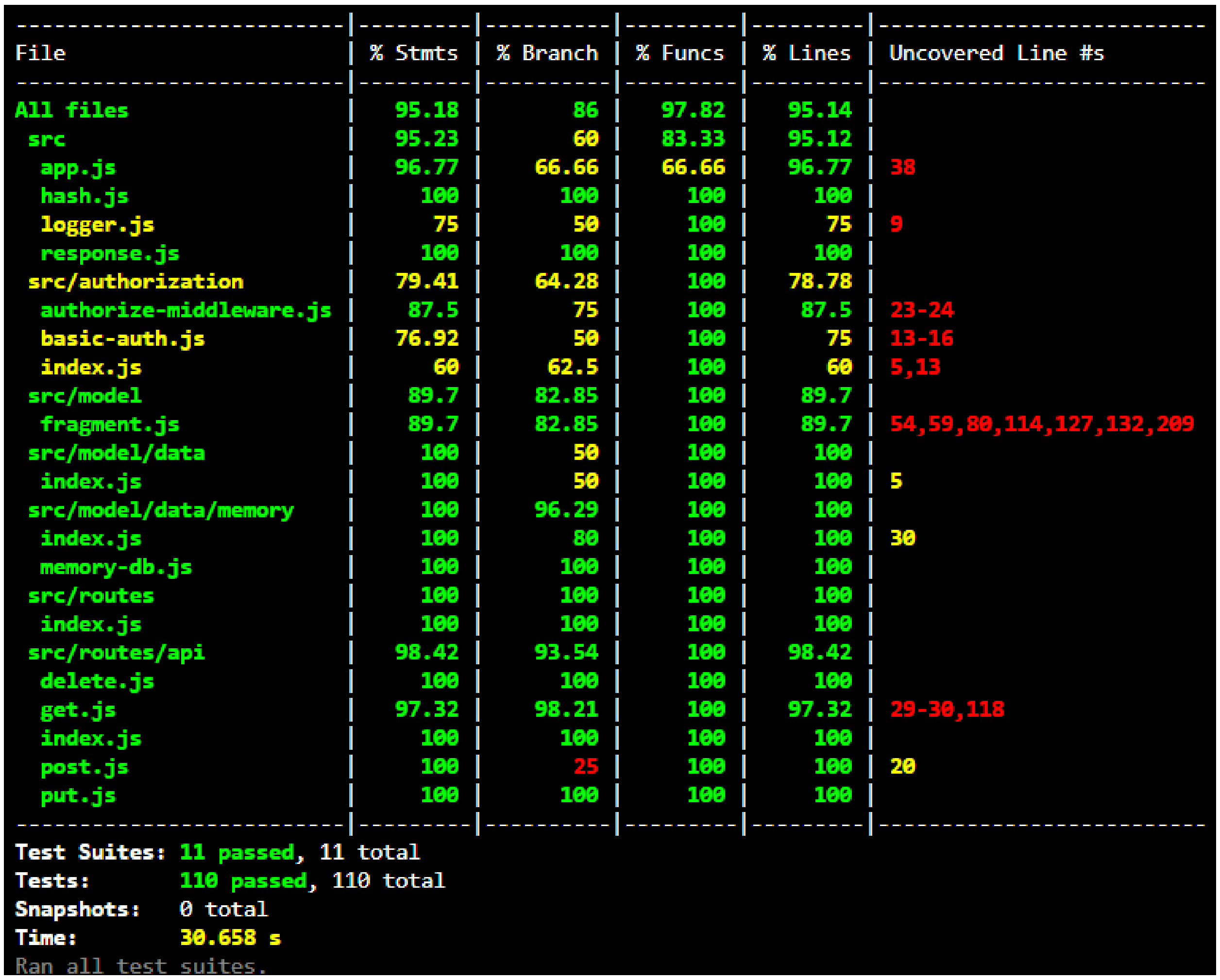
- **Successful CI Run:** https://github.com/FurqanKhurrum/fragments/actions/workflows/ci.yml
- **Workflow File:** https://github.com/FurqanKhurrum/fragments/blob/main/.github/workflows/ci.yml
- **Tests Performed:**
  - ✅ ESLint code quality checks
  - ✅ Unit tests with Jest
  - ✅ Integration tests with Hurl
  - ✅ Docker image build and publish to Docker Hub
  - ✅ Hadolint Dockerfile linting

