

■ Policy Toggle Service - Frontend Code Export

Total files: 51 | JavaScript: 38 | JSON: 3 | Markdown: 5 | Docker: 4

■ Frontend Files:

```
[DOCKER] .dockerignore
[JSON] .eslintrc.json
[JS] .gitignore
[DOCKER] Dockerfile
[MD] PROJECT_SUMMARY.md
[MD] README.md
[DOCKER] docker-compose.test.yml
[DOCKER] docker-compose.yml
[MD] docs\API.md
[MD] docs\QUICKSTART.md
[MD] docs\TEST_RESULTS.md
[JSON] package.json
[JSON] postman_collection.json
[JS] scripts\generate-test-docs.js
[JS] scripts\seed.js
[JS] src\app.js
[JS] src\config\database.js
[JS] src\config\index.js
[JS] src\index.js
[JS] src\middleware\auth.js
[JS] src\middleware\authorization.js
[JS] src\middleware\errorHandler.js
[JS] src\middleware\featureToggle.js
[JS] src\middleware\index.js
[JS] src\middleware\rateGuard.js
[JS] src\middleware\requestLogger.js
[JS] src\middleware\validation.js
[JS] src\models\AuditLog.js
[JS] src\models\FeatureToggle.js
[JS] src\models\RateGuard.js
[JS] src\models\User.js
[JS] src\models\index.js
[JS] src\routes\audit.routes.js
[JS] src\routes\auth.routes.js
[JS] src\routes\features.routes.js
[JS] src\routes\index.js
[JS] src\routes\rateGuards.routes.js
[JS] src\routes\system.routes.js
[JS] src\routes\users.routes.js
[JS] src\services\auditService.js
[JS] src\services\authService.js
[JS] src\services\featureToggleService.js
[JS] src\services\index.js
[JS] src\services\rateGuardService.js
[JS] src\services\userService.js
[JS] src\utils\logger.js
[JS] tests\helpers.js
[JS] tests\integration\auth.routes.test.js
[JS] tests\setup.js
[JS] tests\system\featureToggle.flow.test.js
[JS] tests\unit\user.model.test.js
```

■ File: .dockerignore

```
=====
1: # Dependencies
2: node_modules
3: npm-debug.log
4: yarn-error.log
5:
6: # Testing
7: coverage
8: *.test.js
9: tests
10:
11: # Documentation
12: *.md
13: docs
14:
15: # Git
16: .git
17: .gitignore
18:
19: # Environment
20: .env
21: .env.local
22: .env.*.local
23:
24: # Logs
25: logs
26: *.log
27:
28: # IDE
29: .vscode
30: .idea
31: *.swp
32: *.swo
33: *~
34:
35: # OS
36: .DS_Store
37: Thumbs.db
38:
39: # Docker
40: Dockerfile
41: docker-compose*.yaml
42: .dockerignore
-----
```

■ File: .eslintrc.json

```
=====
1: {
2:   "env": {
3:     "node": true,
4:     "es2021": true,
5:     "jest": true
6:   },
7:   "extends": ["eslint:recommended"],
8:   "parserOptions": {
9:     "ecmaVersion": 2021,
10:    "sourceType": "module"
11:  },
12:  "rules": {
13:    "indent": ["error", 2],
14:    "linebreak-style": ["error", "unix"],
15:    "quotes": ["error", "single"],
16:    "semi": ["error", "always"],
17:    "no-unused-vars": ["warn", { "argsIgnorePattern": "^_" }],
18:    "no-console": "off",
19:    "prefer-const": "error",
20:    "arrow-spacing": "error",
21:    "comma-dangle": ["error", "never"],
22:    "eol": ["error", "always"],
23:    "no-var": "error"
24:  }
}
```

```
25: }
```

■ File: .gitignore

```
1: # Dependencies
2: node_modules/
3: package-lock.json
4: yarn.lock
5:
6: # Environment variables
7: .env
8: .env.local
9: .env.*.local
10: .env.production
11:
12: # Logs
13: logs/
14: *.log
15: npm-debug.log*
16: yarn-debug.log*
17: yarn-error.log*
18: lerna-debug.log*
19:
20: # Runtime data
21: pids
22: *.pid
23: *.seed
24: *.pid.lock
25:
26: # Testing
27: coverage/
28: .nyc_output/
29: *.lcov
30:
31: # Build outputs
32: dist/
33: build/
34: out/
35:
36: # IDE
37: .vscode/
38: .idea/
39: *.swp
40: *.swo
41: *~
42: .DS_Store
43:
44: # OS
45: Thumbs.db
46: .DS_Store
47:
48: # Temporary files
49: *.tmp
50: *.temp
51: .cache/
52:
53: # Database
54: *.db
55: *.sqlite
56: *.sqlite3
57: *.py
58:
59: # Docker
60: .dockerignore
61: docker-compose.override.yml
62:
63: # Documentation (auto-generated)
64: docs/TEST_RESULTS.md
```

■ File: Dockerfile

```
=====
1: # Multi-stage build for optimized production image
2:
3: # Stage 1: Dependencies
4: FROM node:18-alpine AS dependencies
5: WORKDIR /app
6:
7: # Copy package files
8: COPY package*.json ./
9:
10: # Install production dependencies only
11: RUN npm ci --only=production
12:
13: # Stage 2: Build
14: FROM node:18-alpine AS build
15: WORKDIR /app
16:
17: # Copy package files
18: COPY package*.json ./
19:
20: # Install all dependencies (including dev)
21: RUN npm ci
22:
23: # Copy source code
24: COPY . .
25:
26: # Stage 3: Production
27: FROM node:18-alpine AS production
28: WORKDIR /app
29:
30: # Create non-root user
31: RUN addgroup -g 1001 -S nodejs && \
32:     adduser -S nodejs -u 1001
33:
34: # Copy production dependencies
35: COPY --from=dependencies /app/node_modules ./node_modules
36:
37: # Copy application code
38: COPY --chown=nodejs:nodejs . .
39:
40: # Create logs directory
41: RUN mkdir -p logs && chown nodejs:nodejs logs
42:
43: # Switch to non-root user
44: USER nodejs
45:
46: # Expose port
47: EXPOSE 3000
48:
49: # Health check
50: HEALTHCHECK --interval=30s --timeout=3s --start-period=40s --retries=3 \
51:     CMD node -e "require('http').get('http://localhost:3000/api/health', (r) => {process.exit(r.statusCode === 200 ? 0 : 1)})"
52:
53: # Start application
54: CMD ["node", "src/index.js"]
=====
```

■ File: PROJECT_SUMMARY.md

```
=====
1: # Project Summary: Policy-Driven Feature Toggle & Rate Guard Service
2:
3: ## Project Overview
4: A **production-grade backend system** demonstrating dynamic feature access control and API rate limiting through a policy-driven architecture.
5:
6: ---
7:
8: ## Project Statistics
9:
10: ### Code Metrics
11: - **Total Files**: 48+
12: - **Source Files**: 30+
```



```

86: ? ? ??? user.model.test.js # Model unit tests
87: ? ?
88: ? ??? integration/
89: ? ? ??? auth.routes.test.js # API integration tests
90: ? ?
91: ? ??? system/
92: ? ? ??? featureToggle.flow.test.js # E2E system tests
93: ? ?
94: ? ??? setup.js # Test environment setup
95: ? ??? helpers.js # Test helper utilities
96: ?
97: ??? scripts/ # Utility scripts
98: ? ??? generate-test-docs.js # Auto-generate test docs
99: ? ??? seed.js # Database seeding
100: ?
101: ??? docs/ # Documentation
102: ? ??? API.md # Complete API reference
103: ? ??? QUICKSTART.md # Quick start guide
104: ? ??? TEST_RESULTS.md # Auto-generated test results
105: ?
106: ??? .github/workflows/ # CI/CD (placeholder)
107: ?
108: ??? Dockerfile # Docker image definition
109: ??? docker-compose.yml # Development environment
110: ??? docker-compose.test.yml # Testing environment
111: ??? .dockerignore # Docker ignore rules
112: ?
113: ??? package.json # Dependencies & scripts
114: ??? .env.example # Environment template
115: ??? .env # Environment variables
116: ??? .gitignore # Git ignore rules
117: ??? .eslintrc.json # ESLint configuration
118: ?
119: ??? postman_collection.json # Postman API collection
120: ??? README.md # Main documentation
121: ??? CONTRIBUTING.md # Contribution guidelines
122: ??? PROJECT_SUMMARY.md # This file
123: ``
124:
125: ---
126:
127: ## Key Features Implemented
128:
129: ### 1. Authentication & Authorization
130: - [x] JWT-based authentication
131: - [x] Password hashing with bcrypt
132: - [x] Role-based access control (Admin, User, Guest)
133: - [x] Account locking after failed attempts
134: - [x] Profile management
135: - [x] Password change functionality
136: - [x] Refresh token support
137:
138: ### 2. Feature Toggle System
139: - [x] Admin-managed feature rules
140: - [x] Role-based feature access
141: - [x] Environment-specific settings (dev/staging/prod)
142: - [x] Percentage-based rollout
143: - [x] Feature dependencies
144: - [x] Real-time feature checking
145: - [x] Bulk operations support
146:
147: ### 3. Rate Guard (API Rate Limiting)
148: - [x] Dynamic rate limiting rules
149: - [x] Role-based limits
150: - [x] Route-specific configuration
151: - [x] IP-based or user-based tracking
152: - [x] Whitelist support
153: - [x] Real-time rule updates
154: - [x] Custom error messages
155:
156: ### 4. Audit Logging
157: - [x] Complete action tracking
158: - [x] User activity logs

```

```
159: - [x] Resource change history
160: - [x] Failed action monitoring
161: - [x] Security event tracking
162: - [x] Export capabilities (JSON/CSV)
163: - [x] Automatic log rotation (90 days TTL)
164:
165: ### 5. Testing Infrastructure
166: - [x] Unit tests (models, utilities)
167: - [x] Integration tests (API endpoints)
168: - [x] System tests (complete workflows)
169: - [x] Test helpers and fixtures
170: - [x] In-memory MongoDB for tests
171: - [x] Automated test documentation
172: - [x] Code coverage reporting
173:
174: ### 6. DevOps & Deployment
175: - [x] Docker containerization
176: - [x] Docker Compose orchestration
177: - [x] Separate test environment
178: - [x] Health check endpoints
179: - [x] Graceful shutdown
180: - [x] Environment-based configuration
181: - [x] Production-ready logging
182:
183: ---
184:
185: ## Testing Coverage
186:
187: ### Test Suites
188: 1. **Unit Tests**
189:     - User model validation
190:     - Password hashing
191:     - Account locking
192:     - Role checking
193:
194: 2. **Integration Tests**
195:     - Authentication flows
196:     - Protected endpoints
197:     - Token validation
198:     - Error handling
199:
200: 3. **System Tests**
201:     - Feature toggle lifecycle
202:     - Rate guard enforcement
203:     - Multi-user workflows
204:     - Complete E2E scenarios
205:
206: ### Test Commands
207: ```bash
208: npm test           # All tests
209: npm run test:unit  # Unit tests
210: npm run test:integration # Integration tests
211: npm run test:system # System tests
212: npm run test:docs  # Generate documentation
213: ```
214:
215: ---
216:
217: ## Deployment Options
218:
219: ### Option 1: Docker (Recommended)
220: ```bash
221: docker-compose up -d
222: ```
223: Includes: App, MongoDB, Redis
224:
225: ### Option 2: Local Development
226: ```bash
227: npm install
228: npm run seed
229: npm run dev
230: ```
231:
```

```
232: ### Option 3: Production
233: ```bash
234: npm install --production
235: NODE_ENV=production npm start
236: ```
237:
238: ---
239:
240: ## API Endpoints Summary
241:
242: ### Public Endpoints
243: - `POST /api/auth/register` - User registration
244: - `POST /api/auth/login` - User login
245: - `GET /api/health` - Health check
246: - `GET /api/status` - System status
247:
248: ### Protected Endpoints (All Users)
249: - `GET /api/auth/me` - Current user profile
250: - `PUT /api/auth/profile` - Update profile
251: - `POST /api/auth/logout` - Logout
252: - `GET /api/features` - List features
253: - `POST /api/features/check` - Check feature access
254:
255: ### Admin-Only Endpoints
256: - `POST /api/features` - Create feature toggle
257: - `PUT /api/features/:id` - Update feature
258: - `DELETE /api/features/:id` - Delete feature
259: - `POST /api/rate-guards` - Create rate limit
260: - `GET /api/users` - List users
261: - `GET /api/audit` - View audit logs
262:
263: ---
264:
265: ## Technical Highlights
266:
267: ### Backend Patterns
268: - MVC Architecture with service layer
269: - Middleware-based request processing
270: - Policy-driven authorization
271: - Repository pattern for data access
272: - Dependency injection for testability
273:
274: ### Security Features
275: - JWT authentication
276: - Bcrypt password hashing
277: - Rate limiting per role
278: - Input validation and sanitization
279: - Helmet.js security headers
280: - CORS configuration
281: - Account lockout protection
282:
283: ### Code Quality
284: - ESLint configuration
285: - Consistent code style
286: - Comprehensive error handling
287: - Detailed logging (Winston)
288: - Type validation (Joi)
289: - Test coverage >70%
290:
291: ### Database Design
292: - Users: Authentication & roles
293: - FeatureToggles: Dynamic feature flags
294: - RateGuards: API rate limiting rules
295: - AuditLogs: Complete audit trail
296:
297: ---
298:
299: ## Documentation
300:
301: ### Available Documentation
302: 1. README.md - Main project documentation
303: 2. docs/API.md - Complete API reference
304: 3. docs/QUICKSTART.md - 5-minute setup guide
```



```
305: 4. **docs/TEST_RESULTS.md** - Auto-generated test results
306: 5. **CONTRIBUTING.md** - Contribution guidelines
307: 6. **postman_collection.json** - Postman API tests
308:
309: ---
310:
311: ## Use Cases Demonstrated
312:
313: 1. **Feature Rollout Strategy**
314:   - Enable features for specific roles
315:   - Gradual rollout with percentage
316:   - Environment-specific features
317:   - A/B testing capability
318:
319: 2. **API Protection**
320:   - Prevent abuse with rate limiting
321:   - Different limits per user role
322:   - Whitelist VIP users
323:   - Custom error messaging
324:
325: 3. **Compliance & Auditing**
326:   - Track all system changes
327:   - User action history
328:   - Security event monitoring
329:   - Data retention policies
330:
331: 4. **User Management**
332:   - Role-based permissions
333:   - Account lifecycle management
334:   - Self-service profile updates
335:   - Administrative controls
336:
337: ---
338:
339: ## Future Enhancements (Optional)
340:
341: ### Potential Additions
342: - [ ] Redis integration for distributed rate limiting
343: - [ ] WebSocket support for real-time updates
344: - [ ] GraphQL API layer
345: - [ ] Advanced analytics dashboard
346: - [ ] Multi-tenancy support
347: - [ ] SSO integration (OAuth, SAML)
348: - [ ] API versioning
349: - [ ] Webhook notifications
350: - [ ] Feature flag analytics
351: - [ ] A/B testing results tracking
352:
353: ---
354:
355: ## Project Metrics
356:
357: - **Development Time**: Optimized for learning and demonstration
358: - **Code Quality**: Production-grade with best practices
359: - **Test Coverage**: 70%+ across all layers
360: - **Documentation**: Comprehensive and auto-generated
361: - **Dependencies**: Minimal and well-maintained
362: - **Docker Support**: Full containerization
363: - **Scalability**: Designed for horizontal scaling
364:
365: ---
366:
367: ## Project Completeness Checklist
368:
369: ### Core Functionality
370: - User authentication with JWT
371: - Role-based authorization
372: - Feature toggle management
373: - Dynamic rate limiting
374: - Comprehensive audit logging
375: - User management (admin)
376:
377: ### Testing
```

```
378: - Unit tests
379: - Integration tests
380: - System tests
381: - Test documentation generator
382: - Code coverage reporting
383:
384: ### DevOps
385: - Docker containerization
386: - Docker Compose setup
387: - Environment configuration
388: - Logging infrastructure
389: - Health check endpoints
390: - Graceful shutdown
391:
392: ### Documentation
393: - README with full setup
394: - API documentation
395: - Quick start guide
396: - Contributing guidelines
397: - Postman collection
398: - Inline code comments
399:
400: ### Code Quality
401: - ESLint configuration
402: - Consistent code style
403: - Error handling
404: - Input validation
405: - Security best practices
406:
407: ---
408:
409: ## Learning Outcomes
410:
411: This project successfully demonstrates:
412:
413: 1. Clean Architecture: Separation of concerns across layers
414: 2. Middleware Patterns: Request processing pipeline
415: 3. Policy-Driven Design: Runtime rule evaluation
416: 4. RBAC: Role-based access control implementation
417: 5. Testing Pyramid: Unit, integration, and system tests
418: 6. DevOps Practices: Containerization and automation
419: 7. API Design: RESTful principles and best practices
420: 8. Database Modeling: Schema design with Mongoose
421: 9. Security: Authentication, authorization, and protection
422: 10. Documentation: Self-documenting and maintainable code
```

■ File: README.md

```
1: # Policy-Driven Feature Toggle & Rate Guard Service
2:
3: A backend-focused full-stack application that demonstrates modern systems for dynamically controlling *
4:
5: ## Project Overview
6:
7: This system allows administrators to define rules that decide:
8: - Which features are accessible to which users
9: - How frequently certain APIs can be accessed
10: - Who is authorized to change system behavior
11:
12: All rules are enforced at runtime using middleware, fully tested using Jest and Supertest, and execu
13:
14: Goal: Demonstrate backend correctness, testability, and clarity?not UI complexity.
15:
16: ---
17:
18: ## Key Features
19:
20: ### 1. Authentication & Role-Based Authorization
21: - JWT-based authentication
22: - Three user roles: Admin, User, Guest
23: - Role validation through middleware
```

```

24: - Account locking after failed login attempts
25: - Password change and profile management
26:
27: ### 2. **Dynamic Feature Toggle Management**
28: - Admin-defined feature access rules
29: - Role-based feature availability
30: - Environment-specific settings (dev/staging/prod)
31: - Percentage-based rollout (gradual feature deployment)
32: - Feature dependencies support
33:
34: ### 3. **Dynamic API Rate Limiting (Rate Guard)**
35: - Route-specific rate limiting rules
36: - Role-based limit differentiation
37: - IP-based or user-based tracking
38: - Whitelist support for exempt users/IPs
39: - Real-time rule updates without restart
40:
41: ### 4. **Comprehensive Audit Logging**
42: - Every system change is recorded
43: - Includes: who, what, when
44: - Failed action tracking
45: - Security event monitoring
46: - Export capabilities (JSON/CSV)
47:
48: ---
49:
50: ## Architecture
51:
52: ```
53: policy-toggle-service/
54: ??? src/
55: ?   ??? config/           # Configuration (env, database)
56: ?   ??? middleware/       # Auth, authorization, feature toggle, rate guard
57: ?   ??? models/           # Mongoose models (User, FeatureToggle, RateGuard, AuditLog)
58: ?   ??? routes/           # Express routes
59: ?   ??? services/         # Business logic layer
60: ?   ??? utils/            # Logger and helpers
61: ?   ??? app.js            # Express app setup
62: ?   ??? index.js          # Server entry point
63: ??? tests/
64: ?   ??? unit/             # Unit tests
65: ?   ??? integration/      # Integration tests
66: ?   ??? system/           # End-to-end system tests
67: ??? scripts/              # Seeding and utility scripts
68: ??? docs/                 # Auto-generated documentation
69: ??? docker-compose.yml    # Container orchestration
70: ```
71:
72: ---
73:
74: ## Getting Started
75:
76: ### Prerequisites
77: - Node.js 18+ and npm
78: - MongoDB 7.0+
79: - Docker & Docker Compose (optional)
80:
81: ### Local Development Setup
82:
83: 1. **Clone the repository**
84: ```bash
85: git clone <repository-url>
86: cd policy-toggle-service
87: ```
88:
89: 2. **Install dependencies**
90: ```bash
91: npm install
92: ```
93:
94: 3. **Configure environment**
95: ```bash
96: cp .env.example .env

```

```
97:   # Edit .env with your configuration
98:   ```
99:
100: 4. **Start MongoDB** (if not using Docker)
101:   ```bash
102:   # Using local MongoDB
103:   mongod --dbpath /path/to/data
104:   ```
105:
106: 5. **Run the application**
107:   ```bash
108:   # Development mode with auto-reload
109:   npm run dev
110:
111:   # Production mode
112:   npm start
113:   ```
114:
115: 6. **Seed sample data** (optional)
116:   ```bash
117:   npm run seed
118:   ```
119:
120: ### Docker Setup
121:
122: 1. **Start all services with Docker Compose**
123:   ```bash
124:   # Build and start
125:   npm run docker:build
126:   npm run docker:up
127:
128:   # Stop services
129:   npm run docker:down
130:   ```
131:
132: 2. **Access the application**
133:   - API: http://localhost:3000/api
134:   - Health: http://localhost:3000/api/health
135:
136: ---
137:
138: ## Testing
139:
140: ### Run All Tests
141: ```bash
142: npm test
143: ```
144:
145: ### Run Specific Test Suites
146: ```bash
147: # Unit tests only
148: npm run test:unit
149:
150: # Integration tests only
151: npm run test:integration
152:
153: # System tests only
154: npm run test:system
155: ```
156:
157: ### Test with Docker
158: ```bash
159: npm run docker:test
160: ```
161:
162: ### Generate Test Documentation
163: ```bash
164: npm run test:docs
165: ```
166:
167: This creates `docs/TEST_RESULTS.md` with:
168: - Total tests executed
169: - Pass/fail summary
```

```
170: - Test grouping (Unit/Integration/System)
171: - Coverage metrics
172: - Execution timestamp
173:
174: ### Code Coverage
175: Coverage reports are generated in the `coverage/` directory after running tests.
176:
177: ---
178:
179: ## ? API Documentation
180:
181: ### Base URL
182: ```
183: http://localhost:3000/api
184: ```
185:
186: ### Authentication Endpoints
187:
188: #### Register
189: ```http
190: POST /api/auth/register
191: Content-Type: application/json
192:
193: {
194:   "email": "user@example.com",
195:   "password": "password123",
196:   "firstName": "John",
197:   "lastName": "Doe"
198: }
199: ```
200:
201: #### Login
202: ```http
203: POST /api/auth/login
204: Content-Type: application/json
205:
206: {
207:   "email": "user@example.com",
208:   "password": "password123"
209: }
210: ```
211:
212: Response:
213: ```json
214: {
215:   "success": true,
216:   "data": {
217:     "user": { ... },
218:     "tokens": {
219:       "accessToken": "jwt-token",
220:       "refreshToken": "refresh-token"
221:     }
222:   }
223: }
224: ```
225:
226: #### Get Current User
227: ```http
228: GET /api/auth/me
229: Authorization: Bearer <token>
230: ```
231:
232: ### Feature Toggle Endpoints
233:
234: #### Create Feature (Admin only)
235: ```http
236: POST /api/features
237: Authorization: Bearer <admin-token>
238: Content-Type: application/json
239:
240: {
241:   "featureName": "premium-features",
242:   "description": "Premium features for paid users",
```

```
243:   "enabled": true,
244:   "allowedRoles": ["admin", "user"],
245:   "rolloutPercentage": 100
246: }
247: ```
248:
249: ##### Get All Features
250: ```http
251: GET /api/features
252: Authorization: Bearer <token>
253: ```
254:
255: ##### Check Feature Access
256: ```http
257: POST /api/features/check
258: Authorization: Bearer <token>
259: Content-Type: application/json
260:
261: {
262:   "featureName": "premium-features"
263: }
264: ```
265:
266: ### Rate Guard Endpoints
267:
268: ##### Create Rate Limit Rule (Admin only)
269: ```http
270: POST /api/rate-guards
271: Authorization: Bearer <admin-token>
272: Content-Type: application/json
273:
274: {
275:   "routePath": "/api/upload",
276:   "method": "POST",
277:   "enabled": true,
278:   "limits": {
279:     "admin": { "maxRequests": 100, "windowMs": 60000 },
280:     "user": { "maxRequests": 20, "windowMs": 60000 },
281:     "guest": { "maxRequests": 3, "windowMs": 60000 }
282:   }
283: }
284: ```
285:
286: ##### Get All Rate Guard Rules
287: ```http
288: GET /api/rate-guards
289: Authorization: Bearer <token>
290: ```
291:
292: ### User Management Endpoints (Admin only)
293:
294: ##### Get All Users
295: ```http
296: GET /api/users?page=1&limit=10
297: Authorization: Bearer <admin-token>
298: ```
299:
300: ##### Update User Role
301: ```http
302: PUT /api/users/:id/role
303: Authorization: Bearer <admin-token>
304: Content-Type: application/json
305:
306: {
307:   "role": "admin"
308: }
309: ```
310:
311: ### Audit Log Endpoints (Admin only)
312:
313: ##### Get Audit Logs
314: ```http
315: GET /api/audit?page=1&limit=50
```

```

316: Authorization: Bearer <admin-token>
317: ```
318:
319: ##### Get Failed Actions
320: ```http
321: GET /api/audit/failed?hours=24
322: Authorization: Bearer <admin-token>
323: ```
324:
325: ### System Endpoints
326:
327: ##### Health Check
328: ```http
329: GET /api/health
330: ```
331:
332: ##### System Status
333: ```http
334: GET /api/status
335: ```
336:
337: ---
338:
339: ## Default Credentials
340:
341: After running `npm run seed`, use these credentials:
342:
343: | Role | Email | Password |
344: |-----|-----|-----|
345: | Admin | admin@example.com | Admin@123456 |
346: | User | user@example.com | User@123456 |
347: | Guest | guest@example.com | Guest@123456 |
348:
349: ** Change these in production!**
350:
351: ---
352:
353: ## Configuration
354:
355: Key environment variables (`.env`):
356:
357: ```env
358: # Server
359: NODE_ENV=development
360: PORT=3000
361:
362: # Database
363: MONGODB_URI=mongodb://localhost:27017/policy-toggle-service
364:
365: # JWT
366: JWT_SECRET=your-secret-key
367: JWT_EXPIRES_IN=24h
368:
369: # Admin Credentials
370: ADMIN_EMAIL=admin@example.com
371: ADMIN_PASSWORD=Admin@123456
372: ```
373:
374: ---
375:
376: ## Testing Strategy
377:
378: ### Test Types
379:
380: 1. **Unit Tests** (`tests/unit/`)
381: - Model validation
382: - Business logic
383: - Utility functions
384: - Edge cases
385:
386: 2. **Integration Tests** (`tests/integration/`)
387: - API endpoints
388: - Middleware chains

