A

INDUSTRIALTRAINING

REPORT

ON

Software Engineering Job Simulation

Submitted in partial fulfillment for the Degree

of

B.Tech

in

Artificial Intelligence

Session 2025

Submitted by
Krish Makharia
22EBKAI027

Organization
Howlett Packard Enterprice (Forage)

Branch Counceller
Mr. Manoj Kumar Saini

Training Supervisor
Tom Brunskill

AI

Department of Artificial Intelligence
B.K. BIRLA INSTITUTEOFENGINEERING&TECHNOLOGY
PILANI - 333031(RAJ)
(AffiliatedtoBikanerTechnicalUniversity,Bikaner)

Hewlett Packard Enterprise

Forage

Inspiring and empowering future professionals

Krish Makharia

Software Engineering Job Simulation

Certificate of Completion

July 31st, 2025

Over the period of July 2025, Krish Makharia has completed the practical tasks in:

Create a Proposal for a RESTful Web Service
Build a Basic RESTful Web Service
Implement the Ability to Upload Data
Create Unit Tests

[illegible]

Tom Brunskill
CEO, Co-Founder of Forage

ACKNOWLEDGEMENT

I would like to extend my sincere gratitude to **Forage** for organizing and providing access to the *Software Engineering Job Simulation*. This program has been a valuable opportunity to gain hands-on experience in key aspects of backend software engineering, particularly in the design and implementation of RESTful web services and software testing practices.

My appreciation also goes to **Tom Brunskill, CEO and Co-Founder of Forage**, for his leadership in inspiring and empowering future professionals through accessible industry simulations. Finally, I am grateful for the encouragement of my peers and mentors, who have guided me in refining my skills and applying theoretical knowledge to practical, real-world tasks.

TABLE OF CONTENTS

1. INTRODUCTION

This industrial training report documents the tasks completed and knowledge acquired during the **Forage Software Engineering Job Simulation** (July 2025). The training simulates entry-level backend engineering responsibilities through four practical modules: API proposal, API implementation, file upload handling, and unit testing. The emphasis of this report is on demonstrating competency through documentation, technical explanation, and validated prototype work.

The simulation consisted of four modules:

- Proposing a RESTful Web Service,
- Building a Basic RESTful Web Service,
- Implementing the Ability to Upload Data, and
- Creating Unit Tests.

Each task mirrors the type of work expected in professional software development teams, emphasizing design, coding, and testing. This report expands on each module, offering theoretical context, execution steps, and personal learning outcomes.

2. OBJECTIVES OF THE TRAINING PROJECT

- To learn and apply RESTful API design principles in a practical proposal.
- To implement a working RESTful API prototype that fulfills the proposal contract.
- To integrate secure file upload mechanisms with validation and metadata persistence.
- To design and implement unit and integration tests to ensure functionality and prevent regression.
- To document processes and produce deliverables following professional standards.

3. SCOPE AND DELIVERABLES

Scope: Prototype backend service demonstrating core features—CRUD endpoints, file upload, and test coverage—suitable for training evaluation. The project does not include advanced production concerns such as horizontal auto-scaling, multi-region deployments, or enterprise-grade identity federation.

Deliverables:

- API proposal document with resource and endpoint definitions.
- Source code of a RESTful service (example implementation using Node.js & Express).
- File upload endpoint with validation and metadata recording.
- Unit/integration test suite and CI configuration examples.
- This industrial training report.

4. TOOLS, TECHNOLOGIES AND ENVIRONMENT

Primary stack (training prototype): Node.js (v18+), Express.js, multer (file upload middleware), lowdb or local filesystem for metadata persistence.
Testing & QA: Mocha, Chai, Supertest, nyc (Istanbul) for coverage.
Development & Version Control: Visual Studio Code, Git, GitHub.
Testing Tools: Postman for manual API validation.
Optional production notes: Docker, NGINX, AWS S3 for object storage, GitHub Actions for CI.