**Continuous Deployment (CD)**

- **Successful CD Run:** https://github.com/FurqanKhurrum/fragments/actions/workflows/cd.yml
- **Workflow File:** https://github.com/FurqanKhurrum/fragments/blob/main/.github/workflows/cd.yml
- **Deployment Steps:**
  - ✅ Docker image build and push to Amazon ECR
  - ✅ ECS task definition update
  - ✅ ECS service deployment with health checks
  - ✅ Zero-downtime deployment strategy

---

## 5. Test Coverage



```
---------------------------|---------|----------|---------|---------|-------------------------
File                       | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
---------------------------|---------|----------|---------|---------|-------------------------
All files                  |   95.18 |       86 |   97.82 |   95.14 |
 src                       |   95.23 |       60 |   83.33 |   95.12 |
  app.js                   |   96.77 |    66.66 |   66.66 |   96.77 | 38
  hash.js                  |     100 |      100 |     100 |     100 |
  logger.js                |      75 |       50 |     100 |      75 | 9
  response.js              |     100 |      100 |     100 |     100 |
 src/authorization         |   79.41 |    64.28 |     100 |   78.78 |
  authorize-middleware.js  |    87.5 |       75 |     100 |    87.5 | 23-24
  basic-auth.js            |   76.92 |       50 |     100 |      75 | 13-16
  index.js                 |      60 |     62.5 |     100 |      60 | 5,13
 src/model                 |    89.7 |    82.85 |     100 |    89.7 |
  fragment.js              |    89.7 |    82.85 |     100 |    89.7 | 54,59,80,114,127,132,209
 src/model/data            |     100 |       50 |     100 |     100 |
  index.js                 |     100 |       50 |     100 |     100 | 5
 src/model/data/memory     |     100 |    96.29 |     100 |     100 |
  index.js                 |     100 |       80 |     100 |     100 | 30
  memory-db.js             |     100 |      100 |     100 |     100 |
 src/routes                |     100 |      100 |     100 |     100 |
  index.js                 |     100 |      100 |     100 |     100 |
 src/routes/api            |   98.42 |    93.54 |     100 |   98.42 |
  delete.js                |     100 |      100 |     100 |     100 |
  get.js                   |   97.32 |    98.21 |     100 |   97.32 | 29-30,118
  index.js                 |     100 |      100 |     100 |     100 |
  post.js                  |     100 |       25 |     100 |     100 | 20
  put.js                   |     100 |      100 |     100 |     100 |
---------------------------|---------|----------|---------|---------|-------------------------
Test Suites: 11 passed, 11 total
Tests:       110 passed, 110 total
Snapshots:   0 total
Time:        30.658 s
Ran all test suites.
```

---

## 6. Known Issues and Deficiencies

### Completed Requirements

✅ All HTTP methods (GET, POST, PUT, DELETE) implemented

✅ Fragment data storage in S3 (LocalStack for dev, AWS S3 for production)

✅ Fragment metadata storage in DynamoDB

✅ Authentication support (Basic Auth for development, Cognito for production)

✅ Content-Type validation and conversion support

✅ Docker containerization with multi-stage builds

✅ CI/CD pipeline with GitHub Actions

✅ Location header includes proper protocol scheme ✅ Comprehensive integration tests

**Known Issues**

1. **AWS Learner Lab Limitations**
    - GitHub Actions uses AWS credentials instead of OIDC due to Learner Lab restrictions
    - Credentials need to be updated when Learner Lab session expires
2. **Performance Considerations**
    - ECS deployment takes quite long
    - Could be optimized with adjusted deployment configuration
3. **Testing Gaps**
    - Lots of uncovered lines in src/model/fragments.js