```

```

389: - Database operations
390: - Authentication flows
391:
392: 3. **System Tests** (`tests/system/`)
393: - Complete workflows
394: - Multi-step processes
395: - Role-based access
396: - Feature toggle lifecycle
397:
398: ### Coverage Goals
399:
400: - Branches: 70%+
401: - Functions: 70%+
402: - Lines: 70%+
403: - Statements: 70%+
404:
405: ---
406:
407: ## Docker Deployment
408:
409: ### Production Deployment
410: ```bash
411: docker-compose up -d
412: ```
413:
414: Services:
415: - **App**: Node.js application (port 3000)
416: - **MongoDB**: Database (port 27017)
417: - **Redis**: Caching/rate limiting (port 6379)
418:
419: ### Health Checks
420: All services include health checks for monitoring.
421:
422: ---
423:
424: ## Project Structure
425:
426: ```
427: src/
428:   ??? config/
429:   ?   ??? index.js           # Configuration loader
430:   ?   ??? database.js        # Database connection
431:   ??? middleware/
432:   ?   ??? auth.js            # JWT authentication
433:   ?   ??? authorization.js    # Role-based access
434:   ?   ??? featureToggle.js    # Feature enforcement
435:   ?   ??? rateGuard.js        # Dynamic rate limiting
436:   ?   ??? validation.js       # Request validation
437:   ?   ??? errorHandler.js     # Error handling
438:   ?   ??? requestLogger.js    # Request logging
439:   ??? models/
440:   ?   ??? User.js             # User model with auth
441:   ?   ??? FeatureToggle.js    # Feature toggle model
442:   ?   ??? RateGuard.js        # Rate guard model
443:   ?   ??? AuditLog.js         # Audit logging model
444:   ??? routes/
445:   ?   ??? auth.routes.js      # Authentication routes
446:   ?   ??? users.routes.js     # User management
447:   ?   ??? features.routes.js  # Feature toggles
448:   ?   ??? rateGuards.routes.js # Rate guards
449:   ?   ??? audit.routes.js     # Audit logs
450:   ??? services/
451:   ?   ??? authService.js      # Auth business logic
452:   ?   ??? userService.js      # User management logic
453:   ?   ??? featureToggleService.js # Feature logic
454:   ?   ??? rateGuardService.js # Rate limiting logic
455:   ??? utils/
456:   ?   ??? logger.js           # Winston logger
457: ```
458:
459: ---
460:
461: ## Learning Outcomes

```



```
462:
463: This project demonstrates:
464:
465: **Clean Architecture** - Separation of concerns
466: **Middleware Patterns** - Request processing pipeline
467: **Policy-Driven Design** - Dynamic rule evaluation
468: **Role-Based Access Control** - Authorization patterns
469: **Comprehensive Testing** - Unit, integration, system tests
470: **Automated Documentation** - Self-documenting test results
471: **Docker Containerization** - Consistent environments
472: **Database Design** - Schema modeling with Mongoose
473: **API Design** - RESTful endpoint structure
474: **Security Best Practices** - JWT, password hashing, rate limiting
```

■ File: docker-compose.test.yml

```
=====
1: version: "3.8"
2:
3: services:
4:   # MongoDB for testing
5:   mongodb-test:
6:     image: mongo:7.0
7:     container_name: policy-toggle-mongodb-test
8:     environment:
9:       MONGO_INITDB_DATABASE: policy-toggle-service-test
10:    ports:
11:      - "27018:27017"
12:    networks:
13:      - policy-toggle-test-network
14:    tmpfs:
15:      - /data/db
16:
17:   # Test runner
18:   test:
19:     build:
20:       context: .
21:       dockerfile: Dockerfile
22:       target: build
23:     container_name: policy-toggle-test
24:     environment:
25:       NODE_ENV: test
26:       MONGODB_TEST_URI: mongodb://mongodb-test:27017/policy-toggle-service-test
27:       JWT_SECRET: test-secret-key
28:     depends_on:
29:       - mongodb-test
30:     networks:
31:       - policy-toggle-test-network
32:     volumes:
33:       - ./src:/app/src
34:       - ./tests:/app/tests
35:       - ./coverage:/app/coverage
36:     command: npm test
37:
38:   networks:
39:     policy-toggle-test-network:
40:       driver: bridge
=====
```

■ File: docker-compose.yml

```
=====
1: version: "3.8"
2:
3: services:
4:   # MongoDB Database
5:   mongodb:
6:     image: mongo:7.0
7:     container_name: policy-toggle-mongodb
8:     restart: unless-stopped
9:     environment:
10:       MONGO_INITDB_DATABASE: policy-toggle-service
```

```

11:     ports:
12:       - "27017:27017"
13:     volumes:
14:       - mongodb_data:/data/db
15:       - mongodb_config:/data/configdb
16:     networks:
17:       - policy-toggle-network
18:     healthcheck:
19:       test: echo 'db.runCommand("ping").ok' | mongosh localhost:27017/test --quiet
20:       interval: 10s
21:       timeout: 5s
22:       retries: 5
23:
24: # Redis (optional - for advanced rate limiting)
25: redis:
26:   image: redis:7-alpine
27:   container_name: policy-toggle-redis
28:   restart: unless-stopped
29:   ports:
30:     - "6379:6379"
31:   volumes:
32:     - redis_data:/data
33:   networks:
34:     - policy-toggle-network
35:   healthcheck:
36:     test: ["CMD", "redis-cli", "ping"]
37:     interval: 10s
38:     timeout: 3s
39:     retries: 5
40:
41: # Application Service
42: app:
43:   build:
44:     context: .
45:     dockerfile: Dockerfile
46:     target: production
47:   container_name: policy-toggle-app
48:   restart: unless-stopped
49:   ports:
50:     - "3000:3000"
51:   environment:
52:     NODE_ENV: production
53:     PORT: 3000
54:     MONGODB_URI: mongodb://mongodb:27017/policy-toggle-service
55:     REDIS_HOST: redis
56:     REDIS_PORT: 6379
57:     JWT_SECRET: ${JWT_SECRET:-change-this-secret-in-production}
58:     ADMIN_EMAIL: ${ADMIN_EMAIL:-admin@example.com}
59:     ADMIN_PASSWORD: ${ADMIN_PASSWORD:-Admin@123456}
60:   depends_on:
61:     mongodb:
62:       condition: service_healthy
63:     redis:
64:       condition: service_healthy
65:   networks:
66:     - policy-toggle-network
67:   volumes:
68:     - ./logs:/app/logs
69:   healthcheck:
70:     test:
71:       [
72:         "CMD",
73:         "node",
74:         "-e",
75:         "require('http').get('http://localhost:3000/api/health', (r) => {process.exit(r.statusCode === 20
76:       ]
77:     interval: 30s
78:     timeout: 10s
79:     retries: 3
80:     start_period: 40s
81:
82: networks:
83:   policy-toggle-network:

```

```
84:     driver: bridge
85:
86: volumes:
87:   mongodb_data:
88:     driver: local
89:   mongodb_config:
90:     driver: local
91:   redis_data:
92:     driver: local
```

■ File: docs\API.md

```
1: # API Documentation
2:
3: ## Base URL
4: ```
5: http://localhost:3000/api
6: ```
7:
8: ## Authentication
9:
10: All protected endpoints require a JWT token in the Authorization header:
11: ```
12: Authorization: Bearer <your-jwt-token>
13: ```
14:
15: ---
16:
17: ## Authentication Endpoints
18:
19: ### Register New User
20: Creates a new user account.
21:
22: **Endpoint:** `POST /api/auth/register`
23: **Access:** Public
24:
25: **Request Body:**
26: ```json
27: {
28:   "email": "user@example.com",
29:   "password": "password123",
30:   "firstName": "John",
31:   "lastName": "Doe"
32: }
33: ```
34:
35: **Response:** `201 Created`
36: ```json
37: {
38:   "success": true,
39:   "message": "User registered successfully",
40:   "data": {
41:     "user": {
42:       "_id": "user-id",
43:       "email": "user@example.com",
44:       "role": "user",
45:       "firstName": "John",
46:       "lastName": "Doe"
47:     },
48:     "tokens": {
49:       "accessToken": "jwt-access-token",
50:       "refreshToken": "jwt-refresh-token"
51:     }
52:   }
53: }
54: ```
55:
56: ### Login
57: Authenticates a user and returns JWT tokens.
58:
59: **Endpoint:** `POST /api/auth/login`
```

```
60: **Access:** Public
61:
62: **Request Body:**
63: ```json
64: {
65:   "email": "user@example.com",
66:   "password": "password123"
67: }
68: ```
69:
70: **Response:** `200 OK`
71: ```json
72: {
73:   "success": true,
74:   "message": "Login successful",
75:   "data": {
76:     "user": { ... },
77:     "tokens": {
78:       "accessToken": "jwt-access-token",
79:       "refreshToken": "jwt-refresh-token"
80:     }
81:   }
82: }
83: ```
84:
85: **Error Responses:**
86: - `401` - Invalid credentials
87: - `403` - Account deactivated
88: - `423` - Account locked
89:
90: ### Get Current User
91: Retrieves the authenticated user's profile.
92:
93: **Endpoint:** `GET /api/auth/me`
94: **Access:** Private
95:
96: **Response:** `200 OK`
97: ```json
98: {
99:   "success": true,
100:   "data": {
101:     "user": {
102:       "_id": "user-id",
103:       "email": "user@example.com",
104:       "role": "user",
105:       "firstName": "John",
106:       "lastName": "Doe",
107:       "isActive": true,
108:       "createdAt": "2025-01-01T00:00:00.000Z"
109:     }
110:   }
111: }
112: ```
113:
114: ### Update Profile
115: Updates the authenticated user's profile.
116:
117: **Endpoint:** `PUT /api/auth/profile`
118: **Access:** Private
119:
120: **Request Body:**
121: ```json
122: {
123:   "firstName": "Jane",
124:   "lastName": "Smith"
125: }
126: ```
127:
128: **Response:** `200 OK`
129:
130: ### Change Password
131: Changes the authenticated user's password.
132:
```

```
133: **Endpoint:** `PUT /api/auth/change-password`
134: **Access:** Private
135:
136: **Request Body:**
137: ```json
138: {
139:   "currentPassword": "oldPassword123",
140:   "newPassword": "newPassword456"
141: }
142: ```
143:
144: **Response:** `200 OK`
145:
146: ### Logout
147: Logs out the current user.
148:
149: **Endpoint:** `POST /api/auth/logout`
150: **Access:** Private
151:
152: **Response:** `200 OK`
153:
154: ---
155:
156: ## Feature Toggle Endpoints
157:
158: ### Get All Features
159: Retrieves all feature toggles.
160:
161: **Endpoint:** `GET /api/features`
162: **Access:** Private
163:
164: **Query Parameters:**
165: - `enabled` (boolean) - Filter by enabled status
166: - `search` (string) - Search by name or description
167:
168: **Response:** `200 OK`
169: ```json
170: {
171:   "success": true,
172:   "data": {
173:     "features": [
174:       {
175:         "_id": "feature-id",
176:         "featureName": "premium-features",
177:         "description": "Premium features",
178:         "enabled": true,
179:         "allowedRoles": ["admin", "user"],
180:         "rolloutPercentage": 100,
181:         "environments": {
182:           "development": { "enabled": true },
183:           "production": { "enabled": false }
184:         }
185:       }
186:     ],
187:     "count": 1
188:   }
189: }
190: ```
191:
192: ### Get Enabled Features
193: Retrieves features enabled for the current user's role.
194:
195: **Endpoint:** `GET /api/features/enabled`
196: **Access:** Private
197:
198: **Response:** `200 OK`
199:
200: ### Check Feature Access
201: Checks if a specific feature is enabled for the current user.
202:
203: **Endpoint:** `POST /api/features/check`
204: **Access:** Private
205:
```

```
206: **Request Body:**
207: ```json
208: {
209:   "featureName": "premium-features"
210: }
211: ```
212:
213: **Response:** `200 OK`
214: ```json
215: {
216:   "success": true,
217:   "data": {
218:     "featureName": "premium-features",
219:     "enabled": true,
220:     "role": "user",
221:     "environment": "development"
222:   }
223: }
224: ```
225:
226: ### Create Feature (Admin)
227: Creates a new feature toggle.
228:
229: **Endpoint:** `POST /api/features`
230: **Access:** Admin Only
231:
232: **Request Body:**
233: ```json
234: {
235:   "featureName": "new-feature",
236:   "description": "Description of the feature",
237:   "enabled": true,
238:   "allowedRoles": ["admin", "user"],
239:   "rolloutPercentage": 50,
240:   "environments": {
241:     "development": { "enabled": true },
242:     "production": { "enabled": false }
243:   }
244: }
245: ```
246:
247: **Response:** `201 Created`
248:
249: ### Update Feature (Admin)
250: Updates an existing feature toggle.
251:
252: **Endpoint:** `PUT /api/features/:id`
253: **Access:** Admin Only
254:
255: **Request Body:**
256: ```json
257: {
258:   "description": "Updated description",
259:   "rolloutPercentage": 75
260: }
261: ```
262:
263: **Response:** `200 OK`
264:
265: ### Toggle Feature (Admin)
266: Enables or disables a feature.
267:
268: **Endpoint:** `PUT /api/features/:id/toggle`
269: **Access:** Admin Only
270:
271: **Request Body:**
272: ```json
273: {
274:   "enabled": false
275: }
276: ```
277:
278: **Response:** `200 OK`
```

```
279:
280: ### Delete Feature (Admin)
281: Deletes a feature toggle.
282:
283: **Endpoint:** `DELETE /api/features/:id`
284: **Access:** Admin Only
285:
286: **Response:** `200 OK`
287:
288: ### Get Feature Statistics (Admin)
289: Retrieves feature toggle statistics.
290:
291: **Endpoint:** `GET /api/features/stats`
292: **Access:** Admin Only
293:
294: **Response:** `200 OK`
295: ``json
296: {
297:   "success": true,
298:   "data": {
299:     "total": 10,
300:     "enabled": 7,
301:     "disabled": 3,
302:     "byEnvironment": {
303:       "development": 10,
304:       "production": 5
305:     }
306:   }
307: }
308: ``
309:
310: ---
311:
312: ## Rate Guard Endpoints
313:
314: ### Get All Rate Guards
315: Retrieves all rate guard rules.
316:
317: **Endpoint:** `GET /api/rate-guards`
318: **Access:** Private
319:
320: **Query Parameters:**
321: - `enabled` (boolean) - Filter by enabled status
322: - `method` (string) - Filter by HTTP method
323: - `search` (string) - Search by route path
324:
325: **Response:** `200 OK`
326:
327: ### Create Rate Guard (Admin)
328: Creates a new rate limiting rule.
329:
330: **Endpoint:** `POST /api/rate-guards`
331: **Access:** Admin Only
332:
333: **Request Body:**
334: ``json
335: {
336:   "routePath": "/api/upload",
337:   "method": "POST",
338:   "description": "Upload rate limiting",
339:   "enabled": true,
340:   "limits": {
341:     "admin": {
342:       "maxRequests": 100,
343:       "windowMs": 60000
344:     },
345:     "user": {
346:       "maxRequests": 20,
347:       "windowMs": 60000
348:     },
349:     "guest": {
350:       "maxRequests": 3,
351:       "windowMs": 60000
```

```
352:     }
353:   },
354:   "errorMessage": "Rate limit exceeded. Please try again later."
355: }
356: ```
357:
358: **Response:** `201 Created`
359:
360: ### Test Rate Limit
361: Tests the rate limit for a specific route.
362:
363: **Endpoint:** `POST /api/rate-guards/test`
364: **Access:** Private
365:
366: **Request Body:**
367: ```json
368: {
369:   "routePath": "/api/upload",
370:   "method": "POST"
371: }
372: ```
373:
374: **Response:** `200 OK`
375: ```json
376: {
377:   "success": true,
378:   "data": {
379:     "hasRule": true,
380:     "enabled": true,
381:     "limit": {
382:       "maxRequests": 20,
383:       "windowMs": 60000
384:     }
385:   }
386: }
387: ```
388:
389: ---
390:
391: ## User Management Endpoints (Admin Only)
392:
393: ### Get All Users
394: Retrieves a paginated list of users.
395:
396: **Endpoint:** `GET /api/users`
397: **Access:** Admin Only
398:
399: **Query Parameters:**
400: - `page` (number) - Page number (default: 1)
401: - `limit` (number) - Items per page (default: 10)
402: - `role` (string) - Filter by role
403: - `isActive` (boolean) - Filter by active status
404: - `search` (string) - Search by email or name
405:
406: **Response:** `200 OK`
407: ```json
408: {
409:   "success": true,
410:   "data": {
411:     "users": [ ... ],
412:     "pagination": {
413:       "page": 1,
414:       "limit": 10,
415:       "total": 50,
416:       "pages": 5
417:     }
418:   }
419: }
420: ```
421:
422: ### Update User
423: Updates a user's information.
424:
```



```
425: **Endpoint:** `PUT /api/users/:id`
426: **Access:** Admin Only
427:
428: **Request Body:**
429: ```json
430: {
431:   "firstName": "Updated",
432:   "role": "admin",
433:   "isActive": true
434: }
435: ```
436:
437: **Response:** `200 OK`
438:
439: ### Change User Role
440: Changes a user's role.
441:
442: **Endpoint:** `PUT /api/users/:id/role`
443: **Access:** Admin Only
444:
445: **Request Body:**
446: ```json
447: {
448:   "role": "admin"
449: }
450: ```
451:
452: **Response:** `200 OK`
453:
454: ### Deactivate User
455: Deactivates a user account.
456:
457: **Endpoint:** `PUT /api/users/:id/deactivate`
458: **Access:** Admin Only
459:
460: **Response:** `200 OK`
461:
462: ### Activate User
463: Activates a user account.
464:
465: **Endpoint:** `PUT /api/users/:id/activate`
466: **Access:** Admin Only
467:
468: **Response:** `200 OK`
469:
470: ### Delete User
471: Permanently deletes a user.
472:
473: **Endpoint:** `DELETE /api/users/:id`
474: **Access:** Admin Only
475:
476: **Response:** `200 OK`
477:
478: ---
479:
480: ## Audit Log Endpoints (Admin Only)
481:
482: ### Get Audit Logs
483: Retrieves audit logs with filtering.
484:
485: **Endpoint:** `GET /api/audit`
486: **Access:** Admin Only
487:
488: **Query Parameters:**
489: - `action` (string) - Filter by action type
490: - `resourceType` (string) - Filter by resource type
491: - `userId` (string) - Filter by user ID
492: - `success` (boolean) - Filter by success status
493: - `startDate` (ISO date) - Start date for range
494: - `endDate` (ISO date) - End date for range
495: - `page` (number) - Page number
496: - `limit` (number) - Items per page
497:
```

```
498: **Response:** `200 OK`
499:
500: ### Get User Audit Logs
501: Retrieves audit logs for a specific user.
502:
503: **Endpoint:** `GET /api/audit/user/:userId`
504: **Access:** Admin Only
505:
506: **Response:** `200 OK`
507:
508: ### Get Failed Actions
509: Retrieves recent failed actions.
510:
511: **Endpoint:** `GET /api/audit/failed`
512: **Access:** Admin Only
513:
514: **Query Parameters:**
515: - `hours` (number) - Look back period in hours (default: 24)
516:
517: **Response:** `200 OK`
518:
519: ### Get Audit Statistics
520: Retrieves audit log statistics.
521:
522: **Endpoint:** `GET /api/audit/stats`
523: **Access:** Admin Only
524:
525: **Query Parameters:**
526: - `startDate` (ISO date) - Start date
527: - `endDate` (ISO date) - End date
528:
529: **Response:** `200 OK`
530:
531: ---
532:
533: ## System Endpoints
534:
535: ### Health Check
536: Basic health check endpoint.
537:
538: **Endpoint:** `GET /api/health`
539: **Access:** Public
540:
541: **Response:** `200 OK`
542: ```json
543: {
544:   "success": true,
545:   "status": "healthy",
546:   "timestamp": "2025-01-30T10:00:00.000Z",
547:   "uptime": 12345
548: }
549: ```
550:
551: ### System Status
552: Detailed system status.
553:
554: **Endpoint:** `GET /api/status`
555: **Access:** Public
556:
557: **Response:** `200 OK`
558: ```json
559: {
560:   "success": true,
561:   "data": {
562:     "service": "Policy Toggle Service",
563:     "version": "1.0.0",
564:     "environment": "development",
565:     "status": "operational",
566:     "database": "connected",
567:     "timestamp": "2025-01-30T10:00:00.000Z"
568:   }
569: }
570: ```
```

```
571:
572: ---
573:
574: ## Error Responses
575:
576: All endpoints may return these error responses:
577:
578: ### 400 Bad Request
579: Invalid request data or validation error.
580: ```json
581: {
582:   "success": false,
583:   "message": "Validation failed",
584:   "errors": [
585:     {
586:       "field": "email",
587:       "message": "Please provide a valid email"
588:     }
589:   ]
590: }
591: ```
592:
593: ### 401 Unauthorized
594: Missing or invalid authentication token.
595: ```json
596: {
597:   "success": false,
598:   "message": "Authentication required"
599: }
600: ```
601:
602: ### 403 Forbidden
603: Insufficient permissions.
604: ```json
605: {
606:   "success": false,
607:   "message": "Insufficient permissions"
608: }
609: ```
610:
611: ### 404 Not Found
612: Resource not found.
613: ```json
614: {
615:   "success": false,
616:   "message": "Resource not found"
617: }
618: ```
619:
620: ### 429 Too Many Requests
621: Rate limit exceeded.
622: ```json
623: {
624:   "success": false,
625:   "message": "Rate limit exceeded. Please try again later.",
626:   "retryAfter": 60
627: }
628: ```
629:
630: ### 500 Internal Server Error
631: Server error.
632: ```json
633: {
634:   "success": false,
635:   "message": "Something went wrong. Please try again later."
636: }
637: ```
638:
639: ---
640:
641: ## Rate Limiting
642:
643: Rate limits are applied based on user role and route configuration. Headers are included in responses:
```

```
644:
645: ```
646: X-RateLimit-Limit: 100
647: X-RateLimit-Remaining: 95
648: X-RateLimit-Reset: 1234567890
649: ```
650:
651: ---
652:
653: ## Pagination
654:
655: Paginated endpoints return data in this format:
656:
657: ```json
658: {
659:   "success": true,
660:   "data": {
661:     "items": [ ... ],
662:     "pagination": {
663:       "page": 1,
664:       "limit": 10,
665:       "total": 100,
666:       "pages": 10
667:     }
668:   }
669: }
670: ```
```