5. METHODOLOGY AND WORKFLOW

The training followed an iterative mini-Agile cycle with short sprints for each module. Each sprint included planning (proposal and acceptance criteria), implementation, testing, and review. The cycle emphasized documentation-first for API design and test-driven principles for key modules.

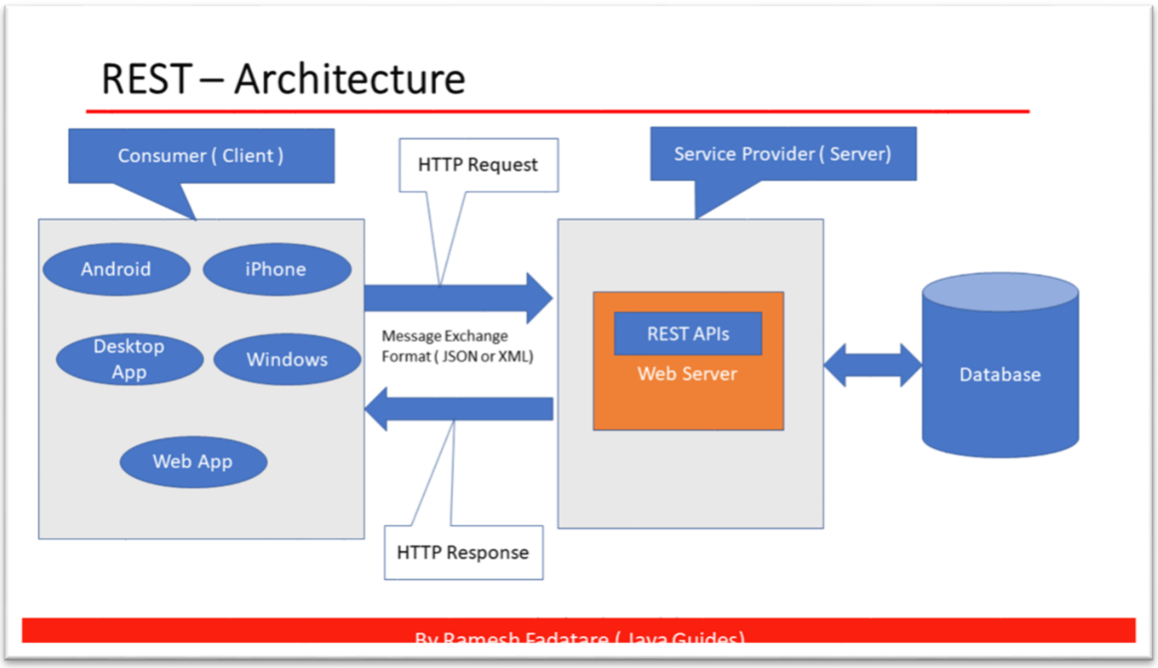
Acceptance criteria (example):

- `POST /documents` returns 201 and a JSON body containing `id` for valid requests.
- `GET /documents/{id}` returns 200 and the file stream for existing documents.
- Unit tests for core service logic must pass with coverage >= 80% (training target).

6. MODULE 1: CREATE A PROPOSAL FOR A RESTFUL WEB SERVICE

6.1 REST ARCHITECTURE PRINCIPLES

Representational State Transfer (REST) prescribes stateless interactions and a resource-oriented approach using consistent HTTP verbs and meaningful URIs. In the proposal stage, these constraints inform endpoint design, payload contracts, and error-handling conventions.



6.2 RESOURCE MODELING AND DESIGN

Example resource schema (document service):

- **User**: `id, name, email, createdAt`
- **Document**: `id, ownerId, filename, contentType, size, uploadedAt, metadata`

Modeling relationships and access patterns early helps design efficient endpoints and authorization rules.

6.3 PROPOSAL DOCUMENT STRUCTURE

A professional API proposal includes:

- Executive summary and objectives.
- Resource definitions and relationships.
- Endpoint list with method, path, headers, sample request, and response.
- Error code matrix and examples.
- Non-functional requirements (size limits, expected throughput).