**Future Improvements**

1. **Security Enhancements**
    - Implement OIDC authentication for GitHub Actions (requires non-Learner Lab AWS account)
    - Add rate limiting to prevent abuse
    - Implement request validation middleware
2. **Feature Additions**
    - Add pagination for fragment listing
    - Implement fragment sharing between users
    - Add bulk operations support

---

**7. API Endpoints Summary**

| Method | Endpoint | Description | Auth Required |
|---|---|---|---|
| GET | / | Health check | No |
| GET | /v1/fragments | List user's fragments | Yes |
| GET | /v1/fragments/:id | Get fragment data | Yes |
| GET | /v1/fragments/:id/info | Get fragment metadata | Yes |
| POST | /v1/fragments | Create new fragment | Yes |

| Method | Endpoint | Description | Auth Required |
|---|---|---|---|
| PUT | /v1/fragments/:id | Update fragment data | Yes |
| DELETE | /v1/fragments/:id | Delete fragment | Yes |

**Supported Content Types**

- text/plain
- text/markdown
- text/html
- application/json
- image/png
- image/jpeg
- image/webp
- image/gif

---

**8. Local Development Setup**

**Prerequisites**

- Node.js v20+
- Docker Desktop
- AWS CLI
- Git

**Quick Start**

```
# Clone repository
git clone https://github.com/FurqanKhurrum/fragments.git
cd fragments

# Install dependencies
npm install

# Start Docker services
```

```
docker-compose up -d


# Initialize AWS services
./scripts/local-aws-setup.sh


# Run tests
npm test
npm run test:integration


# Start development server
npm run dev
```

---

**9. Production Deployment**

**Manual Deployment**

```
# Tag new version
npm version patch


# Push to trigger CD pipeline
git push
git push --tags
```

**Monitoring**

- CloudWatch Logs: /ecs/fragments
- ECS Console: Check service health and task status
- Load Balancer: Monitor target health