■ File: docs\QUICKSTART.md

```
1: # Quick Start Guide
2:
3: Get the Policy Toggle Service running in 5 minutes!
4:
5: ## Option 1: Docker (Recommended)
6:
7: ### Step 1: Prerequisites
8: - Docker and Docker Compose installed
9:
10: ### Step 2: Start Services
11: ```bash
12: # Clone repository
13: git clone <repo-url>
14: cd policy-toggle-service
15:
16: # Start all services
17: docker compose up -d
18: ```
19:
20: ### Step 3: Verify
21: ```bash
22: # Check health
23: curl http://localhost:3000/api/health
24:
25: # Expected response:
26: # {"success":true,"status":"healthy", ...}
27: ```
28:
29: ### Step 4: Login
30: ```bash
31: # Login as admin
32: curl -X POST http://localhost:3000/api/auth/login \
33:   -H "Content-Type: application/json" \
34:   -d '{"email":"admin@example.com","password":"Admin@123456"}'
35: ```
36:
37: **Done!**
38:
39: ---
40:
41: ## Option 2: Local Development
```

```
42:
43: ### Step 1: Prerequisites
44: - Node.js 18+
45: - MongoDB 7.0+
46:
47: ### Step 2: Install
48: ```bash
49: # Clone repository
50: git clone <repo-url>
51: cd policy-toggle-service
52:
53: # Install dependencies
54: npm install
55:
56: # Copy environment file
57: cp .env.example .env
58: ```
59:
60: ### Step 3: Start MongoDB
61: ```bash
62: # Start MongoDB (in separate terminal)
63: mongod --dbpath /path/to/data
64: ```
65:
66: ### Step 4: Start Application
67: ```bash
68: # Seed sample data
69: npm run seed
70:
71: # Start development server
72: npm run dev
73: ```
74:
75: ### Step 5: Verify
76: ```bash
77: # Check health
78: curl http://localhost:3000/api/health
79: ```
80:
81: **Done!**
82:
83: ---
84:
85: ## Quick Test Commands
86:
87: ### 1. Register a New User
88: ```bash
89: curl -X POST http://localhost:3000/api/auth/register \
90:   -H "Content-Type: application/json" \
91:   -d '{
92:     "email": "test@example.com",
93:     "password": "password123",
94:     "firstName": "Test",
95:     "lastName": "User"
96:   }'
97: ```
98:
99: ### 2. Login
100: ```bash
101: curl -X POST http://localhost:3000/api/auth/login \
102:   -H "Content-Type: application/json" \
103:   -d '{
104:     "email": "test@example.com",
105:     "password": "password123"
106:   }'
107: ```
108:
109: Save the `accessToken` from the response!
110:
111: ### 3. Get Your Profile
112: ```bash
113: TOKEN="your-access-token-here"
114:
```

```
115: curl http://localhost:3000/api/auth/me \
116:   -H "Authorization: Bearer $TOKEN"
117: ```
118:
119: ### 4. Check Feature Access
120: ```bash
121: curl -X POST http://localhost:3000/api/features/check \
122:   -H "Authorization: Bearer $TOKEN" \
123:   -H "Content-Type: application/json" \
124:   -d '{"featureName": "premium-features"}'
125: ```
126:
127: ### 5. Get Available Features
128: ```bash
129: curl http://localhost:3000/api/features/enabled \
130:   -H "Authorization: Bearer $TOKEN"
131: ```
132:
133: ---
134:
135: ## Admin Actions
136:
137: ### Login as Admin
138: ```bash
139: curl -X POST http://localhost:3000/api/auth/login \
140:   -H "Content-Type: application/json" \
141:   -d '{
142:     "email": "admin@example.com",
143:     "password": "Admin@123456"
144:   }'
145: ```
146:
147: ### Create a New Feature Toggle
148: ```bash
149: ADMIN_TOKEN="your-admin-token"
150:
151: curl -X POST http://localhost:3000/api/features \
152:   -H "Authorization: Bearer $ADMIN_TOKEN" \
153:   -H "Content-Type: application/json" \
154:   -d '{
155:     "featureName": "new-feature",
156:     "description": "My new feature",
157:     "enabled": true,
158:     "allowedRoles": ["admin", "user"],
159:     "rolloutPercentage": 100
160:   }'
161: ```
162:
163: ### Create a Rate Limit Rule
164: ```bash
165: curl -X POST http://localhost:3000/api/rate-guards \
166:   -H "Authorization: Bearer $ADMIN_TOKEN" \
167:   -H "Content-Type: application/json" \
168:   -d '{
169:     "routePath": "/api/test",
170:     "method": "POST",
171:     "enabled": true,
172:     "limits": {
173:       "admin": {"maxRequests": 100, "windowMs": 60000},
174:       "user": {"maxRequests": 20, "windowMs": 60000},
175:       "guest": {"maxRequests": 5, "windowMs": 60000}
176:     }
177:   }'
178: ```
179:
180: ### View Audit Logs
181: ```bash
182: curl http://localhost:3000/api/audit \
183:   -H "Authorization: Bearer $ADMIN_TOKEN"
184: ```
185:
186: ---
187:
```

```
188: ## Running Tests
189:
190: ```bash
191: # All tests
192: npm test
193:
194: # Unit tests only
195: npm run test:unit
196:
197: # Integration tests
198: npm run test:integration
199:
200: # System tests
201: npm run test:system
202:
203: # Generate test documentation
204: npm run test:docs
205: ```
206:
207: ---
208:
209: ## Default Credentials
210:
211: | Role | Email | Password |
212: |-----|-----|-----|
213: | Admin | admin@example.com | Admin@123456 |
214: | User | user@example.com | User@123456 |
215: | Guest | guest@example.com | Guest@123456 |
216:
217: ---
218:
219: ## Common Issues
220:
221: ### MongoDB Connection Error
222: ```bash
223: # Make sure MongoDB is running
224: mongod --version
225:
226: # Check MongoDB is accessible
227: mongo --eval "db.version()"
228: ```
229:
230: ### Port Already in Use
231: ```bash
232: # Kill process on port 3000
233: lsof -ti:3000 | xargs kill -9
234: ```
235:
236: ### Permission Denied (Docker)
237: ```bash
238: # Run with sudo or add user to docker group
239: sudo docker-compose up -d
240: ```
241:
242: ---
243:
244: ## Next Steps
245:
246: 1. Read [API Documentation](docs/API.md)
247: 2. Explore the codebase structure
248: 3. Run the test suite
249: 4. Try modifying feature toggles
250: 5. Create custom rate limiting rules
251:
252: ---
253:
254: ## Support
255:
256: - Documentation: See `/docs` folder
257: - Issues: Open a GitHub issue
258: - Tests: Run `npm test` for examples
```

■ File: docs\TEST_RESULTS.md

■ File: package.json

```
1: {
2:   "name": "policy-toggle-service",
3:   "version": "1.0.0",
4:   "description": "Policy-Driven Feature Toggle & Rate Guard Service - A backend-focused system for dynamic
5:   "main": "src/index.js",
6:   "scripts": {
7:     "start": "node src/index.js",
8:     "dev": "nodemon src/index.js",
9:     "test": "NODE_ENV=test jest --coverage --verbose",
10:    "test:unit": "NODE_ENV=test jest tests/unit --coverage",
11:    "test:integration": "NODE_ENV=test jest tests/integration --coverage",
12:    "test:system": "NODE_ENV=test jest tests/system --coverage",
13:    "test:watch": "NODE_ENV=test jest --watch",
14:    "test:docs": "node scripts/generate-test-docs.js",
15:    "docker:build": "docker-compose build",
16:    "docker:up": "docker-compose up -d",
17:    "docker:down": "docker-compose down",
18:    "docker:test": "docker-compose -f docker-compose.test.yml up --abort-on-container-exit",
19:    "lint": "eslint src tests",
20:    "lint:fix": "eslint src tests --fix",
21:    "migrate": "node scripts/migrate.js",
22:    "seed": "node scripts/seed.js"
23:  },
24:  "keywords": [
25:    "feature-toggle",
26:    "rate-limiting",
27:    "policy-engine",
28:    "backend",
29:    "nodejs",
30:    "express",
31:    "jwt",
32:    "docker"
33:  ],
34:  "author": "Your Name",
35:  "license": "MIT",
36:  "dependencies": {
37:    "bcryptjs": "^2.4.3",
38:    "cors": "^2.8.5",
39:    "dotenv": "^16.3.1",
40:    "express": "^4.18.2",
41:    "express-rate-limit": "^7.1.5",
42:    "helmet": "^7.1.0",
43:    "joi": "^17.11.0",
44:    "jsonwebtoken": "^9.0.2",
45:    "mongoose": "^8.0.3",
46:    "morgan": "^1.10.0",
47:    "redis": "^4.6.11",
48:    "winston": "^3.11.0"
49:  },
50:  "devDependencies": {
51:    "@faker-js/faker": "^8.3.1",
52:    "eslint": "^8.55.0",
53:    "jest": "^29.7.0",
54:    "nodemon": "^3.0.2",
55:    "supertest": "^6.3.3"
56:  },
57:  "jest": {
58:    "testEnvironment": "node",
59:    "coverageDirectory": "coverage",
60:    "collectCoverageFrom": [
61:      "src/**/*.js",
62:      "!src/index.js"
63:    ],
64:    "testMatch": [
65:      "**/tests/**/*.test.js"
66:    ],
```



```

67:     "setupFilesAfterEnv": [
68:         "<rootDir>/tests/setup.js"
69:     ],
70:     "coverageThreshold": {
71:         "global": {
72:             "branches": 70,
73:             "functions": 70,
74:             "lines": 70,
75:             "statements": 70
76:         }
77:     },
78: },
79: "engines": {
80:     "node": ">=18.0.0",
81:     "npm": ">=9.0.0"
82: }
83: }

```

■ File: postman_collection.json

```

=====
1: {
2:   "info": {
3:     "name": "Policy Toggle Service API",
4:     "description": "Complete API collection for Policy-Driven Feature Toggle & Rate Guard Service",
5:     "schema": "https://schema.getpostman.com/json/collection/v2.1.0/collection.json"
6:   },
7:   "auth": {
8:     "type": "bearer",
9:     "bearer": [
10:      {
11:        "key": "token",
12:        "value": "{{accessToken}}",
13:        "type": "string"
14:      }
15:    ]
16:  },
17:  "variable": [
18:    {
19:      "key": "baseUrl",
20:      "value": "http://localhost:3000/api",
21:      "type": "string"
22:    },
23:    {
24:      "key": "accessToken",
25:      "value": "",
26:      "type": "string"
27:    },
28:    {
29:      "key": "adminToken",
30:      "value": "",
31:      "type": "string"
32:    }
33:  ],
34:  "item": [
35:    {
36:      "name": "Authentication",
37:      "item": [
38:        {
39:          "name": "Register",
40:          "request": {
41:            "method": "POST",
42:            "header": [{ "key": "Content-Type", "value": "application/json" }],
43:            "url": "{{baseUrl}}/auth/register",
44:            "body": {
45:              "mode": "raw",
46:              "raw": "{\n  \"email\": \"test@example.com\",\n  \"password\": \"password123\",\n  \"firstName":
47:            }
48:          },
49:        },
50:        {
51:          "name": "Login",

```

```

52:     "event": [
53:         {
54:             "listen": "test",
55:             "script": {
56:                 "exec": [
57:                     "if (pm.response.code === 200) {",
58:                     "     const response = pm.response.json();",
59:                     "     pm.environment.set('accessToken', response.data.tokens.accessToken);",
60:                     "}"
61:                 ]
62:             }
63:         }
64:     ],
65:     "request": {
66:         "method": "POST",
67:         "header": [{ "key": "Content-Type", "value": "application/json" }],
68:         "url": "{{baseUrl}}/auth/login",
69:         "body": {
70:             "mode": "raw",
71:             "raw": "{\n  \"email\": \"user@example.com\", \n  \"password\": \"User@123456\"\n}"
72:         }
73:     }
74: },
75: {
76:     "name": "Login as Admin",
77:     "event": [
78:         {
79:             "listen": "test",
80:             "script": {
81:                 "exec": [
82:                     "if (pm.response.code === 200) {",
83:                     "     const response = pm.response.json();",
84:                     "     pm.environment.set('adminToken', response.data.tokens.accessToken);",
85:                     "     pm.environment.set('accessToken', response.data.tokens.accessToken);",
86:                     "}"
87:                 ]
88:             }
89:         }
90:     ],
91:     "request": {
92:         "method": "POST",
93:         "header": [{ "key": "Content-Type", "value": "application/json" }],
94:         "url": "{{baseUrl}}/auth/login",
95:         "body": {
96:             "mode": "raw",
97:             "raw": "{\n  \"email\": \"admin@example.com\", \n  \"password\": \"Admin@123456\"\n}"
98:         }
99:     }
100: },
101: {
102:     "name": "Get Current User",
103:     "request": {
104:         "method": "GET",
105:         "url": "{{baseUrl}}/auth/me"
106:     }
107: },
108: {
109:     "name": "Update Profile",
110:     "request": {
111:         "method": "PUT",
112:         "header": [{ "key": "Content-Type", "value": "application/json" }],
113:         "url": "{{baseUrl}}/auth/profile",
114:         "body": {
115:             "mode": "raw",
116:             "raw": "{\n  \"firstName\": \"Updated\", \n  \"lastName\": \"Name\"\n}"
117:         }
118:     }
119: },
120: {
121:     "name": "Logout",
122:     "request": {
123:         "method": "POST",
124:         "url": "{{baseUrl}}/auth/logout"

```

```

125:         }
126:     }
127: ]
128: },
129: {
130:     "name": "Features",
131:     "item": [
132:         {
133:             "name": "Get All Features",
134:             "request": {
135:                 "method": "GET",
136:                 "url": "{{baseUrl}}/features"
137:             }
138:         },
139:         {
140:             "name": "Get Enabled Features",
141:             "request": {
142:                 "method": "GET",
143:                 "url": "{{baseUrl}}/features/enabled"
144:             }
145:         },
146:         {
147:             "name": "Check Feature Access",
148:             "request": {
149:                 "method": "POST",
150:                 "header": [{ "key": "Content-Type", "value": "application/json" }],
151:                 "url": "{{baseUrl}}/features/check",
152:                 "body": {
153:                     "mode": "raw",
154:                     "raw": "{\n  \"featureName\": \"premium-features\"\n}"
155:                 }
156:             }
157:         },
158:         {
159:             "name": "Create Feature (Admin)",
160:             "request": {
161:                 "auth": {
162:                     "type": "bearer",
163:                     "bearer": [{ "key": "token", "value": "{{adminToken}}" }]
164:                 },
165:                 "method": "POST",
166:                 "header": [{ "key": "Content-Type", "value": "application/json" }],
167:                 "url": "{{baseUrl}}/features",
168:                 "body": {
169:                     "mode": "raw",
170:                     "raw": "{\n  \"featureName\": \"new-feature\",\n  \"description\": \"New feature description\"
171:                 }
172:             }
173:         },
174:         {
175:             "name": "Get Feature Stats (Admin)",
176:             "request": {
177:                 "auth": {
178:                     "type": "bearer",
179:                     "bearer": [{ "key": "token", "value": "{{adminToken}}" }]
180:                 },
181:                 "method": "GET",
182:                 "url": "{{baseUrl}}/features/stats"
183:             }
184:         }
185:     ]
186: },
187: {
188:     "name": "Rate Guards",
189:     "item": [
190:         {
191:             "name": "Get All Rate Guards",
192:             "request": {
193:                 "method": "GET",
194:                 "url": "{{baseUrl}}/rate-guards"
195:             }
196:         },
197:     ],

```

```

198:         "name": "Test Rate Limit",
199:         "request": {
200:             "method": "POST",
201:             "header": [{ "key": "Content-Type", "value": "application/json" }],
202:             "url": "{{baseUrl}}/rate-guards/test",
203:             "body": {
204:                 "mode": "raw",
205:                 "raw": "{\n  \"routePath\": \"/api/upload\", \n  \"method\": \"POST\"\n}"
206:             }
207:         },
208:     },
209:     {
210:         "name": "Create Rate Guard (Admin)",
211:         "request": {
212:             "auth": {
213:                 "type": "bearer",
214:                 "bearer": [{ "key": "token", "value": "{{adminToken}}" }]
215:             },
216:             "method": "POST",
217:             "header": [{ "key": "Content-Type", "value": "application/json" }],
218:             "url": "{{baseUrl}}/rate-guards",
219:             "body": {
220:                 "mode": "raw",
221:                 "raw": "{\n  \"routePath\": \"/api/test\", \n  \"method\": \"POST\", \n  \"description\": \"Test\""
222:             }
223:         }
224:     }
225: ],
226: },
227: {
228:     "name": "Users (Admin)",
229:     "item": [
230:         {
231:             "name": "Get All Users",
232:             "request": {
233:                 "auth": {
234:                     "type": "bearer",
235:                     "bearer": [{ "key": "token", "value": "{{adminToken}}" }]
236:                 },
237:                 "method": "GET",
238:                 "url": {
239:                     "raw": "{{baseUrl}}/users?page=1&limit=10",
240:                     "query": [
241:                         { "key": "page", "value": "1" },
242:                         { "key": "limit", "value": "10" }
243:                     ]
244:                 }
245:             }
246:         },
247:         {
248:             "name": "Get User Stats",
249:             "request": {
250:                 "auth": {
251:                     "type": "bearer",
252:                     "bearer": [{ "key": "token", "value": "{{adminToken}}" }]
253:                 },
254:                 "method": "GET",
255:                 "url": "{{baseUrl}}/users/stats"
256:             }
257:         }
258:     ]
259: },
260: {
261:     "name": "Audit (Admin)",
262:     "item": [
263:         {
264:             "name": "Get Audit Logs",
265:             "request": {
266:                 "auth": {
267:                     "type": "bearer",
268:                     "bearer": [{ "key": "token", "value": "{{adminToken}}" }]
269:                 },
270:                 "method": "GET",

```

```

271:         "url": "{{baseUrl}}/audit"
272:     },
273: },
274: {
275:     "name": "Get Failed Actions",
276:     "request": {
277:         "auth": {
278:             "type": "bearer",
279:             "bearer": [{ "key": "token", "value": "{{adminToken}}" }]
280:         },
281:         "method": "GET",
282:         "url": "{{baseUrl}}/audit/failed"
283:     }
284: },
285: {
286:     "name": "Get Audit Stats",
287:     "request": {
288:         "auth": {
289:             "type": "bearer",
290:             "bearer": [{ "key": "token", "value": "{{adminToken}}" }]
291:         },
292:         "method": "GET",
293:         "url": "{{baseUrl}}/audit/stats"
294:     }
295: }
296: ]
297: },
298: {
299:     "name": "System",
300:     "item": [
301:         {
302:             "name": "Health Check",
303:             "request": {
304:                 "auth": { "type": "noauth" },
305:                 "method": "GET",
306:                 "url": "{{baseUrl}}/health"
307:             }
308:         },
309:         {
310:             "name": "System Status",
311:             "request": {
312:                 "auth": { "type": "noauth" },
313:                 "method": "GET",
314:                 "url": "{{baseUrl}}/status"
315:             }
316:         }
317:     ]
318: }
319: ]
320: }

```

■ File: scripts\generate-test-docs.js

```

=====
1: const fs = require("fs");
2: const path = require("path");
3: const { execSync } = require("child_process");
4:
5: /**
6:  * Generate Test Documentation
7:  * Automatically creates a markdown document from test results
8:  */
9: class TestDocGenerator {
10:     constructor() {
11:         this.outputPath = path.join(__dirname, "../docs/TEST_RESULTS.md");
12:         this.docsDir = path.join(__dirname, "../docs");
13:     }
14:
15:     /**
16:      * Run tests and capture output
17:      */
18:     runTests() {

```

```

19:     console.log("Running test suite...\n");
20:
21:     try {
22:         const output = execSync("npm test -- --json", {
23:             encoding: "utf-8",
24:             stdio: ["pipe", "pipe", "pipe"],
25:         });
26:
27:         return JSON.parse(output);
28:     } catch (error) {
29:         // Tests may fail but we still want the results
30:         if (error.stdout) {
31:             try {
32:                 return JSON.parse(error.stdout);
33:             } catch (parseError) {
34:                 console.error("Failed to parse test output");
35:                 return null;
36:             }
37:         }
38:         return null;
39:     }
40: }
41:
42: /**
43:  * Generate markdown documentation
44:  */
45: generateMarkdown(testResults) {
46:     const timestamp = new Date().toISOString();
47:
48:     let markdown = `# Test Results Documentation\n\n`;
49:     markdown += `**Generated:** ${timestamp}\n\n`;
50:     markdown += `---\n\n`;
51:
52:     // Summary
53:     markdown += `## ? Summary\n\n`;
54:     markdown += `| Metric | Value |\n`;
55:     markdown += `|-----|-----|\n`;
56:     markdown += `| Total Tests | ${testResults.numTotalTests || 0} |\n`;
57:     markdown += `| Passed | ${testResults.numPassedTests || 0} |\n`;
58:     markdown += `| Failed | ${testResults.numFailedTests || 0} |\n`;
59:     markdown += `| Pending | ${testResults.numPendingTests || 0} |\n`;
60:     markdown += `| Success Rate | ${this.calculateSuccessRate(testResults)}% |\n`;
61:     markdown += `| Duration | ${this.formatDuration(testResults.testDuration || 0)} |\n\n`;
62:
63:     // Test Suites
64:     markdown += `## ? Test Suites\n\n`;
65:
66:     if (testResults.testResults) {
67:         const groupedTests = this.groupTestsByType(testResults.testResults);
68:
69:         markdown += `### Unit Tests\n\n`;
70:         markdown += this.formatTestGroup(groupedTests.unit);
71:
72:         markdown += `### Integration Tests\n\n`;
73:         markdown += this.formatTestGroup(groupedTests.integration);
74:
75:         markdown += `### System Tests\n\n`;
76:         markdown += this.formatTestGroup(groupedTests.system);
77:     }
78:
79:     // Coverage
80:     if (testResults.coverage) {
81:         markdown += `## ? Coverage\n\n`;
82:         markdown += this.formatCoverage(testResults.coverage);
83:     }
84:
85:     // Test Categories
86:     markdown += `## ? Test Categories\n\n`;
87:     markdown += `~ **Unit Tests**: Test individual components in isolation\n`;
88:     markdown += `~ **Integration Tests**: Test API endpoints and middleware integration\n`;
89:     markdown += `~ **System Tests**: Test complete workflows end-to-end\n\n`;
90:
91:     // Areas Covered

```

```

92:     markdown += `## ? Areas Covered\n\n`;
93:     markdown += ` - User Authentication & Authorization\n`;
94:     markdown += ` - Feature Toggle Management\n`;
95:     markdown += ` - Rate Guard (API Rate Limiting)\n`;
96:     markdown += ` - Audit Logging\n`;
97:     markdown += ` - User Management\n`;
98:     markdown += ` - Model Validation\n`;
99:     markdown += ` - Middleware Enforcement\n`;
100:    markdown += ` - Error Handling\n\n`;
101:
102:    markdown += `---\n\n`;
103:    markdown += `*This document is automatically generated from test execution results.*\n\n`;
104:
105:    return markdown;
106:  }
107:
108:  /**
109:   * Group tests by type
110:   */
111:  groupTestsByType(testResults) {
112:    return {
113:      unit: testResults.filter((t) => t.name.includes("/unit/")),
114:      integration: testResults.filter((t) => t.name.includes("/integration/")),
115:      system: testResults.filter((t) => t.name.includes("/system/")),
116:    };
117:  }
118:
119:  /**
120:   * Format test group
121:   */
122:  formatTestGroup(tests) {
123:    if (!tests || tests.length === 0) {
124:      return `*No tests in this category*\n\n`;
125:    }
126:
127:    let output = "";
128:    tests.forEach((test) => {
129:      const fileName = path.basename(test.name);
130:      const status = test.status === "passed" ? "?" : "";
131:      const testCount = test.numPassingTests || 0;
132:      const totalTests =
133:        (test.numPassingTests || 0) + (test.numFailingTests || 0);
134:
135:      output += `**${status} ${fileName}**\n`;
136:      output += ` - Tests: ${testCount}/${totalTests} passed\n`;
137:      output += ` - Duration: ${this.formatDuration(test.perfStats?.runtime || 0)}\n\n`;
138:    });
139:
140:    return output;
141:  }
142:
143:  /**
144:   * Format coverage information
145:   */
146:  formatCoverage(coverage) {
147:    let output = "| Category | Percentage |\n";
148:    output += "|-----|-----|\n";
149:    output += `| Statements | ${coverage.statements || 0}% |\n`;
150:    output += `| Branches | ${coverage.branches || 0}% |\n`;
151:    output += `| Functions | ${coverage.functions || 0}% |\n`;
152:    output += `| Lines | ${coverage.lines || 0}% |\n\n`;
153:
154:    return output;
155:  }
156:
157:  /**
158:   * Calculate success rate
159:   */
160:  calculateSuccessRate(testResults) {
161:    const total = testResults.numTotalTests || 0;
162:    const passed = testResults.numPassedTests || 0;
163:
164:    if (total === 0) return 0;

```

```

165:     return ((passed / total) * 100).toFixed(2);
166: }
167:
168: /**
169:  * Format duration
170:  */
171: formatDuration(ms) {
172:     if (ms < 1000) return `${ms}ms`;
173:     return `${(ms / 1000).toFixed(2)}s`;
174: }
175:
176: /**
177:  * Ensure docs directory exists
178:  */
179: ensureDocsDir() {
180:     if (!fs.existsSync(this.docsDir)) {
181:         fs.mkdirSync(this.docsDir, { recursive: true });
182:     }
183: }
184:
185: /**
186:  * Save documentation
187:  */
188: saveDocumentation(markdown) {
189:     this.ensureDocsDir();
190:     fs.writeFileSync(this.outputPath, markdown, "utf-8");
191:     console.log(`\n? Test documentation generated: ${this.outputPath}`);
192: }
193:
194: /**
195:  * Main execution
196:  */
197: async generate() {
198:     console.log("? Test Documentation Generator\n");
199:
200:     // Run tests
201:     const testResults = this.runTests();
202:
203:     if (!testResults) {
204:         console.error("? Failed to get test results");
205:         process.exit(1);
206:     }
207:
208:     // Generate markdown
209:     const markdown = this.generateMarkdown(testResults);
210:
211:     // Save documentation
212:     this.saveDocumentation(markdown);
213:
214:     console.log("? Documentation generation complete!\n");
215: }
216: }
217:
218: // Run generator
219: if (require.main === module) {
220:     const generator = new TestDocGenerator();
221:     generator.generate().catch((error) => {
222:         console.error("Error generating documentation:", error);
223:         process.exit(1);
224:     });
225: }
226:
227: module.exports = TestDocGenerator;

```

■ File: scripts\seed.js

```

=====
1: const database = require("../src/config/database");
2: const logger = require("../src/utils/logger");
3: const { User, FeatureToggle, RateGuard, ROLES } = require("../src/models");
4: const config = require("../src/config");
5:

```



```

6: /**
7:  * Database Seeder
8:  * Populates database with sample data for development
9:  */
10: class DatabaseSeeder {
11:   async seed() {
12:     try {
13:       console.log("? Starting database seeding...\n");
14:
15:       // Connect to database
16:       await database.connect();
17:
18:       // Clear existing data
19:       await this.clearData();
20:
21:       // Seed data
22:       const admin = await this.seedUsers();
23:       await this.seedFeatureToggles(admin._id);
24:       await this.seedRateGuards(admin._id);
25:
26:       console.log("\n? Database seeding completed successfully!");
27:       console.log("\n? Sample Credentials:");
28:       console.log("  Admin: admin@example.com / Admin@123456");
29:       console.log("  User:  user@example.com / User@123456");
30:       console.log("  Guest: guest@example.com / Guest@123456\n");
31:
32:       await database.disconnect();
33:       process.exit(0);
34:     } catch (error) {
35:       console.error("? Seeding failed:", error);
36:       process.exit(1);
37:     }
38:   }
39:
40:   async clearData() {
41:     console.log("?? Clearing existing data...");
42:
43:     await User.deleteMany({});
44:     await FeatureToggle.deleteMany({});
45:     await RateGuard.deleteMany({});
46:
47:     console.log("  ? Data cleared");
48:   }
49:
50:   async seedUsers() {
51:     console.log("? Seeding users...");
52:
53:     // Create admin user
54:     const admin = await User.create({
55:       email: "admin@example.com",
56:       password: "Admin@123456",
57:       firstName: "System",
58:       lastName: "Administrator",
59:       role: ROLES.ADMIN,
60:       isActive: true,
61:     });
62:     console.log(`  ? Admin created: ${admin.email}`);
63:
64:     // Create regular user
65:     const user = await User.create({
66:       email: "user@example.com",
67:       password: "User@123456",
68:       firstName: "Regular",
69:       lastName: "User",
70:       role: ROLES.USER,
71:       isActive: true,
72:     });
73:     console.log(`  ? User created: ${user.email}`);
74:
75:     // Create guest user
76:     const guest = await User.create({
77:       email: "guest@example.com",
78:       password: "Guest@123456",

```

```

79:     firstName: "Guest",
80:     lastName: "User",
81:     role: ROLES.GUEST,
82:     isActive: true,
83:   });
84:   console.log(`    ? Guest created: ${guest.email}`);
85:
86:   return admin;
87: }
88:
89: async seedFeatureToggles(adminId) {
90:   console.log("?? Seeding feature toggles...");
91:
92:   const features = [
93:     {
94:       featureName: "premium-features",
95:       description: "Access to premium features",
96:       enabled: true,
97:       allowedRoles: [ROLES.ADMIN, ROLES.USER],
98:       rolloutPercentage: 100,
99:       environments: {
100:         development: { enabled: true },
101:         staging: { enabled: true },
102:         production: { enabled: false },
103:       },
104:       createdBy: adminId,
105:     },
106:     {
107:       featureName: "analytics-dashboard",
108:       description: "Advanced analytics dashboard",
109:       enabled: true,
110:       allowedRoles: [ROLES.ADMIN],
111:       rolloutPercentage: 100,
112:       environments: {
113:         development: { enabled: true },
114:         staging: { enabled: true },
115:         production: { enabled: true },
116:       },
117:       createdBy: adminId,
118:     },
119:     {
120:       featureName: "beta-features",
121:       description: "Early access to beta features",
122:       enabled: true,
123:       allowedRoles: [ROLES.ADMIN, ROLES.USER],
124:       rolloutPercentage: 50,
125:       environments: {
126:         development: { enabled: true },
127:         staging: { enabled: true },
128:         production: { enabled: false },
129:       },
130:       createdBy: adminId,
131:     },
132:     {
133:       featureName: "file-upload",
134:       description: "File upload functionality",
135:       enabled: true,
136:       allowedRoles: [ROLES.ADMIN, ROLES.USER, ROLES.GUEST],
137:       rolloutPercentage: 100,
138:       environments: {
139:         development: { enabled: true },
140:         staging: { enabled: true },
141:         production: { enabled: true },
142:       },
143:       createdBy: adminId,
144:     },
145:     {
146:       featureName: "advanced-search",
147:       description: "Advanced search capabilities",
148:       enabled: false,
149:       allowedRoles: [ROLES.ADMIN, ROLES.USER],
150:       rolloutPercentage: 0,
151:       environments: {

```

```

152:         development: { enabled: false },
153:         staging: { enabled: false },
154:         production: { enabled: false },
155:     },
156:     createdBy: adminId,
157: },
158: ];
159:
160: for (const feature of features) {
161:     await FeatureToggle.create(feature);
162:     console.log(`    ? Feature created: ${feature.featureName}`);
163: }
164: }
165:
166: async seedRateGuards(adminId) {
167:     console.log("?? Seeding rate guard rules...");
168:
169:     const rules = [
170:         {
171:             routePath: "/api/auth/login",
172:             method: "POST",
173:             description: "Login rate limiting",
174:             enabled: true,
175:             limits: {
176:                 admin: {
177:                     maxRequests: 100,
178:                     windowMs: 60000,
179:                 },
180:                 user: {
181:                     maxRequests: 10,
182:                     windowMs: 60000,
183:                 },
184:                 guest: {
185:                     maxRequests: 5,
186:                     windowMs: 60000,
187:                 },
188:             },
189:             errorMessage: "Too many login attempts. Please try again later.",
190:             createdBy: adminId,
191:         },
192:         {
193:             routePath: "/api/features",
194:             method: "POST",
195:             description: "Feature creation rate limiting",
196:             enabled: true,
197:             limits: {
198:                 admin: {
199:                     maxRequests: 50,
200:                     windowMs: 60000,
201:                 },
202:                 user: {
203:                     maxRequests: 0,
204:                     windowMs: 60000,
205:                 },
206:                 guest: {
207:                     maxRequests: 0,
208:                     windowMs: 60000,
209:                 },
210:             },
211:             createdBy: adminId,
212:         },
213:         {
214:             routePath: "/api/users",
215:             method: "GET",
216:             description: "User list rate limiting",
217:             enabled: true,
218:             limits: {
219:                 admin: {
220:                     maxRequests: 100,
221:                     windowMs: 60000,
222:                 },
223:                 user: {
224:                     maxRequests: 20,

```

```

225:         windowMs: 60000,
226:     },
227:     guest: {
228:         maxRequests: 5,
229:         windowMs: 60000,
230:     },
231: },
232: createdBy: adminId,
233: },
234: {
235:     routePath: "/api/audit",
236:     method: "ALL",
237:     description: "Audit log access rate limiting",
238:     enabled: true,
239:     limits: {
240:         admin: {
241:             maxRequests: 200,
242:             windowMs: 60000,
243:         },
244:         user: {
245:             maxRequests: 0,
246:             windowMs: 60000,
247:         },
248:         guest: {
249:             maxRequests: 0,
250:             windowMs: 60000,
251:         },
252:     },
253:     createdBy: adminId,
254: },
255: ];
256:
257: for (const rule of rules) {
258:     await RateGuard.create(rule);
259:     console.log(`    ? Rate guard created: ${rule.method} ${rule.routePath}`);
260: }
261: }
262: }
263:
264: // Run seeder
265: if (require.main === module) {
266:     const seeder = new DatabaseSeeder();
267:     seeder.seed();
268: }
269:
270: module.exports = DatabaseSeeder;

```

File: src\app.js

```

=====
1: const express = require("express");
2: const cors = require("cors");
3: const helmet = require("helmet");
4: const config = require("./config");
5: const logger = require("./utils/logger");
6: const routes = require("./routes");
7: const {
8:     httpLogger,
9:     requestTimer,
10:    auditLogger,
11:    requestContext,
12:    sanitize,
13:    applyRateGuard,
14:    errorHandler,
15:    notFound,
16: } = require("./middleware");
17:
18: /**
19:  * Create Express Application
20:  */
21: const app = express();
22:

```

```

23: /**
24:  * Security Middleware
25:  */
26: app.use(helmet());
27: app.use(cors(config.cors));
28:
29: /**
30:  * Request Processing Middleware
31:  */
32: app.use(express.json({ limit: "10mb" }));
33: app.use(express.urlencoded({ extended: true, limit: "10mb" }));
34:
35: /**
36:  * Logging Middleware
37:  */
38: app.use(httpLogger);
39: app.use(requestTimer);
40: app.use(requestContext);
41:
42: /**
43:  * Input Sanitization
44:  */
45: app.use(sanitize);
46:
47: /**
48:  * Trust proxy (for accurate IP addresses behind proxy/load balancer)
49:  */
50: app.set("trust proxy", 1);
51:
52: /**
53:  * Apply Dynamic Rate Guard Middleware
54:  * This checks database rules and applies rate limiting
55:  */
56: app.use(applyRateGuard);
57:
58: /**
59:  * Audit Logger (logs after response)
60:  */
61: app.use(auditLogger);
62:
63: /**
64:  * API Routes
65:  */
66: app.use("/api", routes);
67:
68: /**
69:  * Root Route
70:  */
71: app.get("/", (req, res) => {
72:   res.json({
73:     success: true,
74:     message: "Policy-Driven Feature Toggle & Rate Guard Service",
75:     version: "1.0.0",
76:     status: "operational",
77:     api: "/api",
78:     health: "/api/health",
79:     documentation: "See README.md",
80:   });
81: });
82:
83: /**
84:  * 404 Handler
85:  */
86: app.use(notFound);
87:
88: /**
89:  * Global Error Handler
90:  */
91: app.use(errorHandler);
92:
93: module.exports = app;

```

■ File: src\config\database.js

```
=====
1: const mongoose = require('mongoose');
2: const config = require('../config');
3: const logger = require('../utils/logger');
4:
5: /**
6:  * Database Connection Manager
7:  */
8: class Database {
9:   constructor() {
10:    this.connection = null;
11:   }
12:
13:   /**
14:    * Connect to MongoDB
15:    */
16:   async connect() {
17:     try {
18:       if (this.connection) {
19:         logger.warn('Database already connected');
20:         return this.connection;
21:       }
22:
23:       mongoose.set('strictQuery', false);
24:
25:       this.connection = await mongoose.connect(config.mongodb.uri, config.mongodb.options);
26:
27:       logger.info(`MongoDB connected successfully: ${config.mongodb.uri.split('@')[1] || 'localhost'}`);
28:
29:       // Handle connection events
30:       mongoose.connection.on('error', (err) => {
31:         logger.error('MongoDB connection error:', err);
32:       });
33:
34:       mongoose.connection.on('disconnected', () => {
35:         logger.warn('MongoDB disconnected');
36:       });
37:
38:       // Graceful shutdown
39:       process.on('SIGINT', async () => {
40:         await this.disconnect();
41:         process.exit(0);
42:       });
43:
44:       return this.connection;
45:     } catch (error) {
46:       logger.error('MongoDB connection failed:', error);
47:       throw error;
48:     }
49:   }
50:
51:   /**
52:    * Disconnect from MongoDB
53:    */
54:   async disconnect() {
55:     try {
56:       if (!this.connection) {
57:         return;
58:       }
59:
60:       await mongoose.connection.close();
61:       this.connection = null;
62:       logger.info('MongoDB disconnected successfully');
63:     } catch (error) {
64:       logger.error('MongoDB disconnect error:', error);
65:       throw error;
66:     }
67:   }
68:
69:   /**
70:    * Clear all collections (for testing)
71:    */
```

```

72:   async clearDatabase() {
73:     if (config.env !== 'test') {
74:       throw new Error('clearDatabase can only be used in test environment');
75:     }
76:
77:     const collections = mongoose.connection.collections;
78:     for (const key in collections) {
79:       await collections[key].deleteMany({});
80:     }
81:     logger.info('Database cleared');
82:   }
83:
84:   /**
85:    * Get connection status
86:    */
87:   isConnected() {
88:     return mongoose.connection.readyState === 1;
89:   }
90: }
91:
92: module.exports = new Database();

```

■ File: src\config\index.js

```

1: require("dotenv").config();
2:
3: /**
4:  * Centralized Application Configuration
5:  * All environment variables are validated and exported from here
6:  */
7:
8: const config = {
9:   // Server
10:  env: process.env.NODE_ENV || "development",
11:  port: parseInt(process.env.PORT, 10) || 3000,
12:  apiVersion: process.env.API_VERSION || "v1",
13:
14:  // Database
15:  mongodb: {
16:    uri:
17:      process.env.NODE_ENV === "test"
18:        ? process.env.MONGODB_TEST_URI
19:        : process.env.MONGODB_URI,
20:    options: {
21:      useNewUrlParser: true,
22:      useUnifiedTopology: true,
23:    },
24:  },
25:
26:  // JWT
27:  jwt: {
28:    secret: process.env.JWT_SECRET || "fallback-secret-key",
29:    expiresIn: process.env.JWT_EXPIRES_IN || "24h",
30:    refreshExpiresIn: process.env.JWT_REFRESH_EXPIRES_IN || "7d",
31:  },
32:
33:  // Redis
34:  redis: {
35:    host: process.env.REDIS_HOST || "localhost",
36:    port: parseInt(process.env.REDIS_PORT, 10) || 6379,
37:    password: process.env.REDIS_PASSWORD || undefined,
38:  },
39:
40:  // Rate Limiting
41:  rateLimit: {
42:    windowMs: parseInt(process.env.RATE_LIMIT_WINDOW_MS, 10) || 60000,
43:    maxRequests: parseInt(process.env.RATE_LIMIT_MAX_REQUESTS, 10) || 100,
44:  },
45:
46:  // Logging
47:  logging: {

```

```

48:     level: process.env.LOG_LEVEL || "info",
49:     file: process.env.LOG_FILE || "logs/app.log",
50:   },
51:
52:   // CORS
53:   cors: {
54:     origin: process.env.CORS_ORIGIN || "http://localhost:3001",
55:     credentials: true,
56:   },
57:
58:   // Admin
59:   admin: {
60:     email: process.env.ADMIN_EMAIL || "admin@example.com",
61:     password: process.env.ADMIN_PASSWORD || "Admin@123456",
62:   },
63:
64:   // Feature Flags (Default System Features)
65:   features: {
66:     enableAuditLog: true,
67:     enableRateGuard: true,
68:     enableFeatureToggle: true,
69:   },
70: };
71:
72: // Validation
73: const requiredEnvVars = ["JWT_SECRET"];
74: const missingEnvVars = requiredEnvVars.filter(
75:   (varName) => !process.env[varName],
76: );
77:
78: if (missingEnvVars.length > 0 && config.env === "production") {
79:   throw new Error(
80:     `Missing required environment variables: ${missingEnvVars.join(", ")}`,
81:   );
82: }
83:
84: module.exports = config;

```

File: src\index.js

```

=====
1: const app = require("./app");
2: const config = require("./config");
3: const database = require("./config/database");
4: const logger = require("./utils/logger");
5: const { User, ROLES } = require("./models");
6:
7: /**
8:  * Start Server
9:  */
10: const startServer = async () => {
11:   try {
12:     // Connect to database
13:     await database.connect();
14:     logger.info("Database connection established");
15:
16:     // Create default admin user if not exists
17:     await createDefaultAdmin();
18:
19:     // Start Express server
20:     const server = app.listen(config.port, () => {
21:       logger.info(`Server started successfully`);
22:       logger.info(`Environment: ${config.env}`);
23:       logger.info(`Port: ${config.port}`);
24:       logger.info(`API: http://localhost:${config.port}/api`);
25:       logger.info(`Health: http://localhost:${config.port}/api/health`);
26:     });
27:
28:     // Graceful shutdown
29:     process.on("SIGTERM", () => gracefulShutdown(server));
30:     process.on("SIGINT", () => gracefulShutdown(server));
31:   } catch (error) {

```



```

32:     logger.error("Failed to start server:", error);
33:     process.exit(1);
34: }
35: };
36:
37: /**
38:  * Create Default Admin User
39:  */
40: const createDefaultAdmin = async () => {
41:     try {
42:         const adminExists = await User.findOne({ role: ROLES.ADMIN });
43:
44:         if (!adminExists) {
45:             const admin = await User.create({
46:                 email: config.admin.email,
47:                 password: config.admin.password,
48:                 role: ROLES.ADMIN,
49:                 firstName: "System",
50:                 lastName: "Administrator",
51:                 isActive: true,
52:             });
53:
54:             logger.info(`Default admin user created: ${admin.email}`);
55:             logger.warn(`Please change the default admin password immediately!`);
56:         } else {
57:             logger.info("Admin user already exists");
58:         }
59:     } catch (error) {
60:         logger.error("Failed to create default admin:", error);
61:     }
62: };
63:
64: /**
65:  * Graceful Shutdown
66:  */
67: const gracefulShutdown = async (server) => {
68:     logger.info("Received shutdown signal, closing server gracefully...");
69:
70:     // Stop accepting new requests
71:     server.close(async () => {
72:         logger.info("HTTP server closed");
73:
74:         try {
75:             // Close database connection
76:             await database.disconnect();
77:             logger.info("Database connection closed");
78:
79:             logger.info("Graceful shutdown completed");
80:             process.exit(0);
81:         } catch (error) {
82:             logger.error("Error during shutdown:", error);
83:             process.exit(1);
84:         }
85:     });
86:
87:     // Force shutdown after 30 seconds
88:     setTimeout(() => {
89:         logger.error("Forcing shutdown due to timeout");
90:         process.exit(1);
91:     }, 30000);
92: };
93:
94: // Start the server
95: startServer();

```

■ File: src\middleware\auth.js

```

=====
1: const jwt = require("jsonwebtoken");
2: const config = require("../config");
3: const { User } = require("../models");
4: const logger = require("../utils/logger");

```

```

5:
6: /**
7:  * Authentication Middleware
8:  * Verifies JWT token and attaches user to request
9:  */
10: const authenticate = async (req, res, next) => {
11:   try {
12:     // Get token from header
13:     const authHeader = req.headers.authorization;
14:
15:     if (!authHeader || !authHeader.startsWith("Bearer ")) {
16:       return res.status(401).json({
17:         success: false,
18:         message: "Authentication required. Please provide a valid token.",
19:       });
20:     }
21:
22:     const token = authHeader.substring(7); // Remove 'Bearer ' prefix
23:
24:     // Verify token
25:     let decoded;
26:     try {
27:       decoded = jwt.verify(token, config.jwt.secret);
28:     } catch (error) {
29:       if (error.name === "TokenExpiredError") {
30:         return res.status(401).json({
31:           success: false,
32:           message: "Token has expired. Please login again.",
33:         });
34:       }
35:
36:       if (error.name === "JsonWebTokenError") {
37:         return res.status(401).json({
38:           success: false,
39:           message: "Invalid token. Please login again.",
40:         });
41:       }
42:
43:       throw error;
44:     }
45:
46:     // Find user
47:     const user = await User.findById(decoded.userId);
48:
49:     if (!user) {
50:       return res.status(401).json({
51:         success: false,
52:         message: "User not found. Token may be invalid.",
53:       });
54:     }
55:
56:     // Check if user is active
57:     if (!user.isActive) {
58:       return res.status(403).json({
59:         success: false,
60:         message: "Account has been deactivated. Please contact support.",
61:       });
62:     }
63:
64:     // Check if account is locked
65:     if (user.isLocked()) {
66:       return res.status(423).json({
67:         success: false,
68:         message:
69:           "Account is temporarily locked due to multiple failed login attempts.",
70:       });
71:     }
72:
73:     // Attach user to request
74:     req.user = user;
75:     req.token = token;
76:
77:     next();

```

```

78:   } catch (error) {
79:     logger.error("Authentication error:", error);
80:     res.status(500).json({
81:       success: false,
82:       message: "Authentication failed. Please try again.",
83:     });
84:   }
85: };
86:
87: /**
88:  * Optional Authentication
89:  * Attaches user if token is valid, but doesn't require it
90:  */
91: const optionalAuth = async (req, res, next) => {
92:   try {
93:     const authHeader = req.headers.authorization;
94:
95:     if (!authHeader || !authHeader.startsWith("Bearer ")) {
96:       req.user = null;
97:       return next();
98:     }
99:
100:    const token = authHeader.substring(7);
101:
102:    try {
103:      const decoded = jwt.verify(token, config.jwt.secret);
104:      const user = await User.findById(decoded.userId);
105:
106:      if (user && user.isActive && !user.isLocked()) {
107:        req.user = user;
108:        req.token = token;
109:      }
110:    } catch (error) {
111:      // Silently ignore invalid tokens for optional auth
112:      req.user = null;
113:    }
114:
115:    next();
116:  } catch (error) {
117:    logger.error("Optional authentication error:", error);
118:    req.user = null;
119:    next();
120:  }
121: };
122:
123: /**
124:  * Generate JWT token
125:  */
126: const generateToken = (userId, expiresIn = config.jwt.expiresIn) => {
127:   return jwt.sign({ userId }, config.jwt.secret, { expiresIn });
128: };
129:
130: /**
131:  * Generate refresh token
132:  */
133: const generateRefreshToken = (userId) => {
134:   return jwt.sign({ userId, type: "refresh" }, config.jwt.secret, {
135:     expiresIn: config.jwt.refreshExpiresIn,
136:   });
137: };
138:
139: /**
140:  * Verify refresh token
141:  */
142: const verifyRefreshToken = (token) => {
143:   try {
144:     const decoded = jwt.verify(token, config.jwt.secret);
145:     if (decoded.type !== "refresh") {
146:       throw new Error("Invalid token type");
147:     }
148:     return decoded;
149:   } catch (error) {
150:     throw new Error("Invalid or expired refresh token");

```

```

151:   }
152: };
153:
154: module.exports = {
155:   authenticate,
156:   optionalAuth,
157:   generateToken,
158:   generateRefreshToken,
159:   verifyRefreshToken,
160: };

```

File: src\middleware\authorization.js

```

=====
1: const { ROLES } = require("../models");
2: const {
3:   AuditLog,
4:   AUDIT_ACTIONS,
5:   RESOURCE_TYPES,
6: } = require("../models/AuditLog");
7: const logger = require("../utils/logger");
8:
9: /**
10:  * Role-based Authorization Middleware
11:  * Restricts access based on user roles
12:  */
13: const authorize = (...allowedRoles) => {
14:   return async (req, res, next) => {
15:     try {
16:       // User must be authenticated first
17:       if (!req.user) {
18:         return res.status(401).json({
19:           success: false,
20:           message: "Authentication required",
21:         });
22:       }
23:
24:       // Check if user's role is in allowed roles
25:       if (!allowedRoles.includes(req.user.role)) {
26:         // Log access denial
27:         await AuditLog.log({
28:           action: AUDIT_ACTIONS.ACCESS_DENIED,
29:           resourceType: RESOURCE_TYPES.API,
30:           userId: req.user._id,
31:           userEmail: req.user.email,
32:           userRole: req.user.role,
33:           success: false,
34:           metadata: {
35:             ip: req.ip,
36:             userAgent: req.get("user-agent"),
37:             method: req.method,
38:             path: req.path,
39:             requiredRoles: allowedRoles,
40:           },
41:           details: `Access denied - required roles: ${allowedRoles.join(", ")}`,
42:         });
43:
44:         return res.status(403).json({
45:           success: false,
46:           message:
47:             "Insufficient permissions. This action requires higher privileges.",
48:           requiredRoles: allowedRoles,
49:         });
50:       }
51:
52:       next();
53:     } catch (error) {
54:       logger.error("Authorization error:", error);
55:       res.status(500).json({
56:         success: false,
57:         message: "Authorization check failed",
58:       });

```

```

59:     }
60:   };
61: };
62:
63: /**
64:  * Admin Only Middleware
65:  */
66: const adminOnly = authorize(ROLES.ADMIN);
67:
68: /**
69:  * Admin or User Middleware
70:  */
71: const authenticatedUsers = authorize(ROLES.ADMIN, ROLES.USER);
72:
73: /**
74:  * Check if user owns resource
75:  */
76: const isOwner = (resourceUserIdField = "userId") => {
77:   return async (req, res, next) => {
78:     try {
79:       if (!req.user) {
80:         return res.status(401).json({
81:           success: false,
82:           message: "Authentication required",
83:         });
84:       }
85:
86:       // Admins can access any resource
87:       if (req.user.role === ROLES.ADMIN) {
88:         return next();
89:       }
90:
91:       // Get resource from request (could be in params, body, or attached by previous middleware)
92:       const resource = req.resource || req.body;
93:       const resourceUserId = resource?.[resourceUserIdField];
94:
95:       if (!resourceUserId) {
96:         return res.status(400).json({
97:           success: false,
98:           message: "Resource ownership could not be determined",
99:         });
100:      }
101:
102:      // Check if user owns the resource
103:      if (resourceUserId.toString() !== req.user._id.toString()) {
104:        await AuditLog.log({
105:          action: AUDIT_ACTIONS.ACCESS_DENIED,
106:          resourceType: RESOURCE_TYPES.API,
107:          userId: req.user._id,
108:          userEmail: req.user.email,
109:          success: false,
110:          metadata: {
111:            ip: req.ip,
112:            path: req.path,
113:            method: req.method,
114:          },
115:          details: "Access denied - not resource owner",
116:        });
117:
118:        return res.status(403).json({
119:          success: false,
120:          message: "You do not have permission to access this resource",
121:        });
122:      }
123:
124:      next();
125:    } catch (error) {
126:      logger.error("Ownership check error:", error);
127:      res.status(500).json({
128:        success: false,
129:        message: "Ownership verification failed",
130:      });
131:    }

```

```

132:   };
133: };
134:
135: /**
136:  * Permission check based on custom logic
137:  */
138: const hasPermission = (permissionCheck) => {
139:   return async (req, res, next) => {
140:     try {
141:       if (!req.user) {
142:         return res.status(401).json({
143:           success: false,
144:           message: "Authentication required",
145:         });
146:       }
147:
148:       const hasAccess = await permissionCheck(req.user, req);
149:
150:       if (!hasAccess) {
151:         await AuditLog.log({
152:           action: AUDIT_ACTIONS.ACCESS_DENIED,
153:           resourceType: RESOURCE_TYPES.API,
154:           userId: req.user._id,
155:           userEmail: req.user.email,
156:           success: false,
157:           metadata: {
158:             ip: req.ip,
159:             path: req.path,
160:             method: req.method,
161:           },
162:           details: "Custom permission check failed",
163:         });
164:
165:         return res.status(403).json({
166:           success: false,
167:           message: "You do not have permission to perform this action",
168:         });
169:       }
170:
171:       next();
172:     } catch (error) {
173:       logger.error("Permission check error:", error);
174:       res.status(500).json({
175:         success: false,
176:         message: "Permission verification failed",
177:       });
178:     }
179:   };
180: };
181:
182: module.exports = {
183:   authorize,
184:   adminOnly,
185:   authenticatedUsers,
186:   isOwner,
187:   hasPermission,
188:   ROLES,
189: };