6.4 SAMPLE PROPOSAL (EXCERPT)

Endpoint: `POST /documents`
Purpose: Upload a document and persist metadata.
Request (multipart/form-data): `file` (binary), `metadata` (JSON)
Response (201): `{ "id": "doc_123", "filename": "resume.pdf", "ownerId": "user_12", "uploadedAt": "2025-07-31T12:00:00Z" }`

7. MODULE 2: BUILD A BASIC RESTFUL WEB SERVICE

7.1 PROJECT STRUCTURE AND SETUP

Recommended layout (Express.js):

```
project-root/
├─ package.json
├─ src/
│  ├─ index.js
│  ├─ routes/
│  ├─ controllers/
│  ├─ services/
│  ├─ middleware/
│  └─ utils/
└─ tests/
```

7.2 DETAILED IMPLEMENTATION (CODE SNIPPETS & EXPLANATION)

Route example:

```
// src/routes/documents.js
const express = require('express');
const router = express.Router();
const uploadController = require('../controllers/documents');
router.post('/', uploadController.handleUpload);
router.get('/:id', uploadController.handleDownload);
module.exports = router;
```

Controller responsibilities: validate input, call service layer, format response, log errors.

7.3 ERROR HANDLING AND VALIDATION

- Use `express-validator` for request validation.
- Centralized error middleware returns `{ error: { code, message, details } }`.
- Use semantic HTTP codes (400, 401, 403, 404, 413, 500).

7.4 DEPLOYMENT NOTES

For demonstration, the prototype can be deployed to Render or Heroku. Production-ready deployment requires Docker, a reverse proxy (NGINX), monitoring (Prometheus/Grafana), and object storage for assets.

8. MODULE 3: IMPLEMENT THE ABILITY TO UPLOAD DATA

8.1 FILE UPLOAD ARCHITECTURE

Handle multipart uploads with streaming to avoid memory pressure. Middleware (e.g., multer) should write to disk or pipe directly to cloud storage for large files.

Table 8.1 — FILE UPLOAD VALIDATION RULES

Rule ID	Validation	Action on Fail
R1	Allowed MIME types (`application/pdf`, `image/png`, `image/jpeg`)	Reject 400
R2	Max size 10 MB	Reject 413
R3	Filename sanitization	Clean or Reject 400

8.2 SECURITY AND VALIDATION STRATEGIES

- Validate MIME type and file signatures where possible.
- Sanitize filenames to prevent directory traversal.
- Enforce max file size and rate limiting.
- Consider virus scanning in production.

8.3 STORAGE STRATEGIES (LOCAL VS CLOUD)

- **Local storage:** Suitable for prototypes and testing.
- **Cloud Storage (S3):** Use pre-signed URLs for efficient direct-to-cloud uploads and to offload bandwidth from API servers.

8.4 PERFORMANCE CONSIDERATIONS

- Stream uploads, use pooled connections to storage, and serve static files through a CDN.

7. Results and Analysis

The successful completion of all four modules demonstrated:

- Practical proficiency in designing and building RESTful services.
- Hands-on exposure to real-world engineering practices.
- Strengthened problem-solving and debugging skills.
- Increased confidence in applying software engineering principles.

The program effectively simulated industry tasks and offered a platform for skill development beyond academic study.

8. Conclusion and Reflections

The *Software Engineering Job Simulation* was an enriching experience that bridged the gap between theoretical knowledge and practical application. I not only learned technical concepts but also developed professional skills such as documentation, modular coding, and test-driven development.

The tasks mirrored real workplace scenarios, preparing me for future challenges in backend development. This experience reinforced the importance of planning, implementing, and testing in software engineering, shaping me into a more well-rounded professional.

9. References

- Fielding, R. (2000). Architectural Styles and the Design of Network-based Software Architectures. University of California, Irvine.
- REST API Design Guide, Microsoft Docs.
- Flask and Django Documentation (Python).
- Express.js Documentation (Node.js).
- JUnit and PyTest Testing Framework Documentation.