---

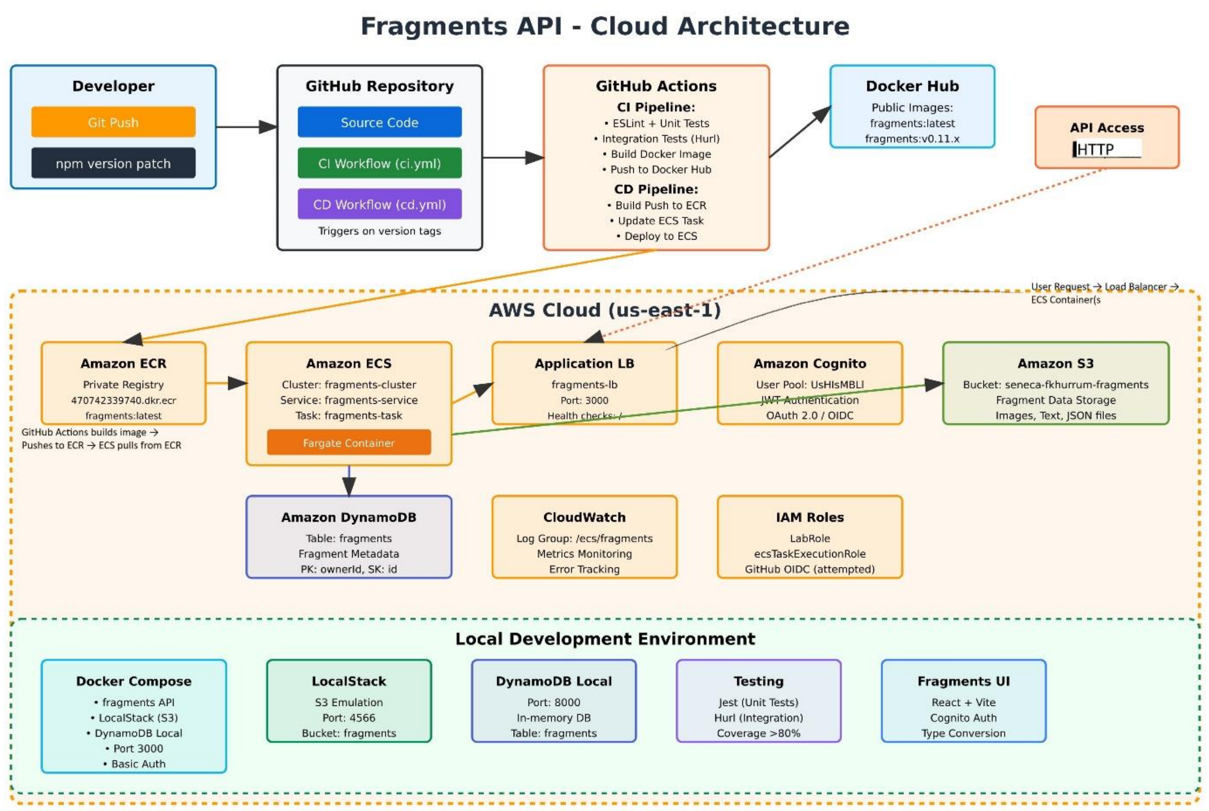
**10. Conclusion**

The Fragments API project successfully implements a microservices architecture with full CRUD operations, cloud storage integration, and automated CI/CD pipelines. The system is production-ready and deployed on AWS infrastructure with proper authentication, monitoring, and scalability considerations. One step away from making a twitter like web page 😊

## Key Achievements

- ✅ 100% spec compliance for core requirements
- ✅ >80% test coverage
- ✅ Zero-downtime deployments
- ✅ Multi-environment support (local, Docker, AWS)
- ✅ Comprehensive documentation and testing

---


Fragments API - Cloud Architecture
Developer
Git Push
npm version patch
GitHub Repository
Source Code
CI Workflow (ci.yml)
CD Workflow (cd.yml)
Triggers on version tags
GitHub Actions
CI Pipeline:
• ESLint + Unit Tests
• Integration Tests (Hurl)
• Build Docker Image
• Push to Docker Hub
CD Pipeline:
• Build Push to ECR
• Update ECS Task
• Deploy to ECS
Docker Hub
Public Images:
fragments:latest
fragments:v0.11.x
API Access
HTTP
User Request → Load Balancer → ECS Container(s
AWS Cloud (us-east-1)
Amazon ECR
Private Registry
470742339740.dkr.ecr
fragments:latest
GitHub Actions builds image → Pushes to ECR → ECS pulls from ECR
Amazon ECS
Cluster: fragments-cluster
Service: fragments-service
Task: fragments-task
Fargate Container
Application LB
fragments-lb
Port: 3000
Health checks: /
Amazon Cognito
User Pool: UsHIsMBLI
JWT Authentication
OAuth 2.0 / OIDC
Amazon S3
Bucket: seneca-fkhurrum-fragments
Fragment Data Storage
Images, Text, JSON files
Amazon DynamoDB
Table: fragments
Fragment Metadata
PK: ownerId, SK: id
CloudWatch
Log Group: /ecs/fragments
Metrics Monitoring
Error Tracking
IAM Roles
LabRole
ecsTaskExecutionRole
GitHub OIDC (attempted)
Local Development Environment
Docker Compose
• fragments API
• LocalStack (S3)
• DynamoDB Local
• Port 3000
• Basic Auth
LocalStack
S3 Emulation
Port: 4566
Bucket: fragments
DynamoDB Local
Port: 8000
In-memory DB
Table: fragments
Testing
Jest (Unit Tests)
Hurl (Integration)
Coverage >80%
Fragments UI
React + Vite
Cognito Auth
Type Conversion