```

■ File: src\middleware\errorHandler.js

```

=====
1: const logger = require("../utils/logger");
2: const config = require("../config");
3:
4: /**
5:  * Custom Application Error Class
6:  */
7: class AppError extends Error {
8:   constructor(message, statusCode = 500, errors = null) {
9:     super(message);
10:    this.statusCode = statusCode;

```

```

11:     this.status = `${statusCode}`.startsWith("4") ? "fail" : "error";
12:     this.isOperational = true;
13:     this.errors = errors;
14:
15:     Error.captureStackTrace(this, this.constructor);
16:   }
17: }
18:
19: /**
20:  * Handle Mongoose Validation Errors
21:  */
22: const handleValidationError = (err) => {
23:   const errors = Object.values(err.errors).map((error) => ({
24:     field: error.path,
25:     message: error.message,
26:   }));
27:
28:   return new AppError("Validation failed", 400, errors);
29: };
30:
31: /**
32:  * Handle Mongoose Duplicate Key Errors
33:  */
34: const handleDuplicateKeyError = (err) => {
35:   const field = Object.keys(err.keyValue)[0];
36:   const value = err.keyValue[field];
37:
38:   return new AppError(
39:     `Duplicate value for field '${field}': ${value}. Please use another value.`,
40:     409,
41:   );
42: };
43:
44: /**
45:  * Handle Mongoose Cast Errors
46:  */
47: const handleCastError = (err) => {
48:   return new AppError(`Invalid ${err.path}: ${err.value}`, 400);
49: };
50:
51: /**
52:  * Handle JWT Errors
53:  */
54: const handleJWTError = () => {
55:   return new AppError("Invalid token. Please login again.", 401);
56: };
57:
58: const handleJWTExpiredError = () => {
59:   return new AppError("Your token has expired. Please login again.", 401);
60: };
61:
62: /**
63:  * Send Error Response in Development
64:  */
65: const sendErrorDev = (err, res) => {
66:   res.status(err.statusCode).json({
67:     success: false,
68:     message: err.message,
69:     error: err,
70:     stack: err.stack,
71:     errors: err.errors,
72:   });
73: };
74:
75: /**
76:  * Send Error Response in Production
77:  */
78: const sendErrorProd = (err, res) => {
79:   // Operational, trusted error: send message to client
80:   if (err.isOperational) {
81:     res.status(err.statusCode).json({
82:       success: false,
83:       message: err.message,

```

```

84:     errors: err.errors,
85:   });
86: } else {
87:   // Programming or unknown error: don't leak error details
88:   logger.error("ERROR:", err);
89:
90:   res.status(500).json({
91:     success: false,
92:     message: "Something went wrong. Please try again later.",
93:   });
94: }
95: };
96:
97: /**
98:  * Global Error Handler Middleware
99:  */
100: const errorHandler = (err, req, res, next) => {
101:   err.statusCode = err.statusCode || 500;
102:   err.status = err.status || "error";
103:
104:   // Log error
105:   if (err.statusCode >= 500) {
106:     logger.error("Server Error:", {
107:       message: err.message,
108:       stack: err.stack,
109:       url: req.originalUrl,
110:       method: req.method,
111:       ip: req.ip,
112:       user: req.user?.email,
113:     });
114:   } else {
115:     logger.warn("Client Error:", {
116:       message: err.message,
117:       url: req.originalUrl,
118:       method: req.method,
119:       statusCode: err.statusCode,
120:       user: req.user?.email,
121:     });
122:   }
123:
124:   if (config.env === "development") {
125:     sendErrorDev(err, res);
126:   } else {
127:     let error = { ...err };
128:     error.message = err.message;
129:
130:     // Handle specific error types
131:     if (err.name === "ValidationError") error = handleValidationError(err);
132:     if (err.code === 11000) error = handleDuplicateKeyError(err);
133:     if (err.name === "CastError") error = handleCastError(err);
134:     if (err.name === "JsonWebTokenError") error = handleJWTError();
135:     if (err.name === "TokenExpiredError") error = handleJWTExpiredError();
136:
137:     sendErrorProd(error, res);
138:   }
139: };
140:
141: /**
142:  * Catch async errors
143:  */
144: const catchAsync = (fn) => {
145:   return (req, res, next) => {
146:     Promise.resolve(fn(req, res, next)).catch(next);
147:   };
148: };
149:
150: /**
151:  * Handle 404 - Not Found
152:  */
153: const notFound = (req, res, next) => {
154:   const error = new AppError(
155:     `Cannot ${req.method} ${req.originalUrl} - Route not found`,
156:     404,

```



```

157:   });
158:   next(error);
159: };
160:
161: /**
162:  * Handle Unhandled Promise Rejections
163:  */
164: process.on("unhandledRejection", (err) => {
165:   logger.error("UNHANDLED REJECTION! Shutting down...");
166:   logger.error(err.name, err.message);
167:   logger.error(err.stack);
168:
169:   process.exit(1);
170: });
171:
172: /**
173:  * Handle Uncaught Exceptions
174:  */
175: process.on("uncaughtException", (err) => {
176:   logger.error("UNCAUGHT EXCEPTION! Shutting down...");
177:   logger.error(err.name, err.message);
178:   logger.error(err.stack);
179:
180:   process.exit(1);
181: });
182:
183: module.exports = {
184:   AppError,
185:   errorHandler,
186:   catchAsync,
187:   notFound,
188: };

```

File: src\middleware\featureToggle.js

```

1: const FeatureToggle = require("../models/FeatureToggle");
2: const {
3:   AuditLog,
4:   AUDIT_ACTIONS,
5:   RESOURCE_TYPES,
6: } = require("../models/AuditLog");
7: const { ROLES } = require("../models");
8: const config = require("../config");
9: const logger = require("../utils/logger");
10:
11: /**
12:  * Feature Toggle Middleware
13:  * Checks if a feature is enabled for the current user
14:  */
15: const requireFeature = (featureName) => {
16:   return async (req, res, next) => {
17:     try {
18:       // Determine user role (guest if not authenticated)
19:       const userRole = req.user?.role || ROLES.GUEST;
20:       const userId = req.user?._id;
21:       const environment = config.env;
22:
23:       // Check if feature is enabled
24:       const isEnabled = await FeatureToggle.checkFeature(
25:         featureName,
26:         userRole,
27:         environment,
28:         userId,
29:       );
30:
31:       if (!isEnabled) {
32:         // Log access denial
33:         await AuditLog.log({
34:           action: AUDIT_ACTIONS.ACCESS_DENIED,
35:           resourceType: RESOURCE_TYPES.FEATURE_TOGGLE,
36:           resourceName: featureName,

```

```

37:         userId: req.user?._id,
38:         userEmail: req.user?.email,
39:         userRole,
40:         success: false,
41:         metadata: {
42:             ip: req.ip,
43:             userAgent: req.get("user-agent"),
44:             method: req.method,
45:             path: req.path,
46:             environment,
47:         },
48:         details: `Feature '${featureName}' is not enabled for role '${userRole}'`,
49:     });
50:
51:     return res.status(403).json({
52:         success: false,
53:         message: `Feature '${featureName}' is not available`,
54:         featureName,
55:         reason: "Feature is disabled or not available for your account",
56:     });
57: }
58:
59: // Feature is enabled, proceed
60: req.enabledFeature = featureName;
61: next();
62: } catch (error) {
63:     logger.error(`Feature toggle check failed for '${featureName}':`, error);
64:
65:     // Fail-safe: allow access if feature toggle system fails
66:     // This prevents feature toggle failures from breaking the entire app
67:     logger.warn(
68:         `Allowing access to '${featureName}' due to feature toggle system failure`,
69:     );
70:     next();
71: }
72: };
73: };
74:
75: /**
76:  * Check multiple features (all must be enabled)
77:  */
78: const requireAllFeatures = (...featureNames) => {
79:     return async (req, res, next) => {
80:         try {
81:             const userRole = req.user?.role || ROLES.GUEST;
82:             const userId = req.user?._id;
83:             const environment = config.env;
84:
85:             // Check all features
86:             const checks = await Promise.all(
87:                 featureNames.map((name) =>
88:                     FeatureToggle.checkFeature(name, userRole, environment, userId),
89:                 ),
90:             );
91:
92:             const allEnabled = checks.every(Boolean);
93:
94:             if (!allEnabled) {
95:                 const disabledFeatures = featureNames.filter(
96:                     (_, index) => !checks[index],
97:                 );
98:
99:                 await AuditLog.log({
100:                     action: AUDIT_ACTIONS.ACCESS_DENIED,
101:                     resourceType: RESOURCE_TYPES.FEATURE_TOGGLE,
102:                     userId: req.user?._id,
103:                     userEmail: req.user?.email,
104:                     userRole,
105:                     success: false,
106:                     metadata: {
107:                         ip: req.ip,
108:                         path: req.path,
109:                         method: req.method,

```

```

110:         disabledFeatures,
111:     },
112:     details: `Required features not enabled: ${disabledFeatures.join(", ")}`,
113: });
114:
115:     return res.status(403).json({
116:         success: false,
117:         message: "Some required features are not available",
118:         disabledFeatures,
119:     });
120: }
121:
122:     req.enabledFeatures = featureNames;
123:     next();
124: } catch (error) {
125:     logger.error("Multiple feature toggle check failed:", error);
126:     next();
127: }
128: };
129: };
130:
131: /**
132:  * Check if any of the features is enabled
133:  */
134: const requireAnyFeature = (...featureNames) => {
135:     return async (req, res, next) => {
136:         try {
137:             const userRole = req.user?.role || ROLES.GUEST;
138:             const userId = req.user?._id;
139:             const environment = config.env;
140:
141:             // Check all features
142:             const checks = await Promise.all(
143:                 featureNames.map((name) =>
144:                     FeatureToggle.checkFeature(name, userRole, environment, userId),
145:                 ),
146:             );
147:
148:             const anyEnabled = checks.some(Boolean);
149:
150:             if (!anyEnabled) {
151:                 await AuditLog.log({
152:                     action: AUDIT_ACTIONS.ACCESS_DENIED,
153:                     resourceType: RESOURCE_TYPES.FEATURE_TOGGLE,
154:                     userId: req.user?._id,
155:                     userEmail: req.user?.email,
156:                     userRole,
157:                     success: false,
158:                     metadata: {
159:                         ip: req.ip,
160:                         path: req.path,
161:                         method: req.method,
162:                         requiredFeatures: featureNames,
163:                     },
164:                     details: `None of the required features are enabled: ${featureNames.join(", ")}`,
165:                 });
166:
167:                 return res.status(403).json({
168:                     success: false,
169:                     message: "None of the required features are available",
170:                     requiredFeatures: featureNames,
171:                 });
172:             }
173:
174:             const enabledFeatures = featureNames.filter((_, index) => checks[index]);
175:             req.enabledFeatures = enabledFeatures;
176:             next();
177:         } catch (error) {
178:             logger.error("Any feature toggle check failed:", error);
179:             next();
180:         }
181:     };
182: };

```

```

183:
184: /**
185:  * Get enabled features for current user (doesn't block, just adds info)
186:  */
187: const attachEnabledFeatures = async (req, res, next) => {
188:   try {
189:     const userRole = req.user?.role || ROLES.GUEST;
190:     const environment = config.env;
191:
192:     const enabledFeatures = await FeatureToggle.getEnabledFeatures(
193:       userRole,
194:       environment,
195:     );
196:     req.availableFeatures = enabledFeatures.map((f) => f.featureName);
197:
198:     next();
199:   } catch (error) {
200:     logger.error("Failed to attach enabled features:", error);
201:     req.availableFeatures = [];
202:     next();
203:   }
204: };
205:
206: module.exports = {
207:   requireFeature,
208:   requireAllFeatures,
209:   requireAnyFeature,
210:   attachEnabledFeatures,
211: };

```

File: src\middleware\index.js

```

1: /**
2:  * Middleware Index
3:  * Centralized export of all middleware modules
4:  */
5:
6: const auth = require("./auth");
7: const authorization = require("./authorization");
8: const featureToggle = require("./featureToggle");
9: const rateGuard = require("./rateGuard");
10: const validation = require("./validation");
11: const errorHandler = require("./errorHandler");
12: const requestLogger = require("./requestLogger");
13:
14: module.exports = {
15:   // Authentication
16:   ...auth,
17:
18:   // Authorization
19:   ...authorization,
20:
21:   // Feature Toggles
22:   ...featureToggle,
23:
24:   // Rate Limiting
25:   ...rateGuard,
26:
27:   // Validation
28:   ...validation,
29:
30:   // Error Handling
31:   ...errorHandler,
32:
33:   // Request Logging
34:   ...requestLogger,
35: };

```

■ File: src\middleware\rateGuard.js

```
=====
1: const rateLimit = require("express-rate-limit");
2: const RateGuard = require("../models/RateGuard");
3: const {
4:   AuditLog,
5:   AUDIT_ACTIONS,
6:   RESOURCE_TYPES,
7: } = require("../models/AuditLog");
8: const { ROLES } = require("../models");
9: const logger = require("../utils/logger");
10:
11: // In-memory store for rate limit instances (can be replaced with Redis)
12: const rateLimiters = new Map();
13:
14: /**
15:  * Dynamic Rate Guard Middleware
16:  * Applies rate limiting based on database rules
17:  */
18: const applyRateGuard = async (req, res, next) => {
19:   try {
20:     const routePath = req.route?.path || req.path;
21:     const method = req.method;
22:     const userRole = req.user?.role || ROLES.GUEST;
23:     const userId = req.user?._id?.toString();
24:     const userIp = req.ip;
25:
26:     // Find applicable rate guard rule
27:     const rule = await RateGuard.findRuleForRoute(routePath, method);
28:
29:     if (!rule) {
30:       // No rate limiting rule found, proceed
31:       return next();
32:     }
33:
34:     // Check if user/IP is whitelisted
35:     if (rule.isWhitelisted(userId) || rule.isWhitelisted(userIp)) {
36:       logger.debug(
37:         `Rate limit bypassed for whitelisted identifier: ${userId || userIp}`,
38:       );
39:       return next();
40:     }
41:
42:     // Get limit for user's role
43:     const limit = rule.getLimitForRole(userRole);
44:
45:     if (!limit) {
46:       // No limit configured, proceed
47:       return next();
48:     }
49:
50:     // Create unique key for this rate limiter
51:     const limiterKey = `${rule._id}_${userRole}`;
52:
53:     // Get or create rate limiter for this rule and role
54:     let limiter = rateLimiters.get(limiterKey);
55:
56:     if (!limiter) {
57:       limiter = rateLimit({
58:         windowMs: limit.windowMs,
59:         max: limit.maxRequests,
60:         message: {
61:           success: false,
62:           message: rule.errorMessage,
63:           retryAfter: Math.ceil(limit.windowMs / 1000),
64:           limit: {
65:             maxRequests: limit.maxRequests,
66:             windowMs: limit.windowMs,
67:           },
68:         },
69:         standardHeaders: true,
70:         legacyHeaders: false,
71:         // Key generator based on IP or user

```

```

72:     keyGenerator: (req) => {
73:         if (rule.ipBased) {
74:             return req.ip;
75:         }
76:         return req.user?._id?.toString() || req.ip;
77:     },
78:     // Custom handler for rate limit exceeded
79:     handler: async (req, res) => {
80:         // Log rate limit exceeded
81:         await AuditLog.log({
82:             action: AUDIT_ACTIONS.RATE_LIMIT_EXCEEDED,
83:             resourceType: RESOURCE_TYPES.API,
84:             resourceId: rule._id,
85:             resourceName: rule.displayName,
86:             userId: req.user?._id,
87:             userEmail: req.user?.email,
88:             userRole,
89:             success: false,
90:             metadata: {
91:                 ip: req.ip,
92:                 userAgent: req.get("user-agent"),
93:                 method: req.method,
94:                 path: req.path,
95:                 limit: limit.maxRequests,
96:                 windowMs: limit.windowMs,
97:             },
98:             details: `Rate limit exceeded: ${limit.maxRequests} requests per ${limit.windowMs}ms`,
99:         });
100:
101:         res.status(429).json({
102:             success: false,
103:             message: rule.errorMessage,
104:             retryAfter: Math.ceil(limit.windowMs / 1000),
105:             limit: {
106:                 maxRequests: limit.maxRequests,
107:                 windowMs: limit.windowMs,
108:                 windowSeconds: Math.ceil(limit.windowMs / 1000),
109:             },
110:         });
111:     },
112:     // Skip function for whitelisted users
113:     skip: (req) => {
114:         const skipUserId = req.user?._id?.toString();
115:         const skipIp = req.ip;
116:         return rule.isWhitelisted(skipUserId) || rule.isWhitelisted(skipIp);
117:     },
118: });
119:
120: rateLimiters.set(limiterKey, limiter);
121:
122: logger.debug(
123:     `Created new rate limiter for ${limiterKey}: ${limit.maxRequests} req/${limit.windowMs}ms`,
124: );
125: }
126:
127: // Apply the rate limiter
128: limiter(req, res, next);
129: } catch (error) {
130:     logger.error("Rate guard middleware error:", error);
131:
132:     // Fail-safe: don't block requests if rate limiting fails
133:     logger.warn("Bypassing rate limit due to error");
134:     next();
135: }
136: };
137:
138: /**
139:  * Clear rate limiter cache
140:  * Useful when rate guard rules are updated
141:  */
142: const clearRateLimiters = () => {
143:     rateLimiters.clear();
144:     logger.info("Rate limiter cache cleared");

```

```

145: };
146:
147: /**
148:  * Get active rate limiters count
149:  */
150: const getActiveLimiters = () => {
151:   return rateLimiters.size;
152: };
153:
154: /**
155:  * Apply specific rate guard by ID
156:  */
157: const applySpecificRateGuard = (ruleId) => {
158:   return async (req, res, next) => {
159:     try {
160:       const rule = await RateGuard.findById(ruleId);
161:
162:       if (!rule || !rule.enabled) {
163:         return next();
164:       }
165:
166:       const userRole = req.user?.role || ROLES.GUEST;
167:       const userId = req.user?._id?.toString();
168:       const userIp = req.ip;
169:
170:       // Check whitelist
171:       if (rule.isWhitelisted(userId) || rule.isWhitelisted(userIp)) {
172:         return next();
173:       }
174:
175:       // Get limit configuration
176:       const config = rule.toRateLimiterConfig(userRole);
177:
178:       if (!config) {
179:         return next();
180:       }
181:
182:       // Create and apply rate limiter
183:       const limiter = rateLimit(config);
184:       limiter(req, res, next);
185:     } catch (error) {
186:       logger.error("Specific rate guard error:", error);
187:       next();
188:     }
189:   };
190: };
191:
192: /**
193:  * Middleware to refresh rate limiters when rules change
194:  */
195: const refreshRateLimitersOnUpdate = (req, res, next) => {
196:   // Clear cache when rate guard rules are modified
197:   if (
198:     req.method === "POST" ||
199:     req.method === "PUT" ||
200:     req.method === "PATCH" ||
201:     req.method === "DELETE"
202:   ) {
203:     if (req.path.includes("/rate-guards")) {
204:       clearRateLimiters();
205:       logger.info("Rate limiters cache cleared due to rule update");
206:     }
207:   }
208:   next();
209: };
210:
211: module.exports = {
212:   applyRateGuard,
213:   clearRateLimiters,
214:   getActiveLimiters,
215:   applySpecificRateGuard,
216:   refreshRateLimitersOnUpdate,
217: };

```

■ File: src\middleware\requestLogger.js

```
=====
1: const morgan = require("morgan");
2: const logger = require("../utils/logger");
3: const { AuditLog } = require("../models");
4:
5: /**
6:  * Morgan HTTP request logger with Winston integration
7:  */
8: const httpLogger = morgan(
9:   ":method :url :status :res[content-length] - :response-time ms",
10:  {
11:    stream: logger.stream,
12:    skip: (req) => {
13:      // Skip logging for health check endpoints
14:      return req.url === "/health" || req.url === "/api/health";
15:    },
16:  },
17: );
18:
19: /**
20:  * Request timing middleware
21:  */
22: const requestTimer = (req, res, next) => {
23:   req.startTime = Date.now();
24:
25:   // Capture response finish event
26:   res.on("finish", () => {
27:     req.duration = Date.now() - req.startTime;
28:   });
29:
30:   next();
31: };
32:
33: /**
34:  * Log API access with audit trail
35:  */
36: const auditLogger = async (req, res, next) => {
37:   // Wait for response to finish
38:   res.on("finish", async () => {
39:     try {
40:       // Only log authenticated requests or failed requests
41:       if (req.user || res.statusCode >= 400) {
42:         await AuditLog.logApiAccess(req, res.statusCode, req.duration);
43:       }
44:     } catch (error) {
45:       // Don't break request flow if audit logging fails
46:       logger.error("Audit logging failed:", error);
47:     }
48:   });
49:
50:   next();
51: };
52:
53: /**
54:  * Request context middleware
55:  * Adds useful context to the request object
56:  */
57: const requestContext = (req, res, next) => {
58:   // Add request ID for tracking
59:   req.id = `req_${Date.now()}_${Math.random().toString(36).substr(2, 9)}`;
60:
61:   // Add request metadata
62:   req.context = {
63:     id: req.id,
64:     ip: req.ip,
65:     method: req.method,
66:     url: req.originalUrl,
67:     userAgent: req.get("user-agent"),
68:     referer: req.get("referer"),
69:     timestamp: new Date(),
70:   };
71:
=====
```



```

72: // Log request start
73: logger.debug(`[${req.id}] ${req.method} ${req.originalUrl}`, {
74:   ip: req.ip,
75:   userAgent: req.get("user-agent"),
76: });
77:
78: next();
79: };
80:
81: /**
82:  * Response logger
83:  * Logs response details
84:  */
85: const responseLogger = (req, res, next) => {
86:   const originalJson = res.json.bind(res);
87:
88:   res.json = function (data) {
89:     // Log response
90:     logger.debug(`[${req.id}] Response:`, {
91:       statusCode: res.statusCode,
92:       duration: req.duration,
93:       dataSize: JSON.stringify(data).length,
94:     });
95:
96:     return originalJson(data);
97:   };
98:
99:   next();
100: };
101:
102: /**
103:  * Security headers logger
104:  */
105: const logSecurityHeaders = (req, res, next) => {
106:   // Log suspicious requests
107:   const suspiciousPatterns = [
108:     /\.\/\./, // Path traversal
109:     /<script/i, // XSS attempt
110:     /union.*select/i, // SQL injection
111:     /javascript:/i, // JavaScript injection
112:   ];
113:
114:   const isSuspicious = suspiciousPatterns.some(
115:     (pattern) =>
116:       pattern.test(req.url) ||
117:       pattern.test(JSON.stringify(req.body)) ||
118:       pattern.test(JSON.stringify(req.query)),
119:   );
120:
121:   if (isSuspicious) {
122:     logger.warn("Suspicious request detected:", {
123:       id: req.id,
124:       ip: req.ip,
125:       url: req.url,
126:       method: req.method,
127:       body: req.body,
128:       query: req.query,
129:       userAgent: req.get("user-agent"),
130:     });
131:   }
132:
133:   next();
134: };
135:
136: /**
137:  * Rate limit logger
138:  */
139: const logRateLimit = (req, res, next) => {
140:   // Add rate limit info to response if available
141:   const rateLimit = {
142:     limit: res.getHeader("X-RateLimit-Limit"),
143:     remaining: res.getHeader("X-RateLimit-Remaining"),
144:     reset: res.getHeader("X-RateLimit-Reset"),

```

```

145:   };
146:
147:   if (rateLimit.limit) {
148:     req.rateLimit = rateLimit;
149:
150:     // Log if approaching rate limit
151:     if (rateLimit.remaining && parseInt(rateLimit.remaining) < 10) {
152:       logger.warn("Approaching rate limit:", {
153:         user: req.user?.email,
154:         ip: req.ip,
155:         remaining: rateLimit.remaining,
156:         limit: rateLimit.limit,
157:       });
158:     }
159:   }
160:
161:   next();
162: };
163:
164: /**
165:  * Create detailed access log
166:  */
167: const detailedAccessLog = (req, res, next) => {
168:   res.on("finish", () => {
169:     const log = {
170:       requestId: req.id,
171:       timestamp: req.context.timestamp,
172:       method: req.method,
173:       url: req.originalUrl,
174:       statusCode: res.statusCode,
175:       duration: req.duration,
176:       ip: req.ip,
177:       user: req.user
178:       ? {
179:         id: req.user._id,
180:         email: req.user.email,
181:         role: req.user.role,
182:       }
183:       : null,
184:       userAgent: req.get("user-agent"),
185:       referer: req.get("referrer"),
186:       contentLength: res.get("content-length"),
187:       rateLimit: req.rateLimit,
188:     };
189:
190:     // Log based on status code
191:     if (res.statusCode >= 500) {
192:       logger.error("Server Error:", log);
193:     } else if (res.statusCode >= 400) {
194:       logger.warn("Client Error:", log);
195:     } else {
196:       logger.info("Access:", log);
197:     }
198:   });
199:
200:   next();
201: };
202:
203: module.exports = {
204:   httpLogger,
205:   requestTimer,
206:   auditLogger,
207:   requestContext,
208:   responseLogger,
209:   logSecurityHeaders,
210:   logRateLimit,
211:   detailedAccessLog,
212: };

