Data Structure Visualizer

Part 2: Implementation Guide

Code, APIs, and Advanced Features

Backend Code
Frontend Design
Deploy & Scale

Continuation from Part 1 | Version 1.1.0

Contents

1 Backend Implementation

1.1 Core Service Layer

FileStorageService.java

```java
@Service
public class FileStorageService {
    private static final long MAX_STORAGE = 50 * 1024 * 1024; // 50MB
    private FileHashMap fileHashMap;
    private Map<String, Long> userStorage;

    public FileStorageService() {
        this.fileHashMap = new FileHashMap();
        this.userStorage = new HashMap<>();
    }

    public Map<String, Object> uploadFile(
        MultipartFile file,
        String userId,
        String structureType
    ) throws IOException {
        // Check quota
        long currentUsage = userStorage.getOrDefault(userId, 0L);
        if (currentUsage + file.getSize() > MAX_STORAGE) {
            throw new QuotaExceededException("Storage limit exceeded!");
        }

        // Save file
        String fileName = UUID.randomUUID() + "_" + file.getOriginalFilenam
        Path filePath = Paths.get("uploads/" + fileName);
        Files.copy(file.getInputStream(), filePath);

        // Create stored file object
        StoredFile storedFile = new StoredFile(
            file.getOriginalFilename(),
            file.getContentType(),
            file.getSize(),
            filePath.toString(),
            userId
        );

        // Update quota
        userStorage.put(userId, currentUsage + file.getSize());

        // Store in structure
        return fileHashMap.put(storedFile);
    }
}
```

1.2 HashMap Data Structure

FileHashMap.java

```
public class FileHashMap {
    private Map<String, List<StoredFile>> buckets;
    private static final int BUCKET_COUNT = 8;
    private static final double START_X = 100;
    private static final double SPACING = 120;

    public FileHashMap() {
        this.buckets = new HashMap<>();
        for (int i = 0; i < BUCKET_COUNT; i++) {
            buckets.put("bucket_" + i, new ArrayList<>());
        }
    }

    public Map<String, Object> put(StoredFile file) {
        // Calculate bucket index
        int bucketIndex = Math.abs(
            file.getId().hashCode() % BUCKET_COUNT
        );
        String bucketKey = "bucket_" + bucketIndex;

        // Get bucket
        List<StoredFile> bucket = buckets.get(bucketKey);
        bucket.add(file);

        // Calculate visual position
        file.setX(START_X + bucketIndex * SPACING);
        file.setY(150 + bucket.size() * 60);

        // Create response
        Map<String, Object> response = new HashMap<>();
        response.put("operation", "put");
        response.put("file", file);
        response.put("bucketIndex", bucketIndex);
        response.put("buckets", convertBucketsToDTO());
        response.put("timestamp", System.currentTimeMillis());

        return response;
    }

    private Map<String, List<Map<String, Object>>>
        convertBucketsToDTO() {
        Map<String, List<Map<String, Object>>> result = new HashMap<>();

        buckets.forEach((key, files) -> {
            List<Map<String, Object>> fileList = new ArrayList<>();
            files.forEach(file -> {
                Map<String, Object> fileData = new HashMap<>();
                fileData.put("id", file.getId());
                fileData.put("originalFileName", file.getOriginalFileName(
                fileData.put("fileSize", file.getFileSize());
```

1.3 REST API Controller

FileStorageController.java

```java
@RestController
@RequestMapping("/api/files")
@CrossOrigin(origins = "*")
public class FileStorageController {

    @Autowired
    private FileStorageService fileStorageService;

    @Autowired
    private SimpMessagingTemplate messagingTemplate;

    @PostMapping("/upload")
    public ResponseEntity<Map<String, Object>> uploadFile(
        @RequestParam("file") MultipartFile file,
        @RequestParam(value = "userId", defaultValue = "demo-user") String
        @RequestParam(value = "structureType", defaultValue = "hashmap")
            String structureType
    ) {
        try {
            Map<String, Object> response =
                fileStorageService.uploadFile(file, userId, structureType)

            // Broadcast via WebSocket
            messagingTemplate.convertAndSend("/topic/file-updates", respons

            return ResponseEntity.ok(response);
        } catch (Exception e) {
            return ResponseEntity.badRequest()
                .body(Map.of("error", e.getMessage()));
        }
    }

    @GetMapping("/download/{fileId}")
    public ResponseEntity<ByteArrayResource> downloadFile(
        @PathVariable String fileId
    ) {
        try {
            byte[] data = fileStorageService.downloadFile(fileId);
            ByteArrayResource resource = new ByteArrayResource(data);

            return ResponseEntity.ok()
                .contentType(MediaType.APPLICATION_OCTET_STREAM)
                .header(HttpHeaders.CONTENT_DISPOSITION,
                    "attachment; filename=\"" + fileId + "\"")
                .body(resource);
        } catch (Exception e) {
            return ResponseEntity.notFound().build();
        }
    }
```