```

■ File: src\middleware\validation.js

```
=====
1: const Joi = require("joi");
2: const logger = require("../utils/logger");
3:
4: /**
5:  * Validate request using Joi schema
6:  */
7: const validate = (schema) => {
8:   return (req, res, next) => {
9:     const validationOptions = {
10:      abortEarly: false, // Return all errors, not just the first one
11:      allowUnknown: true, // Allow unknown keys in the request
12:      stripUnknown: true, // Remove unknown keys from validated data
13:    };
14:
15:    // Determine what to validate (body, query, params)
16:    const toValidate = {};
17:    if (schema.body) toValidate.body = req.body;
18:    if (schema.query) toValidate.query = req.query;
19:    if (schema.params) toValidate.params = req.params;
20:
21:    const { error, value } = Joi.object(schema).validate(
22:      toValidate,
23:      validationOptions,
24:    );
25:
26:    if (error) {
27:      const errors = error.details.map((detail) => ({
28:        field: detail.path.join("."),
29:        message: detail.message,
30:        type: detail.type,
31:      }));
32:
33:      logger.warn("Validation error:", { errors, path: req.path });
34:
35:      return res.status(400).json({
36:        success: false,
37:        message: "Validation failed",
38:        errors,
39:      });
40:    }
41:
42:    // Replace request data with validated data
43:    if (value.body) req.body = value.body;
44:    if (value.query) req.query = value.query;
45:    if (value.params) req.params = value.params;
46:
47:    next();
48:  };
49: };
50:
51: /**
52:  * Common Joi schemas
53:  */
54: const schemas = {
55:   // MongoDB ObjectId validation
56:   objectId: Joi.string()
57:     .regex(/^([0-9a-fA-F]{24})$/),
58:   .message("Invalid ID format"),
59:
60:   // Email validation
61:   email: Joi.string().email().toLowerCase().trim(),
62:
63:   // Password validation
64:   password: Joi.string().min(6).max(128),
65:
66:   // Pagination
67:   pagination: {
68:     page: Joi.number().integer().min(1).default(1),
69:     limit: Joi.number().integer().min(1).max(100).default(10),
70:     sort: Joi.string().default("-createdAt"),
71:     search: Joi.string().trim().allow(""),

```

```

72:   },
73:
74:   // Date range
75:   dateRange: {
76:     startDate: Joi.date().iso(),
77:     endDate: Joi.date().iso().greater(Joi.ref("startDate")),
78:   },
79:
80:   // User registration
81:   register: {
82:     body: Joi.object({
83:       email: Joi.string().email().required(),
84:       password: Joi.string().min(6).required(),
85:       firstName: Joi.string().trim().allow(""),
86:       lastName: Joi.string().trim().allow(""),
87:     }),
88:   },
89:
90:   // User login
91:   login: {
92:     body: Joi.object({
93:       email: Joi.string().email().required(),
94:       password: Joi.string().required(),
95:     }),
96:   },
97:
98:   // Feature toggle creation
99:   createFeatureToggle: {
100:    body: Joi.object({
101:      featureName: Joi.string().trim().required(),
102:      description: Joi.string().trim().allow(""),
103:      enabled: Joi.boolean().default(true),
104:      allowedRoles: Joi.array().items(
105:        Joi.string().valid("admin", "user", "guest"),
106:      ),
107:      rolloutPercentage: Joi.number().min(0).max(100).default(100),
108:      environments: Joi.object({
109:        development: Joi.object({ enabled: Joi.boolean() }),
110:        staging: Joi.object({ enabled: Joi.boolean() }),
111:        production: Joi.object({ enabled: Joi.boolean() }),
112:      }),
113:    }),
114:   },
115:
116:   // Rate guard creation
117:   createRateGuard: {
118:     body: Joi.object({
119:       routePath: Joi.string().trim().required(),
120:       method: Joi.string()
121:         .valid("GET", "POST", "PUT", "PATCH", "DELETE", "ALL")
122:         .default("ALL"),
123:       description: Joi.string().trim().allow(""),
124:       enabled: Joi.boolean().default(true),
125:       limits: Joi.object({
126:         admin: Joi.object({
127:           maxRequests: Joi.number().integer().min(1),
128:           windowMs: Joi.number().integer().min(1000),
129:         }),
130:         user: Joi.object({
131:           maxRequests: Joi.number().integer().min(1),
132:           windowMs: Joi.number().integer().min(1000),
133:         }),
134:         guest: Joi.object({
135:           maxRequests: Joi.number().integer().min(1),
136:           windowMs: Joi.number().integer().min(1000),
137:         }),
138:       }),
139:       ipBased: Joi.boolean().default(false),
140:       errorMessage: Joi.string().trim(),
141:     }),
142:   },
143:
144:   // Update user

```

```

145:   updateUser: {
146:     body: Joi.object({
147:       firstName: Joi.string().trim(),
148:       lastName: Joi.string().trim(),
149:       email: Joi.string().email(),
150:       role: Joi.string().valid("admin", "user", "guest"),
151:       isActive: Joi.boolean(),
152:     }).min(1),
153:   },
154:
155:   // ID parameter
156:   idParam: {
157:     params: Joi.object({
158:       id: Joi.string()
159:         .regex(/^[0-9a-fA-F]{24}$/)
160:         .required()
161:         .messages({
162:           "string.pattern.base": "Invalid ID format",
163:         }),
164:     }),
165:   },
166: };
167:
168: /**
169:  * Sanitize user input
170:  */
171: const sanitize = (req, res, next) => {
172:   // Basic XSS protection - strip potential script tags
173:   const sanitizeString = (str) => {
174:     if (typeof str !== "string") return str;
175:     return str
176:       .replace(/<script\b[^\<]*(?:(!</script>)<[^\<]*)*</script>/gi, "")
177:       .trim();
178:   };
179:
180:   const sanitizeObject = (obj) => {
181:     if (!obj || typeof obj !== "object") return obj;
182:
183:     Object.keys(obj).forEach((key) => {
184:       if (typeof obj[key] === "string") {
185:         obj[key] = sanitizeString(obj[key]);
186:       } else if (typeof obj[key] === "object") {
187:         obj[key] = sanitizeObject(obj[key]);
188:       }
189:     });
190:
191:     return obj;
192:   };
193:
194:   if (req.body) req.body = sanitizeObject(req.body);
195:   if (req.query) req.query = sanitizeObject(req.query);
196:   if (req.params) req.params = sanitizeObject(req.params);
197:
198:   next();
199: };
200:
201: module.exports = {
202:   validate,
203:   schemas,
204:   sanitize,
205: };

```

File: src\models\AuditLog.js

```

=====
1: const mongoose = require("mongoose");
2:
3: /**
4:  * Audit Log Actions Enum
5:  */
6: const AUDIT_ACTIONS = {
7:   CREATE: "create",

```

```

8:   UPDATE: "update",
9:   DELETE: "delete",
10:  LOGIN: "login",
11:  LOGOUT: "logout",
12:  ACCESS_DENIED: "access_denied",
13:  RATE_LIMIT_EXCEEDED: "rate_limit_exceeded",
14: };
15:
16: /**
17:  * Audit Log Resource Types
18:  */
19: const RESOURCE_TYPES = {
20:   USER: "user",
21:   FEATURE_TOGGLE: "feature_toggle",
22:   RATE_GUARD: "rate_guard",
23:   AUTH: "auth",
24:   API: "api",
25: };
26:
27: /**
28:  * Audit Log Schema
29:  * Tracks all significant system events and changes
30:  */
31: const auditLogSchema = new mongoose.Schema(
32:   {
33:     action: {
34:       type: String,
35:       required: [true, "Action is required"],
36:       enum: Object.values(AUDIT_ACTIONS),
37:       index: true,
38:     },
39:     resourceType: {
40:       type: String,
41:       required: [true, "Resource type is required"],
42:       enum: Object.values(RESOURCE_TYPES),
43:       index: true,
44:     },
45:     resourceId: {
46:       type: mongoose.Schema.Types.ObjectId,
47:       index: true,
48:     },
49:     resourceName: {
50:       type: String,
51:       trim: true,
52:     },
53:     userId: {
54:       type: mongoose.Schema.Types.ObjectId,
55:       ref: "User",
56:       index: true,
57:     },
58:     userEmail: {
59:       type: String,
60:       trim: true,
61:     },
62:     userRole: {
63:       type: String,
64:     },
65:     // What changed
66:     changes: {
67:       before: {
68:         type: mongoose.Schema.Types.Mixed,
69:       },
70:       after: {
71:         type: mongoose.Schema.Types.Mixed,
72:       },
73:     },
74:     // Request metadata
75:     metadata: {
76:       ip: String,
77:       userAgent: String,
78:       method: String,
79:       path: String,
80:       statusCode: Number,

```

```

81:         duration: Number, // milliseconds
82:         errorMessage: String,
83:     },
84:     // Success/failure
85:     success: {
86:         type: Boolean,
87:         default: true,
88:     },
89:     // Additional details
90:     details: {
91:         type: String,
92:         trim: true,
93:     },
94:     tags: [
95:         {
96:             type: String,
97:             trim: true,
98:         },
99:     ],
100: },
101: {
102:     timestamps: { createdAt: true, updatedAt: false },
103:     toJSON: { virtuals: true },
104:     toObject: { virtuals: true },
105: },
106: );
107:
108: /**
109:  * Indexes for common queries
110:  */
111: auditLogSchema.index({ createdAt: -1 });
112: auditLogSchema.index({ userId: 1, createdAt: -1 });
113: auditLogSchema.index({ resourceType: 1, resourceId: 1 });
114: auditLogSchema.index({ action: 1, createdAt: -1 });
115: auditLogSchema.index({ success: 1, createdAt: -1 });
116:
117: /**
118:  * TTL index - auto-delete logs older than 90 days
119:  */
120: auditLogSchema.index(
121:     { createdAt: 1 },
122:     { expireAfterSeconds: 90 * 24 * 60 * 60 },
123: );
124:
125: /**
126:  * Static method to log an event
127:  */
128: auditLogSchema.statics.log = async function (data) {
129:     try {
130:         const log = new this(data);
131:         await log.save();
132:         return log;
133:     } catch (error) {
134:         console.error("Failed to create audit log:", error);
135:         // Don't throw - audit logging should not break the application
136:         return null;
137:     }
138: };
139:
140: /**
141:  * Static method to log authentication events
142:  */
143: auditLogSchema.statics.logAuth = async function (
144:     action,
145:     userId,
146:     userEmail,
147:     success,
148:     metadata = {},
149: ) {
150:     return this.log({
151:         action,
152:         resourceType: RESOURCE_TYPES.AUTH,
153:         userId,

```

```

154:     userEmail,
155:     success,
156:     metadata,
157:     details: `User ${success ? "successfully" : "failed to"} ${action}`,
158:   });
159: };
160:
161: /**
162:  * Static method to log resource changes
163:  */
164: auditLogSchema.statics.logResourceChange = async function (
165:   action,
166:   resourceType,
167:   resourceId,
168:   userId,
169:   changes,
170:   metadata = {},
171: ) {
172:   return this.log({
173:     action,
174:     resourceType,
175:     resourceId,
176:     userId,
177:     changes,
178:     metadata,
179:     success: true,
180:   });
181: };
182:
183: /**
184:  * Static method to log API access
185:  */
186: auditLogSchema.statics.logApiAccess = async function (
187:   req,
188:   statusCode,
189:   duration,
190: ) {
191:   return this.log({
192:     action: AUDIT_ACTIONS.ACCESS_DENIED,
193:     resourceType: RESOURCE_TYPES.API,
194:     userId: req.user?._id,
195:     userEmail: req.user?.email,
196:     userRole: req.user?.role,
197:     metadata: {
198:       ip: req.ip,
199:       userAgent: req.get("user-agent"),
200:       method: req.method,
201:       path: req.path,
202:       statusCode,
203:       duration,
204:     },
205:     success: statusCode < 400,
206:   });
207: };
208:
209: /**
210:  * Get audit logs for a specific user
211:  */
212: auditLogSchema.statics.getUserLogs = async function (userId, limit = 50) {
213:   return this.find({ userId }).sort({ createdAt: -1 }).limit(limit).lean();
214: };
215:
216: /**
217:  * Get audit logs for a specific resource
218:  */
219: auditLogSchema.statics.getResourceLogs = async function (
220:   resourceType,
221:   resourceId,
222:   limit = 50,
223: ) {
224:   return this.find({ resourceType, resourceId })
225:     .sort({ createdAt: -1 })
226:     .limit(limit)

```



```

227:     .populate("userId", "email role")
228:     .lean();
229: };
230:
231: /**
232:  * Get failed actions
233:  */
234: auditLogSchema.statics.getFailedActions = async function (
235:     hours = 24,
236:     limit = 100,
237: ) {
238:     const since = new Date(Date.now() - hours * 60 * 60 * 1000);
239:
240:     return this.find({
241:         success: false,
242:         createdAt: { $gte: since },
243:     })
244:         .sort({ createdAt: -1 })
245:         .limit(limit)
246:         .lean();
247: };
248:
249: /**
250:  * Get statistics
251:  */
252: auditLogSchema.statics.getStats = async function (startDate, endDate) {
253:     return this.aggregate([
254:         {
255:             $match: {
256:                 createdAt: {
257:                     $gte: startDate,
258:                     $lte: endDate,
259:                 },
260:             },
261:         },
262:         {
263:             $group: {
264:                 _id: {
265:                     action: "$action",
266:                     resourceType: "$resourceType",
267:                     success: "$success",
268:                 },
269:                 count: { $sum: 1 },
270:             },
271:         },
272:         {
273:             $sort: { count: -1 },
274:         },
275:     ]);
276: };
277:
278: /**
279:  * Virtual for formatted timestamp
280:  */
281: auditLogSchema.virtual("formattedDate").get(function () {
282:     return this.createdAt.toISOString();
283: });
284:
285: const AuditLog = mongoose.model("AuditLog", auditLogSchema);
286:
287: module.exports = {
288:     AuditLog,
289:     AUDIT_ACTIONS,
290:     RESOURCE_TYPES,
291: };

```

■ File: src\models\FeatureToggle.js

```

=====
1: const mongoose = require("mongoose");
2: const { ROLES } = require("../User");
3:

```

```

4: /**
5:  * Feature Toggle Schema
6:  * Defines which features are accessible to which roles
7:  */
8: const featureToggleSchema = new mongoose.Schema(
9:   {
10:    featureName: {
11:      type: String,
12:      required: [true, "Feature name is required"],
13:      unique: true,
14:      trim: true,
15:      index: true,
16:    },
17:    description: {
18:      type: String,
19:      trim: true,
20:    },
21:    enabled: {
22:      type: Boolean,
23:      default: true,
24:    },
25:    // Which roles can access this feature
26:    allowedRoles: [
27:      {
28:        type: String,
29:        enum: Object.values(ROLES),
30:      },
31:    ],
32:    // Environment-specific settings
33:    environments: {
34:      development: {
35:        enabled: { type: Boolean, default: true },
36:      },
37:      staging: {
38:        enabled: { type: Boolean, default: true },
39:      },
40:      production: {
41:        enabled: { type: Boolean, default: false },
42:      },
43:    },
44:    // Percentage rollout (0-100)
45:    rolloutPercentage: {
46:      type: Number,
47:      min: 0,
48:      max: 100,
49:      default: 100,
50:    },
51:    // Feature dependencies
52:    dependsOn: [
53:      {
54:        type: String, // Other feature names
55:      },
56:    ],
57:    // Metadata
58:    metadata: {
59:      type: Map,
60:      of: mongoose.Schema.Types.Mixed,
61:    },
62:    createdBy: {
63:      type: mongoose.Schema.Types.ObjectId,
64:      ref: "User",
65:      required: true,
66:    },
67:    updatedBy: {
68:      type: mongoose.Schema.Types.ObjectId,
69:      ref: "User",
70:    },
71:  },
72:  {
73:    timestamps: true,
74:    toJSON: { virtuals: true },
75:    toObject: { virtuals: true },
76:  },

```

```

77: });
78:
79: /**
80:  * Indexes for performance
81:  */
82: featureToggleSchema.index({ featureName: 1, enabled: 1 });
83: featureToggleSchema.index({ createdAt: -1 });
84:
85: /**
86:  * Check if feature is enabled for a specific role and environment
87:  */
88: featureToggleSchema.methods.isEnabledFor = function (
89:   role,
90:   environment = "development",
91: ) {
92:   // Check if feature is globally enabled
93:   if (!this.enabled) {
94:     return false;
95:   }
96:
97:   // Check environment-specific setting
98:   if (
99:     this.environments[environment] &&
100:     !this.environments[environment].enabled
101:   ) {
102:     return false;
103:   }
104:
105:   // Check role access
106:   if (this.allowedRoles.length === 0) {
107:     return true; // No role restriction = available to all
108:   }
109:
110:   return this.allowedRoles.includes(role);
111: };
112:
113: /**
114:  * Check if feature should be rolled out to user (based on percentage)
115:  */
116: featureToggleSchema.methods.shouldRollout = function (userId) {
117:   if (this.rolloutPercentage === 100) {
118:     return true;
119:   }
120:
121:   if (this.rolloutPercentage === 0) {
122:     return false;
123:   }
124:
125:   // Use consistent hash of userId to determine rollout
126:   const hash = userId
127:     .toString()
128:     .split("")
129:     .reduce((acc, char) => {
130:       return (acc << 5) - acc + char.charCodeAt(0);
131:     }, 0);
132:
133:   const userPercentile = Math.abs(hash % 100);
134:   return userPercentile < this.rolloutPercentage;
135: };
136:
137: /**
138:  * Static method to check if feature is enabled
139:  */
140: featureToggleSchema.statics.checkFeature = async function (
141:   featureName,
142:   role,
143:   environment,
144:   userId,
145: ) {
146:   const feature = await this.findOne({ featureName, enabled: true });
147:
148:   if (!feature) {
149:     return false;

```

```

150:   }
151:
152:   // Check role and environment
153:   if (!feature.isEnabledFor(role, environment)) {
154:     return false;
155:   }
156:
157:   // Check rollout percentage
158:   if (userId && !feature.shouldRollout(userId)) {
159:     return false;
160:   }
161:
162:   return true;
163: };
164:
165: /**
166:  * Get all enabled features for a role
167:  */
168: featureToggleSchema.statics.getEnabledFeatures = async function (
169:   role,
170:   environment = "development",
171: ) {
172:   const features = await this.find({ enabled: true });
173:
174:   return features.filter((feature) => feature.isEnabledFor(role, environment));
175: };
176:
177: /**
178:  * Pre-save validation
179:  */
180: featureToggleSchema.pre("save", function (next) {
181:   // Ensure at least one environment is enabled if feature is enabled
182:   if (this.enabled) {
183:     const hasEnabledEnv = Object.values(this.environments).some(
184:       (env) => env.enabled,
185:     );
186:     if (!hasEnabledEnv && this.isNew) {
187:       this.environments.development.enabled = true;
188:     }
189:   }
190:   next();
191: });
192:
193: const FeatureToggle = mongoose.model("FeatureToggle", featureToggleSchema);
194:
195: module.exports = FeatureToggle;

```

■ File: src\models\RateGuard.js

```

1: const mongoose = require("mongoose");
2: const { ROLES } = require("../User");
3:
4: /**
5:  * Rate Guard Schema
6:  * Defines rate limiting rules for API endpoints
7:  */
8: const rateGuardSchema = new mongoose.Schema(
9:   {
10:    routePath: {
11:      type: String,
12:      required: [true, "Route path is required"],
13:      trim: true,
14:      index: true,
15:    },
16:    method: {
17:      type: String,
18:      enum: ["GET", "POST", "PUT", "PATCH", "DELETE", "ALL"],
19:      default: "ALL",
20:      uppercase: true,
21:    },
22:    description: {

```

```

23:     type: String,
24:     trim: true,
25:   },
26:   enabled: {
27:     type: Boolean,
28:     default: true,
29:   },
30:   // Role-specific rate limits
31:   limits: {
32:     admin: {
33:       maxRequests: { type: Number, default: 1000 },
34:       windowMs: { type: Number, default: 60000 }, // 1 minute
35:     },
36:     user: {
37:       maxRequests: { type: Number, default: 100 },
38:       windowMs: { type: Number, default: 60000 },
39:     },
40:     guest: {
41:       maxRequests: { type: Number, default: 10 },
42:       windowMs: { type: Number, default: 60000 },
43:     },
44:   },
45:   // Global limit (applies to all roles)
46:   globalLimit: {
47:     maxRequests: { type: Number },
48:     windowMs: { type: Number },
49:   },
50:   // IP-based limiting
51:   ipBased: {
52:     type: Boolean,
53:     default: false,
54:   },
55:   // Block on limit exceeded
56:   blockDuration: {
57:     type: Number, // milliseconds
58:     default: 0, // 0 = no blocking, just rate limit
59:   },
60:   // Custom error message
61:   errorMessage: {
62:     type: String,
63:     default: "Rate limit exceeded. Please try again later.",
64:   },
65:   // Whitelist (IPs or user IDs that bypass rate limiting)
66:   whitelist: [
67:     {
68:       type: String,
69:     },
70:   ],
71:   // Metadata
72:   metadata: {
73:     type: Map,
74:     of: mongoose.Schema.Types.Mixed,
75:   },
76:   createdBy: {
77:     type: mongoose.Schema.Types.ObjectId,
78:     ref: "User",
79:     required: true,
80:   },
81:   updatedBy: {
82:     type: mongoose.Schema.Types.ObjectId,
83:     ref: "User",
84:   },
85: },
86: {
87:   timestamps: true,
88:   toJSON: { virtuals: true },
89:   toObject: { virtuals: true },
90: },
91: );
92:
93: /**
94:  * Compound index for route and method lookup
95:  */

```

```

96: rateGuardSchema.index({ routePath: 1, method: 1 });
97: rateGuardSchema.index({ enabled: 1 });
98:
99: /**
100:  * Get rate limit for specific role
101:  */
102: rateGuardSchema.methods.getLimitForRole = function (role) {
103:   if (!this.enabled) {
104:     return null;
105:   }
106:
107:   // Check if global limit exists and should be applied
108:   if (this.globalLimit && this.globalLimit.maxRequests) {
109:     return {
110:       maxRequests: this.globalLimit.maxRequests,
111:       windowMs: this.globalLimit.windowMs,
112:     };
113:   }
114:
115:   // Return role-specific limit
116:   const roleLimit = this.limits[role];
117:   if (!roleLimit) {
118:     return this.limits.user; // Default to user limits
119:   }
120:
121:   return roleLimit;
122: };
123:
124: /**
125:  * Check if identifier is whitelisted
126:  */
127: rateGuardSchema.methods.isWhitelisted = function (identifier) {
128:   return this.whitelist.includes(identifier);
129: };
130:
131: /**
132:  * Static method to find rate guard rule for route
133:  */
134: rateGuardSchema.statics.findRuleForRoute = async function (
135:   routePath,
136:   method = "ALL",
137: ) {
138:   // Try exact match first
139:   let rule = await this.findOne({
140:     routePath,
141:     method: method.toUpperCase(),
142:     enabled: true,
143:   });
144:
145:   // Try with ALL method if specific method not found
146:   if (!rule && method !== "ALL") {
147:     rule = await this.findOne({
148:       routePath,
149:       method: "ALL",
150:       enabled: true,
151:     });
152:   }
153:
154:   // Try pattern matching for wildcard routes
155:   if (!rule) {
156:     const rules = await this.find({ enabled: true });
157:     rule = rules.find((r) => {
158:       const pattern = r.routePath
159:         .replace(/\/*/g, ".*")
160:         .replace(/:\w+/g, "[^/]+");
161:       const regex = new RegExp(`^${pattern}$`);
162:       return (
163:         regex.test(routePath) &&
164:         (r.method === "ALL" || r.method === method.toUpperCase())
165:       );
166:     });
167:   }
168:

```

```

169:   return rule;
170: };
171:
172: /**
173:  * Get all active rate guard rules
174:  */
175: rateGuardSchema.statics.getActiveRules = async function () {
176:   return this.find({ enabled: true }).sort({ routePath: 1 });
177: };
178:
179: /**
180:  * Virtual for display name
181:  */
182: rateGuardSchema.virtual("displayName").get(function () {
183:   return `${this.method} ${this.routePath}`;
184: });
185:
186: /**
187:  * Pre-save validation
188:  */
189: rateGuardSchema.pre("save", function (next) {
190:   // Ensure at least one limit is defined
191:   const hasLimit =
192:     this.globalLimit?.maxRequests ||
193:     Object.values(this.limits).some((limit) => limit.maxRequests > 0);
194:
195:   if (!hasLimit) {
196:     next(new Error("At least one rate limit must be defined"));
197:   }
198:
199:   next();
200: });
201:
202: /**
203:  * Format for rate limiter middleware
204:  */
205: rateGuardSchema.methods.toRateLimiterConfig = function (role) {
206:   const limit = this.getLimitForRole(role);
207:
208:   if (!limit) {
209:     return null;
210:   }
211:
212:   return {
213:     windowMs: limit.windowMs,
214:     max: limit.maxRequests,
215:     message: this.errorMessage,
216:     standardHeaders: true,
217:     legacyHeaders: false,
218:     skip: (req) =>
219:       this.isWhitelisted(req.ip) || this.isWhitelisted(req.user?.id),
220:     handler: (req, res) => {
221:       res.status(429).json({
222:         success: false,
223:         message: this.errorMessage,
224:         retryAfter: Math.ceil(limit.windowMs / 1000),
225:       });
226:     },
227:   };
228: };
229:
230: const RateGuard = mongoose.model("RateGuard", rateGuardSchema);
231:
232: module.exports = RateGuard;

```

■ File: src\models\User.js

```

=====
1: const mongoose = require("mongoose");
2: const bcrypt = require("bcryptjs");
3:
4: /**

```

```

5:  * User Roles Enum
6:  */
7:  const ROLES = {
8:    ADMIN: "admin",
9:    USER: "user",
10:   GUEST: "guest",
11: };
12:
13: /**
14:  * User Schema
15:  */
16: const userSchema = new mongoose.Schema(
17:   {
18:     email: {
19:       type: String,
20:       required: [true, "Email is required"],
21:       unique: true,
22:       lowercase: true,
23:       trim: true,
24:       match: [/^\S+@\S+\.\S+$/, "Please provide a valid email"],
25:     },
26:     password: {
27:       type: String,
28:       required: [true, "Password is required"],
29:       minlength: [6, "Password must be at least 6 characters"],
30:       select: false, // Don't return password by default
31:     },
32:     role: {
33:       type: String,
34:       enum: Object.values(ROLES),
35:       default: ROLES.USER,
36:     },
37:     firstName: {
38:       type: String,
39:       trim: true,
40:     },
41:     lastName: {
42:       type: String,
43:       trim: true,
44:     },
45:     isActive: {
46:       type: Boolean,
47:       default: true,
48:     },
49:     lastLogin: {
50:       type: Date,
51:     },
52:     loginAttempts: {
53:       type: Number,
54:       default: 0,
55:     },
56:     lockUntil: {
57:       type: Date,
58:     },
59:   },
60:   {
61:     timestamps: true,
62:     toJSON: {
63:       transform: (doc, ret) => {
64:         delete ret.password;
65:         delete ret.__v;
66:         return ret;
67:       },
68:     },
69:   },
70: );
71:
72: /**
73:  * Pre-save hook to hash password
74:  */
75: userSchema.pre("save", async function (next) {
76:   // Only hash if password is modified
77:   if (!this.isModified("password")) {

```



```

78:     return next();
79: }
80:
81: try {
82:     const salt = await bcrypt.genSalt(10);
83:     this.password = await bcrypt.hash(this.password, salt);
84:     next();
85: } catch (error) {
86:     next(error);
87: }
88: });
89:
90: /**
91:  * Compare password method
92:  */
93: userSchema.methods.comparePassword = async function (candidatePassword) {
94:     try {
95:         return await bcrypt.compare(candidatePassword, this.password);
96:     } catch (error) {
97:         throw new Error("Password comparison failed");
98:     }
99: };
100:
101: /**
102:  * Check if account is locked
103:  */
104: userSchema.methods.isLocked = function () {
105:     return !(this.lockUntil && this.lockUntil > Date.now());
106: };
107:
108: /**
109:  * Increment login attempts
110:  */
111: userSchema.methods.incLoginAttempts = async function () {
112:     // Reset attempts if lock has expired
113:     if (this.lockUntil && this.lockUntil < Date.now()) {
114:         return this.updateOne({
115:             $set: { loginAttempts: 1 },
116:             $unset: { lockUntil: 1 },
117:         });
118:     }
119:
120:     const updates = { $inc: { loginAttempts: 1 } };
121:
122:     // Lock account after 5 failed attempts for 2 hours
123:     if (this.loginAttempts + 1 >= 5 && !this.isLocked()) {
124:         updates.$set = { lockUntil: Date.now() + 2 * 60 * 60 * 1000 };
125:     }
126:
127:     return this.updateOne(updates);
128: };
129:
130: /**
131:  * Reset login attempts
132:  */
133: userSchema.methods.resetLoginAttempts = async function () {
134:     return this.updateOne({
135:         $set: { loginAttempts: 0, lastLogin: new Date() },
136:         $unset: { lockUntil: 1 },
137:     });
138: };
139:
140: /**
141:  * Get full name
142:  */
143: userSchema.virtual("fullName").get(function () {
144:     return `${this.firstName || ""} ${this.lastName || ""}`.trim() || "Anonymous";
145: });
146:
147: /**
148:  * Static method to find by email
149:  */
150: userSchema.statics.findByEmail = function (email) {

```

```

151:   return this.findOne({ email: email.toLowerCase() }).select("+password");
152: };
153:
154: /**
155:  * Check if user has role
156:  */
157: userSchema.methods.hasRole = function (role) {
158:   return this.role === role;
159: };
160:
161: /**
162:  * Check if user is admin
163:  */
164: userSchema.methods.isAdmin = function () {
165:   return this.role === ROLES.ADMIN;
166: };
167:
168: const User = mongoose.model("User", userSchema);
169:
170: module.exports = {
171:   User,
172:   ROLES,
173: };

```

■ File: src\models\index.js

```

1: /**
2:  * Models Index
3:  * Centralized export of all database models
4:  */
5:
6: const { User, ROLES } = require('./User');
7: const FeatureToggle = require('./FeatureToggle');
8: const RateGuard = require('./RateGuard');
9: const { AuditLog, AUDIT_ACTIONS, RESOURCE_TYPES } = require('./AuditLog');
10:
11: module.exports = {
12:   User,
13:   FeatureToggle,
14:   RateGuard,
15:   AuditLog,
16:   ROLES,
17:   AUDIT_ACTIONS,
18:   RESOURCE_TYPES
19: };

```

■ File: src\routes\audit.routes.js

```

1: const express = require("express");
2: const router = express.Router();
3: const { auditService } = require("../services");
4: const { authenticate, adminOnly, catchAsync } = require("../middleware");
5:
6: /**
7:  * All audit routes require authentication and admin role
8:  */
9: router.use(authenticate);
10: router.use(adminOnly);
11:
12: /**
13:  * @route   GET /api/audit
14:  * @desc    Get all audit logs with filtering
15:  * @access  Private/Admin
16:  */
17: router.get(
18:   "/",
19:   catchAsync(async (req, res) => {
20:     const {
21:       action,

```

```

22:     resourceType,
23:     userId,
24:     success,
25:     startDate,
26:     endDate,
27:     page,
28:     limit,
29:     sort,
30:   } = req.query;
31:
32:   const filters = {
33:     ...(action && { action }),
34:     ...(resourceType && { resourceType }),
35:     ...(userId && { userId }),
36:     ...(success !== undefined && { success: success === "true" }),
37:     ...(startDate && endDate && { startDate, endDate }),
38:   };
39:
40:   const pagination = {
41:     page: parseInt(page) || 1,
42:     limit: parseInt(limit) || 50,
43:     sort: sort || "-createdAt",
44:   };
45:
46:   const result = await auditService.getAllLogs(filters, pagination);
47:
48:   res.json({
49:     success: true,
50:     data: result,
51:   });
52: },
53: );
54:
55: /**
56:  * @route   GET /api/audit/user/:userId
57:  * @desc    Get audit logs for a specific user
58:  * @access  Private/Admin
59:  */
60: router.get(
61:   "/user/:userId",
62:   catchAsync(async (req, res) => {
63:     const { limit } = req.query;
64:     const logs = await auditService.getUserLogs(
65:       req.params.userId,
66:       parseInt(limit) || 50,
67:     );
68:
69:     res.json({
70:       success: true,
71:       data: { logs, count: logs.length },
72:     });
73:   }),
74: );
75:
76: /**
77:  * @route   GET /api/audit/resource/:resourceType/:resourceId
78:  * @desc    Get audit logs for a specific resource
79:  * @access  Private/Admin
80:  */
81: router.get(
82:   "/resource/:resourceType/:resourceId",
83:   catchAsync(async (req, res) => {
84:     const { resourceType, resourceId } = req.params;
85:     const { limit } = req.query;
86:
87:     const logs = await auditService.getResourceLogs(
88:       resourceType,
89:       resourceId,
90:       parseInt(limit) || 50,
91:     );
92:
93:     res.json({
94:       success: true,

```

```

95:         data: { logs, count: logs.length },
96:     });
97: }},
98: );
99:
100: /**
101:  * @route   GET /api/audit/failed
102:  * @desc    Get failed actions
103:  * @access  Private/Admin
104:  */
105: router.get(
106:     "/failed",
107:     catchAsync(async (req, res) => {
108:         const { hours, limit } = req.query;
109:
110:         const logs = await auditService.getFailedActions(
111:             parseInt(hours) || 24,
112:             parseInt(limit) || 100,
113:         );
114:
115:         res.json({
116:             success: true,
117:             data: { logs, count: logs.length },
118:         });
119:     }),
120: );
121:
122: /**
123:  * @route   GET /api/audit/security
124:  * @desc    Get security events
125:  * @access  Private/Admin
126:  */
127: router.get(
128:     "/security",
129:     catchAsync(async (req, res) => {
130:         const { hours } = req.query;
131:
132:         const events = await auditService.getSecurityEvents(parseInt(hours) || 24);
133:
134:         res.json({
135:             success: true,
136:             data: { events, count: events.length },
137:         });
138:     }),
139: );
140:
141: /**
142:  * @route   GET /api/audit/stats
143:  * @desc    Get audit statistics
144:  * @access  Private/Admin
145:  */
146: router.get(
147:     "/stats",
148:     catchAsync(async (req, res) => {
149:         const { startDate, endDate } = req.query;
150:
151:         const stats = await auditService.getStats(startDate, endDate);
152:
153:         res.json({
154:             success: true,
155:             data: stats,
156:         });
157:     }),
158: );
159:
160: /**
161:  * @route   GET /api/audit/export
162:  * @desc    Export audit logs
163:  * @access  Private/Admin
164:  */
165: router.get(
166:     "/export",
167:     catchAsync(async (req, res) => {

```

```

168:     const { startDate, endDate, format } = req.query;
169:
170:     const filters = {
171:       ...(startDate && endDate && { startDate, endDate }),
172:     };
173:
174:     const logs = await auditService.exportLogs(filters, format || "json");
175:
176:     if (format === "csv") {
177:       res.setHeader("Content-Type", "text/csv");
178:       res.setHeader(
179:         "Content-Disposition",
180:         "attachment; filename=audit-logs.csv",
181:       );
182:       res.send(logs);
183:     } else {
184:       res.json({
185:         success: true,
186:         data: { logs, count: logs.length },
187:       });
188:     }
189:   }},
190: );
191:
192: module.exports = router;

```

File: src/routes/auth.routes.js

```

=====
1: const express = require("express");
2: const router = express.Router();
3: const { authService } = require("../services");
4: const {
5:   authenticate,
6:   validate,
7:   schemas,
8:   catchAsync,
9: } = require("../middleware");
10:
11: /**
12:  * @route   POST /api/auth/register
13:  * @desc    Register a new user
14:  * @access  Public
15:  */
16: router.post(
17:   "/register",
18:   validate(schemas.register),
19:   catchAsync(async (req, res) => {
20:     const { email, password, firstName, lastName } = req.body;
21:
22:     const result = await authService.register(
23:       { email, password, firstName, lastName },
24:       {
25:         ip: req.ip,
26:         userAgent: req.get("user-agent"),
27:       },
28:     );
29:
30:     res.status(201).json({
31:       success: true,
32:       message: "User registered successfully",
33:       data: result,
34:     });
35:   }),
36: );
37:
38: /**
39:  * @route   POST /api/auth/login
40:  * @desc    Login user
41:  * @access  Public
42:  */
43: router.post(

```

```

44:   "/login",
45:   validate(schemas.login),
46:   catchAsync(async (req, res) => {
47:     const { email, password } = req.body;
48:
49:     const result = await authService.login(email, password, {
50:       ip: req.ip,
51:       userAgent: req.get("user-agent"),
52:     });
53:
54:     res.json({
55:       success: true,
56:       message: "Login successful",
57:       data: result,
58:     });
59:   }),
60: );
61:
62: /**
63:  * @route   POST /api/auth/logout
64:  * @desc    Logout user
65:  * @access  Private
66:  */
67: router.post(
68:   "/logout",
69:   authenticate,
70:   catchAsync(async (req, res) => {
71:     await authService.logout(req.user._id, {
72:       ip: req.ip,
73:       userAgent: req.get("user-agent"),
74:     });
75:
76:     res.json({
77:       success: true,
78:       message: "Logged out successfully",
79:     });
80:   }),
81: );
82:
83: /**
84:  * @route   POST /api/auth/refresh
85:  * @desc    Refresh access token
86:  * @access  Public
87:  */
88: router.post(
89:   "/refresh",
90:   catchAsync(async (req, res) => {
91:     const { refreshToken } = req.body;
92:
93:     if (!refreshToken) {
94:       return res.status(400).json({
95:         success: false,
96:         message: "Refresh token is required",
97:       });
98:     }
99:
100:    const result = await authService.refreshToken(refreshToken);
101:
102:    res.json({
103:      success: true,
104:      message: "Token refreshed successfully",
105:      data: result,
106:    });
107:  }),
108: );
109:
110: /**
111:  * @route   GET /api/auth/me
112:  * @desc    Get current user profile
113:  * @access  Private
114:  */
115: router.get(
116:   "/me",

```

```

117:   authenticate,
118:   catchAsync(async (req, res) => {
119:     const user = await authService.getCurrentUser(req.user._id);
120:
121:     res.json({
122:       success: true,
123:       data: { user },
124:     });
125:   }),
126: );
127:
128: /**
129:  * @route   PUT /api/auth/profile
130:  * @desc    Update user profile
131:  * @access  Private
132:  */
133: router.put(
134:   "/profile",
135:   authenticate,
136:   catchAsync(async (req, res) => {
137:     const { firstName, lastName } = req.body;
138:
139:     const user = await authService.updateProfile(req.user._id, {
140:       firstName,
141:       lastName,
142:     });
143:
144:     res.json({
145:       success: true,
146:       message: "Profile updated successfully",
147:       data: { user },
148:     });
149:   }),
150: );
151:
152: /**
153:  * @route   PUT /api/auth/change-password
154:  * @desc    Change user password
155:  * @access  Private
156:  */
157: router.put(
158:   "/change-password",
159:   authenticate,
160:   catchAsync(async (req, res) => {
161:     const { currentPassword, newPassword } = req.body;
162:
163:     if (!currentPassword || !newPassword) {
164:       return res.status(400).json({
165:         success: false,
166:         message: "Current password and new password are required",
167:       });
168:     }
169:
170:     if (newPassword.length < 6) {
171:       return res.status(400).json({
172:         success: false,
173:         message: "New password must be at least 6 characters",
174:       });
175:     }
176:
177:     const result = await authService.changePassword(
178:       req.user._id,
179:       currentPassword,
180:       newPassword,
181:     );
182:
183:     res.json({
184:       success: true,
185:       message: result.message,
186:     });
187:   }),
188: );
189:

```

```
190: module.exports = router;
```

■ File: src/routes/features.routes.js

```
=====
1: const express = require("express");
2: const router = express.Router();
3: const { featureToggleService } = require("../services");
4: const {
5:   authenticate,
6:   adminOnly,
7:   validate,
8:   schemas,
9:   catchAsync,
10: } = require("../middleware");
11:
12: /**
13:  * All feature toggle routes require authentication
14:  */
15: router.use(authenticate);
16:
17: /**
18:  * @route   GET /api/features
19:  * @desc    Get all feature toggles
20:  * @access  Private
21:  */
22: router.get(
23:   "/",
24:   catchAsync(async (req, res) => {
25:     const { enabled, search } = req.query;
26:
27:     const filters = {
28:       ...(enabled !== undefined && { enabled: enabled === "true" }),
29:       ...(search && { search }),
30:     };
31:
32:     const features = await featureToggleService.getAllFeatures(filters);
33:
34:     res.json({
35:       success: true,
36:       data: { features, count: features.length },
37:     });
38:   }),
39: );
40:
41: /**
42:  * @route   GET /api/features/stats
43:  * @desc    Get feature toggle statistics (admin only)
44:  * @access  Private/Admin
45:  */
46: router.get(
47:   "/stats",
48:   adminOnly,
49:   catchAsync(async (req, res) => {
50:     const stats = await featureToggleService.getFeatureStats();
51:
52:     res.json({
53:       success: true,
54:       data: stats,
55:     });
56:   }),
57: );
58:
59: /**
60:  * @route   GET /api/features/enabled
61:  * @desc    Get enabled features for current user
62:  * @access  Private
63:  */
64: router.get(
65:   "/enabled",
66:   catchAsync(async (req, res) => {
67:     const features = await featureToggleService.getEnabledFeaturesForRole(

```



```

68:     req.user.role,
69:   );
70:
71:   res.json({
72:     success: true,
73:     data: { features, count: features.length },
74:   });
75: },
76: );
77:
78: /**
79:  * @route   GET /api/features/:id
80:  * @desc    Get feature toggle by ID
81:  * @access  Private
82:  */
83: router.get(
84:   "/:id",
85:   validate(schemas.idParam),
86:   catchAsync(async (req, res) => {
87:     const feature = await featureToggleService.getFeatureById(req.params.id);
88:
89:     res.json({
90:       success: true,
91:       data: { feature },
92:     });
93:   }),
94: );
95:
96: /**
97:  * @route   GET /api/features/name/:featureName
98:  * @desc    Get feature toggle by name
99:  * @access  Private
100:  */
101: router.get(
102:   "/name/:featureName",
103:   catchAsync(async (req, res) => {
104:     const feature = await featureToggleService.getFeatureByName(
105:       req.params.featureName,
106:     );
107:
108:     res.json({
109:       success: true,
110:       data: { feature },
111:     });
112:   }),
113: );
114:
115: /**
116:  * @route   POST /api/features/check
117:  * @desc    Check if a feature is enabled for current user
118:  * @access  Private
119:  */
120: router.post(
121:   "/check",
122:   catchAsync(async (req, res) => {
123:     const { featureName } = req.body;
124:
125:     if (!featureName) {
126:       return res.status(400).json({
127:         success: false,
128:         message: "Feature name is required",
129:       });
130:     }
131:
132:     const result = await featureToggleService.checkFeatureAccess(
133:       featureName,
134:       req.user.role,
135:       req.user._id,
136:     );
137:
138:     res.json({
139:       success: true,
140:       data: result,

```

```

141:     });
142:   }),
143: );
144:
145: /**
146:  * @route   POST /api/features
147:  * @desc    Create new feature toggle (admin only)
148:  * @access  Private/Admin
149:  */
150: router.post(
151:   "/",
152:   adminOnly,
153:   validate(schemas.createFeatureToggle),
154:   catchAsync(async (req, res) => {
155:     const feature = await featureToggleService.createFeature(
156:       req.body,
157:       req.user._id,
158:     );
159:
160:     res.status(201).json({
161:       success: true,
162:       message: "Feature toggle created successfully",
163:       data: { feature },
164:     });
165:   }),
166: );
167:
168: /**
169:  * @route   PUT /api/features/:id
170:  * @desc    Update feature toggle (admin only)
171:  * @access  Private/Admin
172:  */
173: router.put(
174:   "/*:id",
175:   adminOnly,
176:   validate(schemas.idParam),
177:   catchAsync(async (req, res) => {
178:     const feature = await featureToggleService.updateFeature(
179:       req.params.id,
180:       req.body,
181:       req.user._id,
182:     );
183:
184:     res.json({
185:       success: true,
186:       message: "Feature toggle updated successfully",
187:       data: { feature },
188:     });
189:   }),
190: );
191:
192: /**
193:  * @route   PUT /api/features/:id/toggle
194:  * @desc    Toggle feature enable/disable (admin only)
195:  * @access  Private/Admin
196:  */
197: router.put(
198:   "/*:id/toggle",
199:   adminOnly,
200:   validate(schemas.idParam),
201:   catchAsync(async (req, res) => {
202:     const { enabled } = req.body;
203:
204:     if (enabled === undefined) {
205:       return res.status(400).json({
206:         success: false,
207:         message: "enabled field is required (true or false)",
208:       });
209:     }
210:
211:     const feature = await featureToggleService.toggleFeature(
212:       req.params.id,
213:       enabled,

```

```

214:         req.user._id,
215:     );
216:
217:     res.json({
218:         success: true,
219:         message: `Feature ${enabled ? "enabled" : "disabled"} successfully`,
220:         data: { feature },
221:     });
222: });
223: );
224:
225: /**
226:  * @route   DELETE /api/features/:id
227:  * @desc    Delete feature toggle (admin only)
228:  * @access  Private/Admin
229:  */
230: router.delete(
231:     "/:id",
232:     adminOnly,
233:     validate(schemas.idParam),
234:     catchAsync(async (req, res) => {
235:         const result = await featureToggleService.deleteFeature(
236:             req.params.id,
237:             req.user._id,
238:         );
239:
240:         res.json({
241:             success: true,
242:             message: result.message,
243:             data: { featureName: result.featureName },
244:         });
245:     }),
246: );
247:
248: /**
249:  * @route   POST /api/features/bulk-update
250:  * @desc    Bulk update features (admin only)
251:  * @access  Private/Admin
252:  */
253: router.post(
254:     "/bulk-update",
255:     adminOnly,
256:     catchAsync(async (req, res) => {
257:         const { updates } = req.body;
258:
259:         if (!Array.isArray(updates) || updates.length === 0) {
260:             return res.status(400).json({
261:                 success: false,
262:                 message: "updates array is required",
263:             });
264:         }
265:
266:         const results = await featureToggleService.bulkUpdateFeatures(
267:             updates,
268:             req.user._id,
269:         );
270:
271:         res.json({
272:             success: true,
273:             message: "Bulk update completed",
274:             data: { results },
275:         });
276:     }),
277: );
278:
279: module.exports = router;

```

File: src/routes/index.js

```

=====
1: /**
2:  * Routes Index

```

```

3:  * Centralized routing configuration
4:  */
5:
6:  const express = require("express");
7:  const router = express.Router();
8:
9:  // Import route modules
10: const authRoutes = require("./auth.routes");
11: const userRoutes = require("./users.routes");
12: const featureRoutes = require("./features.routes");
13: const rateGuardRoutes = require("./rateGuards.routes");
14: const auditRoutes = require("./audit.routes");
15: const systemRoutes = require("./system.routes");
16:
17: // Mount routes
18: router.use("/auth", authRoutes);
19: router.use("/users", userRoutes);
20: router.use("/features", featureRoutes);
21: router.use("/rate-guards", rateGuardRoutes);
22: router.use("/audit", auditRoutes);
23:
24: // System routes are mounted at root level
25: router.use("/", systemRoutes);
26:
27: // API documentation endpoint
28: router.get("/", (req, res) => {
29:   res.json({
30:     success: true,
31:     message: "Policy-Driven Feature Toggle & Rate Guard Service API",
32:     version: "1.0.0",
33:     endpoints: {
34:       auth: "/api/auth",
35:       users: "/api/users",
36:       features: "/api/features",
37:       rateGuards: "/api/rate-guards",
38:       audit: "/api/audit",
39:       health: "/api/health",
40:       status: "/api/status",
41:     },
42:     documentation: "See README.md for detailed API documentation",
43:   });
44: });
45:
46: module.exports = router;

```

File: src\routes\rateGuards.routes.js

```

1: const express = require("express");
2: const router = express.Router();
3: const { rateGuardService } = require("../services");
4: const {
5:   authenticate,
6:   adminOnly,
7:   validate,
8:   schemas,
9:   catchAsync,
10: } = require("../middleware");
11:
12: /**
13:  * All rate guard routes require authentication
14:  */
15: router.use(authenticate);
16:
17: /**
18:  * @route   GET /api/rate-guards
19:  * @desc    Get all rate guard rules
20:  * @access  Private
21:  */
22: router.get(
23:   "/",
24:   catchAsync(async (req, res) => {

```

```

25:     const { enabled, method, search } = req.query;
26:
27:     const filters = {
28:       ...(enabled !== undefined && { enabled: enabled === "true" }),
29:       ...(method && { method }),
30:       ...(search && { search }),
31:     };
32:
33:     const rules = await rateGuardService.getAllRules(filters);
34:
35:     res.json({
36:       success: true,
37:       data: { rules, count: rules.length },
38:     });
39:   },
40: );
41:
42: /**
43:  * @route   GET /api/rate-guards/active
44:  * @desc    Get active rate guard rules
45:  * @access  Private
46:  */
47: router.get(
48:   "/active",
49:   catchAsync(async (req, res) => {
50:     const rules = await rateGuardService.getActiveRules();
51:
52:     res.json({
53:       success: true,
54:       data: { rules, count: rules.length },
55:     });
56:   }),
57: );
58:
59: /**
60:  * @route   GET /api/rate-guards/stats
61:  * @desc    Get rate guard statistics (admin only)
62:  * @access  Private/Admin
63:  */
64: router.get(
65:   "/stats",
66:   adminOnly,
67:   catchAsync(async (req, res) => {
68:     const stats = await rateGuardService.getRateGuardStats();
69:
70:     res.json({
71:       success: true,
72:       data: stats,
73:     });
74:   }),
75: );
76:
77: /**
78:  * @route   GET /api/rate-guards/:id
79:  * @desc    Get rate guard rule by ID
80:  * @access  Private
81:  */
82: router.get(
83:   "/*:id",
84:   validate(schemas.idParam),
85:   catchAsync(async (req, res) => {
86:     const rule = await rateGuardService.getRuleById(req.params.id);
87:
88:     res.json({
89:       success: true,
90:       data: { rule },
91:     });
92:   }),
93: );
94:
95: /**
96:  * @route   POST /api/rate-guards/test
97:  * @desc    Test rate limit for a route

```

```

98: * @access Private
99: */
100: router.post(
101:   "/test",
102:   catchAsync(async (req, res) => {
103:     const { routePath, method } = req.body;
104:
105:     if (!routePath) {
106:       return res.status(400).json({
107:         success: false,
108:         message: "routePath is required",
109:       });
110:     }
111:
112:     const result = await rateGuardService.testRateLimit(
113:       routePath,
114:       method || "ALL",
115:       req.user.role,
116:     );
117:
118:     res.json({
119:       success: true,
120:       data: result,
121:     });
122:   }),
123: );
124:
125: /**
126:  * @route   POST /api/rate-guards
127:  * @desc    Create new rate guard rule (admin only)
128:  * @access  Private/Admin
129:  */
130: router.post(
131:   "/",
132:   adminOnly,
133:   validate(schemas.createRateGuard),
134:   catchAsync(async (req, res) => {
135:     const rule = await rateGuardService.createRule(req.body, req.user._id);
136:
137:     res.status(201).json({
138:       success: true,
139:       message: "Rate guard rule created successfully",
140:       data: { rule },
141:     });
142:   }),
143: );
144:
145: /**
146:  * @route   PUT /api/rate-guards/:id
147:  * @desc    Update rate guard rule (admin only)
148:  * @access  Private/Admin
149:  */
150: router.put(
151:   "/:id",
152:   adminOnly,
153:   validate(schemas.idParam),
154:   catchAsync(async (req, res) => {
155:     const rule = await rateGuardService.updateRule(
156:       req.params.id,
157:       req.body,
158:       req.user._id,
159:     );
160:
161:     res.json({
162:       success: true,
163:       message: "Rate guard rule updated successfully",
164:       data: { rule },
165:     });
166:   }),
167: );
168:
169: /**
170:  * @route   PUT /api/rate-guards/:id/toggle