2 Frontend Visualization

2.1 Canvas Visualizer Class

file-visualizer.js

```
class FileStorageVisualizer {
    constructor(canvas, structureType) {
        this.canvas = canvas;
        this.ctx = canvas.getContext('2d');
        this.structureType = structureType;
        this.files = [];
        this.animationSpeed = 1;
    }

    async handleFileUpdate(data) {
        const { operation, file, buckets } = data;

        if (operation === 'put' && file) {
            await this.animateFileDrop(file);
        }

        this.buckets = buckets;
        this.render();
    }

    async animateFileDrop(file) {
        const startY = -50;
        const targetY = file.y;
        const duration = 1000 / this.animationSpeed;
        const startTime = Date.now();

        return new Promise(resolve => {
            const animate = () => {
                const elapsed = Date.now() - startTime;
                const progress = Math.min(elapsed / duration, 1);

                // Bounce easing
                const eased = this.easeOutBounce(progress);
                file.y = startY + (targetY - startY) * eased;

                this.render();

                if (progress < 1) {
                    requestAnimationFrame(animate);
                } else {
                    resolve();
                }
            };
            animate();
        });
    }

    easeOutBounce(t) {
        const n1 = 7.5625;
        const d1 = 2.75;
```

3 API Reference

3.1 REST Endpoints

Method	Endpoint	Description
POST	/api/files/upload	Upload new file
GET	/api/files/download/{id}	Download file
DELETE	/api/files/{id}	Delete file
GET	/api/files/storage/{userId}	Get storage info

Table 1: Complete API Endpoints

3.2 Request/Response Examples

Upload Request

```
POST /api/files/upload
Content-Type: multipart/form-data

Parameters:
  file: [binary data]
  userId: demo-user
  structureType: hashmap
```

Response JSON


```
{
  "operation": "put",
  "file": {
    "id": "abc123-456def",
    "originalFileName": "photo.jpg",
    "fileSize": 524288,
    "formattedSize": "512 KB",
    "fileType": "image/jpeg",
    "icon": "",
    "color": "#e74c3c",
    "x": 220,
    "y": 210
  },
  "bucketIndex": 3,
  "storageUsed": 1048576,
  "storageLimit": 52428800,
  "storagePercentage": 2.0,
  "timestamp": 1703001234567
}
```

4 WebSocket Setup

4.1 Backend Configuration

WebSocketConfig.java

```java
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements
    WebSocketMessageBrokerConfigurer {

    @Override
    public void configureMessageBroker(
        MessageBrokerRegistry config
    ) {
        config.enableSimpleBroker("/topic");
        config.setApplicationDestinationPrefixes("/app");
    }

    @Override
    public void registerStompEndpoints(
        StompEndpointRegistry registry
    ) {
        registry.addEndpoint("/ws")
            .setAllowedOrigins("*")
            .withSockJS();
    }
}
```

4.2 Frontend Client

websocket-client.js

```
class WebSocketClient {
    constructor(onMessageCallback) {
        this.onMessageCallback = onMessageCallback;
    }

    connect() {
        const socket = new SockJS('http://localhost:8080/ws');
        this.stompClient = Stomp.over(socket);

        this.stompClient.connect({}, (frame) => {
            console.log('Connected: ' + frame);

            this.stompClient.subscribe('/topic/file-updates', (message) =>
                const data = JSON.parse(message.body);
                this.onMessageCallback(data);
            });
        });
    }
}

// Usage
const wsClient = new WebSocketClient((data) => {
    visualizer.handleFileUpdate(data);
});
wsClient.connect();
```

5 Security & Deployment

5.1 Security Best Practices

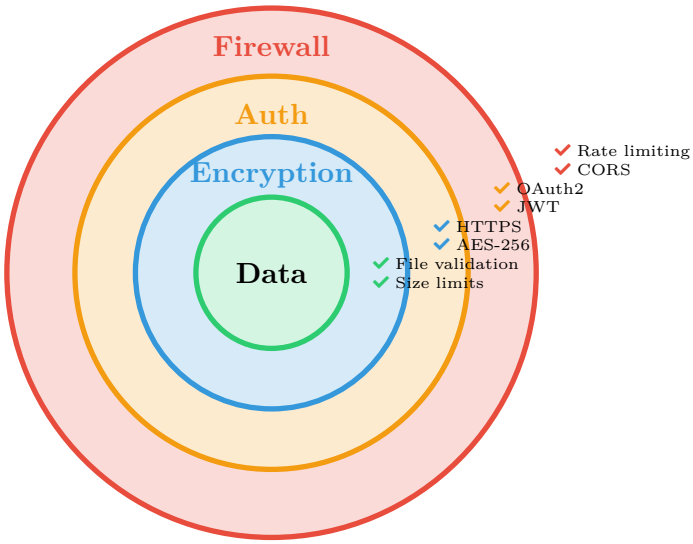

Figure 1: Multi-Layer Security Architecture

5.2 Production Checklist

Deployment Checklist

Before going live:

- ☑ Enable HTTPS/TLS
- ☑ Add authentication (OAuth2)
- ☑ Implement rate limiting
- ☑ Validate file types
- ☑ Set up monitoring
- ☑ Configure backups
- ☑ Test error handling
- ☑ Review CORS settings

5.3 Docker Deployment

Dockerfile

```
FROM openjdk:17-jdk-slim
WORKDIR /app

# Copy application
COPY target/visualizer-1.1.0.jar app.jar

# Create uploads directory
RUN mkdir -p /uploads

# Expose port
EXPOSE 8080

# Run application
ENTRYPOINT ["java", "-jar", "app.jar"]
```

docker-compose.yml

```
version: '3.8'

services:
  backend:
    build: ./backend
    ports:
      - "8080:8080"
    volumes:
      - ./uploads:/uploads
    environment:
      - SPRING_PROFILES_ACTIVE=production

  frontend:
    image: nginx:alpine
    ports:
      - "80:80"
    volumes:
      - ./frontend:/usr/share/nginx/html
```

6 Troubleshooting Guide

6.1 Common Issues

Problem	Solution
WebSocket fails	Check CORS settings, verify port 8080 is open
Upload rejected	Verify file size < 10MB, check quota
Canvas blank	Clear browser cache, check console for errors
Slow performance	Reduce animation speed, limit file count
Build errors	Run `mvn clean install`, check Java version

Table 2: Quick Troubleshooting Reference

6.2 Debug Commands

Useful Commands

```
# Check Java version
java -version

# Test backend
curl http://localhost:8080/health

# View logs
tail -f logs/visualizer.log

# Check WebSocket
wscat -c ws://localhost:8080/ws

# Enable debug mode (Browser)
Press Ctrl+D or Cmd+D
```

7 Summary

Complete Implementation Guide

Part 2 Covered:

Backend:
- FileStorageService
- HashMap implementation
- REST API endpoints
- WebSocket config

Frontend:
- Canvas visualizer
- Animation engine
- WebSocket client
- File rendering

Advanced Topics:
- API documentation with examples
- Security best practices
- Docker deployment
- Troubleshooting guide

You're Ready to Deploy!

7.1 Next Steps

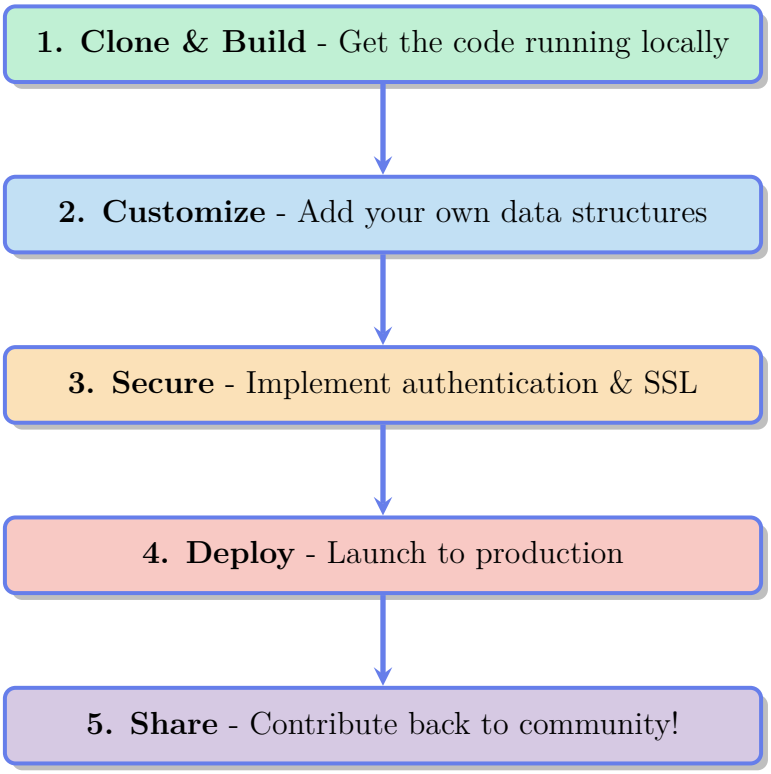

Figure 2: Your Path Forward

Resource	Link
Source Code	`github.com/yourname/ds-visualizer`
Full Docs	`docs.dsvisualizer.com`
Video Tutorials	`youtube.com/@dsvisualizer`
Discord Community	`discord.gg/dsvisualizer`
Support Email	`support@dsvisualizer.com`
Report Issues	`github.com/yourname/ds-visualizer/issues`

7.2 Resources & Support

Thank You!

For building with Data Structure Visualizer

Happy visualizing and teaching!