```

```

171: * @desc    Toggle rate guard rule enable/disable (admin only)
172: * @access   Private/Admin
173: */
174: router.put(
175:   "/:id/toggle",
176:   adminOnly,
177:   validate(schemas.idParam),
178:   catchAsync(async (req, res) => {
179:     const { enabled } = req.body;
180:
181:     if (enabled === undefined) {
182:       return res.status(400).json({
183:         success: false,
184:         message: "enabled field is required (true or false)",
185:       });
186:     }
187:
188:     const rule = await rateGuardService.toggleRule(
189:       req.params.id,
190:       enabled,
191:       req.user._id,
192:     );
193:
194:     res.json({
195:       success: true,
196:       message: `Rate guard rule ${enabled ? "enabled" : "disabled"} successfully`,
197:       data: { rule },
198:     });
199:   }),
200: );
201:
202: /**
203: * @route    PUT /api/rate-guards/:id/whitelist/add
204: * @desc     Add identifier to whitelist (admin only)
205: * @access   Private/Admin
206: */
207: router.put(
208:   "/:id/whitelist/add",
209:   adminOnly,
210:   validate(schemas.idParam),
211:   catchAsync(async (req, res) => {
212:     const { identifier } = req.body;
213:
214:     if (!identifier) {
215:       return res.status(400).json({
216:         success: false,
217:         message: "identifier is required (user ID or IP address)",
218:       });
219:     }
220:
221:     const rule = await rateGuardService.addToWhitelist(
222:       req.params.id,
223:       identifier,
224:       req.user._id,
225:     );
226:
227:     res.json({
228:       success: true,
229:       message: "Identifier added to whitelist successfully",
230:       data: { rule },
231:     });
232:   }),
233: );
234:
235: /**
236: * @route    PUT /api/rate-guards/:id/whitelist/remove
237: * @desc     Remove identifier from whitelist (admin only)
238: * @access   Private/Admin
239: */
240: router.put(
241:   "/:id/whitelist/remove",
242:   adminOnly,
243:   validate(schemas.idParam),

```

```

244:   catchAsync(async (req, res) => {
245:     const { identifier } = req.body;
246:
247:     if (!identifier) {
248:       return res.status(400).json({
249:         success: false,
250:         message: "identifier is required (user ID or IP address)",
251:       });
252:     }
253:
254:     const rule = await rateGuardService.removeFromWhitelist(
255:       req.params.id,
256:       identifier,
257:       req.user._id,
258:     );
259:
260:     res.json({
261:       success: true,
262:       message: "Identifier removed from whitelist successfully",
263:       data: { rule },
264:     });
265:   }),
266: );
267:
268: /**
269:  * @route   DELETE /api/rate-guards/:id
270:  * @desc    Delete rate guard rule (admin only)
271:  * @access  Private/Admin
272:  */
273: router.delete(
274:   "/:id",
275:   adminOnly,
276:   validate(schemas.idParam),
277:   catchAsync(async (req, res) => {
278:     const result = await rateGuardService.deleteRule(
279:       req.params.id,
280:       req.user._id,
281:     );
282:
283:     res.json({
284:       success: true,
285:       message: result.message,
286:       data: { displayName: result.displayName },
287:     });
288:   }),
289: );
290:
291: module.exports = router;

```

File: src/routes/system.routes.js

```

=====
1: const express = require("express");
2: const router = express.Router();
3: const database = require("../config/database");
4: const config = require("../config");
5: const { authenticate, adminOnly, catchAsync } = require("../middleware");
6: const { User, FeatureToggle, RateGuard, AuditLog } = require("../models");
7:
8: /**
9:  * @route   GET /api/health
10:  * @desc    Basic health check
11:  * @access  Public
12:  */
13: router.get("/health", (req, res) => {
14:   res.json({
15:     success: true,
16:     status: "healthy",
17:     timestamp: new Date().toISOString(),
18:     uptime: process.uptime(),
19:   });
20: });

```



```

21:
22: /**
23:  * @route   GET /api/status
24:  * @desc    Detailed system status
25:  * @access   Public
26:  */
27: router.get(
28:   "/status",
29:   catchAsync(async (req, res) => {
30:     const dbStatus = database.isConnected() ? "connected" : "disconnected";
31:
32:     res.json({
33:       success: true,
34:       data: {
35:         service: "Policy Toggle Service",
36:         version: "1.0.0",
37:         environment: config.env,
38:         status: "operational",
39:         database: dbStatus,
40:         timestamp: new Date().toISOString(),
41:         uptime: process.uptime(),
42:       },
43:     });
44:   }),
45: );
46:
47: /**
48:  * @route   GET /api/system/info
49:  * @desc    System information (admin only)
50:  * @access   Private/Admin
51:  */
52: router.get(
53:   "/system/info",
54:   authenticate,
55:   adminOnly,
56:   catchAsync(async (req, res) => {
57:     const [userCount, featureCount, rateGuardCount, auditCount] =
58:       await Promise.all([
59:         User.countDocuments(),
60:         FeatureToggle.countDocuments(),
61:         RateGuard.countDocuments(),
62:         AuditLog.countDocuments(),
63:       ]);
64:
65:     res.json({
66:       success: true,
67:       data: {
68:         system: {
69:           nodeVersion: process.version,
70:           platform: process.platform,
71:           arch: process.arch,
72:           uptime: process.uptime(),
73:           memory: {
74:             total:
75:               Math.round(process.memoryUsage().heapTotal / 1024 / 1024) + " MB",
76:             used:
77:               Math.round(process.memoryUsage().heapUsed / 1024 / 1024) + " MB",
78:           },
79:         },
80:         database: {
81:           status: database.isConnected() ? "connected" : "disconnected",
82:           collections: {
83:             users: userCount,
84:             features: featureCount,
85:             rateGuards: rateGuardCount,
86:             auditLogs: auditCount,
87:           },
88:         },
89:         environment: config.env,
90:         timestamp: new Date().toISOString(),
91:       },
92:     });
93:   }),

```

```

94: );
95:
96: /**
97:  * @route    GET /api/system/metrics
98:  * @desc     System metrics (admin only)
99:  * @access   Private/Admin
100:  */
101: router.get(
102:   "/system/metrics",
103:   authenticate,
104:   adminOnly,
105:   catchAsync(async (req, res) => {
106:     const now = new Date();
107:     const last24Hours = new Date(now - 24 * 60 * 60 * 1000);
108:
109:     const [recentAuditLogs, failedActions, activeUsers] = await Promise.all([
110:       AuditLog.countDocuments({ createdAt: { $gte: last24Hours } }),
111:       AuditLog.countDocuments({
112:         createdAt: { $gte: last24Hours },
113:         success: false,
114:       }),
115:       User.countDocuments({
116:         isActive: true,
117:         lastLogin: { $gte: last24Hours },
118:       }),
119:     ]);
120:
121:     res.json({
122:       success: true,
123:       data: {
124:         period: "24 hours",
125:         metrics: {
126:           totalAuditLogs: recentAuditLogs,
127:           failedActions: failedActions,
128:           successRate:
129:             recentAuditLogs > 0
130:             ? (
131:               ((recentAuditLogs - failedActions) / recentAuditLogs) *
132:               100
133:             ).toFixed(2) + "%"
134:             : "N/A",
135:           activeUsers: activeUsers,
136:         },
137:         timestamp: new Date().toISOString(),
138:       },
139:     });
140:   }),
141: );
142:
143: module.exports = router;

```

File: src/routes/users.routes.js

```

1: const express = require("express");
2: const router = express.Router();
3: const { userService } = require("../services");
4: const {
5:   authenticate,
6:   adminOnly,
7:   validate,
8:   schemas,
9:   catchAsync,
10: } = require("../middleware");
11:
12: /**
13:  * All user routes require authentication
14:  */
15: router.use(authenticate);
16:
17: /**
18:  * @route    GET /api/users

```

```

19: * @desc    Get all users (admin only)
20: * @access   Private/Admin
21: */
22: router.get(
23:   "/",
24:   adminOnly,
25:   catchAsync(async (req, res) => {
26:     const { role, isActive, page, limit, sort, search } = req.query;
27:
28:     const filters = {
29:       ...(role && { role }),
30:       ...(isActive !== undefined && { isActive: isActive === "true" }),
31:     };
32:
33:     const pagination = {
34:       page: parseInt(page) || 1,
35:       limit: parseInt(limit) || 10,
36:       sort: sort || "-createdAt",
37:       search: search || "",
38:     };
39:
40:     const result = await userService.getAllUsers(filters, pagination);
41:
42:     res.json({
43:       success: true,
44:       data: result,
45:     });
46:   }),
47: );
48:
49: /**
50: * @route    GET /api/users/stats
51: * @desc     Get user statistics (admin only)
52: * @access   Private/Admin
53: */
54: router.get(
55:   "/stats",
56:   adminOnly,
57:   catchAsync(async (req, res) => {
58:     const stats = await userService.getUserStats();
59:
60:     res.json({
61:       success: true,
62:       data: stats,
63:     });
64:   }),
65: );
66:
67: /**
68: * @route    GET /api/users/:id
69: * @desc     Get user by ID (admin only)
70: * @access   Private/Admin
71: */
72: router.get(
73:   "/*:id",
74:   adminOnly,
75:   validate(schemas.idParam),
76:   catchAsync(async (req, res) => {
77:     const user = await userService.getUserById(req.params.id);
78:
79:     res.json({
80:       success: true,
81:       data: { user },
82:     });
83:   }),
84: );
85:
86: /**
87: * @route    PUT /api/users/:id
88: * @desc     Update user (admin only)
89: * @access   Private/Admin
90: */
91: router.put(

```

```

92:   "/:id",
93:   adminOnly,
94:   validate(schemas.idParam),
95:   validate(schemas.updateUser),
96:   catchAsync(async (req, res) => {
97:     const user = await userService.updateUser(
98:       req.params.id,
99:       req.body,
100:      req.user._id,
101:    );
102:
103:    res.json({
104:      success: true,
105:      message: "User updated successfully",
106:      data: { user },
107:    });
108:  }),
109: );
110:
111: /**
112:  * @route   PUT /api/users/:id/role
113:  * @desc    Change user role (admin only)
114:  * @access  Private/Admin
115:  */
116: router.put(
117:   "/:id/role",
118:   adminOnly,
119:   validate(schemas.idParam),
120:   catchAsync(async (req, res) => {
121:     const { role } = req.body;
122:
123:     if (!role || ![ "admin", "user", "guest" ].includes(role)) {
124:       return res.status(400).json({
125:         success: false,
126:         message: "Valid role is required (admin, user, or guest)",
127:       });
128:     }
129:
130:     const user = await userService.changeUserRole(
131:       req.params.id,
132:       role,
133:       req.user._id,
134:     );
135:
136:     res.json({
137:       success: true,
138:       message: "User role changed successfully",
139:       data: { user },
140:     });
141:   }),
142: );
143:
144: /**
145:  * @route   PUT /api/users/:id/deactivate
146:  * @desc    Deactivate user (admin only)
147:  * @access  Private/Admin
148:  */
149: router.put(
150:   "/:id/deactivate",
151:   adminOnly,
152:   validate(schemas.idParam),
153:   catchAsync(async (req, res) => {
154:     const user = await userService.deactivateUser(req.params.id, req.user._id);
155:
156:     res.json({
157:       success: true,
158:       message: "User deactivated successfully",
159:       data: { user },
160:     });
161:   }),
162: );
163:
164: /**

```

```

165: * @route   PUT /api/users/:id/activate
166: * @desc    Activate user (admin only)
167: * @access   Private/Admin
168: */
169: router.put(
170:   "[:id/activate",
171:   adminOnly,
172:   validate(schemas.idParam),
173:   catchAsync(async (req, res) => {
174:     const user = await userService.activateUser(req.params.id, req.user._id);
175:
176:     res.json({
177:       success: true,
178:       message: "User activated successfully",
179:       data: { user },
180:     });
181:   }),
182: );
183:
184: /**
185: * @route   PUT /api/users/:id/unlock
186: * @desc    Unlock user account (admin only)
187: * @access   Private/Admin
188: */
189: router.put(
190:   "[:id/unlock",
191:   adminOnly,
192:   validate(schemas.idParam),
193:   catchAsync(async (req, res) => {
194:     const user = await userService.unlockUser(req.params.id, req.user._id);
195:
196:     res.json({
197:       success: true,
198:       message: "User account unlocked successfully",
199:       data: { user },
200:     });
201:   }),
202: );
203:
204: /**
205: * @route   DELETE /api/users/:id
206: * @desc    Delete user (admin only)
207: * @access   Private/Admin
208: */
209: router.delete(
210:   "[:id",
211:   adminOnly,
212:   validate(schemas.idParam),
213:   catchAsync(async (req, res) => {
214:     const result = await userService.deleteUser(req.params.id, req.user._id);
215:
216:     res.json({
217:       success: true,
218:       message: result.message,
219:       data: { email: result.email },
220:     });
221:   }),
222: );
223:
224: module.exports = router;

```

File: src\services\auditService.js

```

=====
1: const { AuditLog } = require("../models");
2: const { AppError } = require("../middleware/errorHandler");
3: const logger = require("../utils/logger");
4:
5: /**
6:  * Audit Service
7:  */
8: class AuditService {

```

```

9:  /**
10:   * Get all audit logs with filtering
11:   */
12:   async getAllLogs(filters = {}, pagination = {}) {
13:     try {
14:       const { page = 1, limit = 50, sort = "-createdAt" } = pagination;
15:
16:       const query = {};
17:
18:       // Apply filters
19:       if (filters.action) {
20:         query.action = filters.action;
21:       }
22:
23:       if (filters.resourceType) {
24:         query.resourceType = filters.resourceType;
25:       }
26:
27:       if (filters.userId) {
28:         query.userId = filters.userId;
29:       }
30:
31:       if (filters.success !== undefined) {
32:         query.success = filters.success;
33:       }
34:
35:       if (filters.startDate && filters.endDate) {
36:         query.createdAt = {
37:           $gte: new Date(filters.startDate),
38:           $lte: new Date(filters.endDate),
39:         };
40:       }
41:
42:       const skip = (page - 1) * limit;
43:
44:       const [logs, total] = await Promise.all([
45:         AuditLog.find(query)
46:           .sort(sort)
47:           .skip(skip)
48:           .limit(limit)
49:           .populate("userId", "email role")
50:           .lean(),
45:         AuditLog.countDocuments(query),
52:       ]);
53:
54:       return {
55:         logs,
56:         pagination: {
57:           page,
58:           limit,
59:           total,
60:           pages: Math.ceil(total / limit),
61:         },
62:       };
63:     } catch (error) {
64:       logger.error("Get all logs error:", error);
65:       throw error;
66:     }
67:   }
68:
69:   /**
70:   * Get logs for a specific user
71:   */
72:   async getUserLogs(userId, limit = 50) {
73:     try {
74:       return await AuditLog.getUserLogs(userId, limit);
75:     } catch (error) {
76:       logger.error("Get user logs error:", error);
77:       throw error;
78:     }
79:   }
80:
81:   /**

```

```

82:     * Get logs for a specific resource
83:     */
84:     async getResourceLogs(resourceType, resourceId, limit = 50) {
85:         try {
86:             return await AuditLog.getResourceLogs(resourceType, resourceId, limit);
87:         } catch (error) {
88:             logger.error("Get resource logs error:", error);
89:             throw error;
90:         }
91:     }
92:
93:     /**
94:     * Get failed actions
95:     */
96:     async getFailedActions(hours = 24, limit = 100) {
97:         try {
98:             return await AuditLog.getFailedActions(hours, limit);
99:         } catch (error) {
100:             logger.error("Get failed actions error:", error);
101:             throw error;
102:         }
103:     }
104:
105:     /**
106:     * Get audit statistics
107:     */
108:     async getStats(startDate, endDate) {
109:         try {
110:             const start = startDate
111:                 ? new Date(startDate)
112:                 : new Date(Date.now() - 7 * 24 * 60 * 60 * 1000);
113:             const end = endDate ? new Date(endDate) : new Date();
114:
115:             const stats = await AuditLog.getStats(start, end);
116:
117:             const totalLogs = await AuditLog.countDocuments({
118:                 createdAt: { $gte: start, $lte: end },
119:             });
120:
121:             const successCount = await AuditLog.countDocuments({
122:                 createdAt: { $gte: start, $lte: end },
123:                 success: true,
124:             });
125:
126:             const failureCount = totalLogs - successCount;
127:
128:             return {
129:                 period: { start, end },
130:                 total: totalLogs,
131:                 success: successCount,
132:                 failure: failureCount,
133:                 byAction: stats,
134:             };
135:         } catch (error) {
136:             logger.error("Get audit stats error:", error);
137:             throw error;
138:         }
139:     }
140:
141:     /**
142:     * Get security events
143:     */
144:     async getSecurityEvents(hours = 24) {
145:         try {
146:             const since = new Date(Date.now() - hours * 60 * 60 * 1000);
147:
148:             const events = await AuditLog.find({
149:                 createdAt: { $gte: since },
150:                 $or: [
151:                     { action: "access_denied" },
152:                     { action: "rate_limit_exceeded" },
153:                     { success: false },
154:                 ],

```

```

155:         })
156:         .sort({ createdAt: -1 })
157:         .limit(100)
158:         .populate("userId", "email role")
159:         .lean();
160:
161:     return events;
162:   } catch (error) {
163:     logger.error("Get security events error:", error);
164:     throw error;
165:   }
166: }
167:
168: /**
169:  * Export audit logs
170:  */
171: async exportLogs(filters = {}, format = "json") {
172:   try {
173:     const query = {};
174:
175:     if (filters.startDate && filters.endDate) {
176:       query.createdAt = {
177:         $gte: new Date(filters.startDate),
178:         $lte: new Date(filters.endDate),
179:       };
180:     }
181:
182:     const logs = await AuditLog.find(query)
183:       .sort({ createdAt: -1 })
184:       .populate("userId", "email role")
185:       .lean();
186:
187:     if (format === "csv") {
188:       return this.convertToCSV(logs);
189:     }
190:
191:     return logs;
192:   } catch (error) {
193:     logger.error("Export logs error:", error);
194:     throw error;
195:   }
196: }
197:
198: /**
199:  * Convert logs to CSV format
200:  */
201: convertToCSV(logs) {
202:   const headers = [
203:     "Timestamp",
204:     "Action",
205:     "Resource Type",
206:     "User Email",
207:     "Success",
208:     "Details",
209:   ];
210:   const rows = logs.map((log) => [
211:     log.createdAt,
212:     log.action,
213:     log.resourceType,
214:     log.userEmail || "N/A",
215:     log.success ? "Yes" : "No",
216:     log.details || "",
217:   ]);
218:
219:   return [headers, ...rows].map((row) => row.join(",")).join("\n");
220: }
221: }
222:
223: module.exports = new AuditService();

```

■ File: src\services\authService.js

```
=====
1: const { User, ROLES } = require("../models");
2: const {
3:   AuditLog,
4:   AUDIT_ACTIONS,
5:   RESOURCE_TYPES,
6: } = require("../models/AuditLog");
7: const {
8:   generateToken,
9:   generateRefreshToken,
10:  verifyRefreshToken,
11: } = require("../middleware/auth");
12: const { AppError } = require("../middleware/errorHandler");
13: const logger = require("../utils/logger");
14:
15: /**
16:  * Authentication Service
17:  */
18: class AuthService {
19:   /**
20:    * Register a new user
21:    */
22:   async register(userData, metadata = {}) {
23:     try {
24:       // Check if user already exists
25:       const existingUser = await User.findOne({ email: userData.email });
26:
27:       if (existingUser) {
28:         throw new AppError("User with this email already exists", 409);
29:       }
30:
31:       // Create user with default role
32:       const user = await User.create({
33:         ...userData,
34:         role: userData.role || ROLES.USER,
35:       });
36:
37:       // Log registration
38:       await AuditLog.log({
39:         action: AUDIT_ACTIONS.CREATE,
40:         resourceType: RESOURCE_TYPES.USER,
41:         resourceId: user._id,
42:         userId: user._id,
43:         userEmail: user.email,
44:         success: true,
45:         metadata,
46:         details: "User registered successfully",
47:       });
48:
49:       // Generate tokens
50:       const accessToken = generateToken(user._id);
51:       const refreshToken = generateRefreshToken(user._id);
52:
53:       logger.info(`New user registered: ${user.email}`);
54:
55:       return {
56:         user: user.toJSON(),
57:         tokens: {
58:           accessToken,
59:           refreshToken,
60:         },
61:       };
62:     } catch (error) {
63:       logger.error("Registration error:", error);
64:       throw error;
65:     }
66:   }
67:
68:   /**
69:    * Login user
70:    */
71:   async login(email, password, metadata = {}) {
=====
```

```

72:     try {
73:         // Find user with password field
74:         const user = await User.findByEmail(email);
75:
76:         if (!user) {
77:             // Log failed login attempt
78:             await AuditLog.logAuth(AUDIT_ACTIONS.LOGIN, null, email, false, {
79:                 ...metadata,
80:                 reason: "User not found",
81:             });
82:
83:             throw new AppError("Invalid email or password", 401);
84:         }
85:
86:         // Check if account is locked
87:         if (user.isLocked()) {
88:             await AuditLog.logAuth(
89:                 AUDIT_ACTIONS.LOGIN,
90:                 user._id,
91:                 user.email,
92:                 false,
93:                 { ...metadata, reason: "Account locked" },
94:             );
95:
96:             throw new AppError(
97:                 "Account is temporarily locked. Please try again later.",
98:                 423,
99:             );
100:         }
101:
102:         // Check if account is active
103:         if (!user.isActive) {
104:             await AuditLog.logAuth(
105:                 AUDIT_ACTIONS.LOGIN,
106:                 user._id,
107:                 user.email,
108:                 false,
109:                 { ...metadata, reason: "Account deactivated" },
110:             );
111:
112:             throw new AppError("Account has been deactivated", 403);
113:         }
114:
115:         // Verify password
116:         const isPasswordValid = await user.comparePassword(password);
117:
118:         if (!isPasswordValid) {
119:             // Increment failed login attempts
120:             await user.incLoginAttempts();
121:
122:             await AuditLog.logAuth(
123:                 AUDIT_ACTIONS.LOGIN,
124:                 user._id,
125:                 user.email,
126:                 false,
127:                 { ...metadata, reason: "Invalid password" },
128:             );
129:
130:             throw new AppError("Invalid email or password", 401);
131:         }
132:
133:         // Reset login attempts on successful login
134:         await user.resetLoginAttempts();
135:
136:         // Log successful login
137:         await AuditLog.logAuth(
138:             AUDIT_ACTIONS.LOGIN,
139:             user._id,
140:             user.email,
141:             true,
142:             metadata,
143:         );
144:

```

```

145:         // Generate tokens
146:         const accessToken = generateToken(user._id);
147:         const refreshToken = generateRefreshToken(user._id);
148:
149:         logger.info(`User logged in: ${user.email}`);
150:
151:         // Remove password from response
152:         const userResponse = user.toJSON();
153:
154:         return {
155:             user: userResponse,
156:             tokens: {
157:                 accessToken,
158:                 refreshToken,
159:             },
160:         };
161:     } catch (error) {
162:         logger.error("Login error:", error);
163:         throw error;
164:     }
165: }
166:
167: /**
168:  * Refresh access token
169:  */
170: async refreshToken(refreshToken) {
171:     try {
172:         // Verify refresh token
173:         const decoded = verifyRefreshToken(refreshToken);
174:
175:         // Find user
176:         const user = await User.findById(decoded.userId);
177:
178:         if (!user || !user.isActive) {
179:             throw new AppError("Invalid refresh token", 401);
180:         }
181:
182:         // Generate new access token
183:         const newAccessToken = generateToken(user._id);
184:
185:         logger.info(`Token refreshed for user: ${user.email}`);
186:
187:         return {
188:             accessToken: newAccessToken,
189:         };
190:     } catch (error) {
191:         logger.error("Token refresh error:", error);
192:         throw new AppError("Invalid or expired refresh token", 401);
193:     }
194: }
195:
196: /**
197:  * Logout user
198:  */
199: async logout(userId, metadata = {}) {
200:     try {
201:         const user = await User.findById(userId);
202:
203:         if (user) {
204:             await AuditLog.logAuth(
205:                 AUDIT_ACTIONS.LOGOUT,
206:                 user._id,
207:                 user.email,
208:                 true,
209:                 metadata,
210:             );
211:
212:             logger.info(`User logged out: ${user.email}`);
213:         }
214:
215:         return { message: "Logged out successfully" };
216:     } catch (error) {
217:         logger.error("Logout error:", error);

```

```

218:         throw error;
219:     }
220: }
221:
222: /**
223:  * Get current user profile
224:  */
225: async getCurrentUser(userId) {
226:     try {
227:         const user = await User.findById(userId);
228:
229:         if (!user) {
230:             throw new AppError("User not found", 404);
231:         }
232:
233:         return user;
234:     } catch (error) {
235:         logger.error("Get current user error:", error);
236:         throw error;
237:     }
238: }
239:
240: /**
241:  * Update user profile
242:  */
243: async updateProfile(userId, updates) {
244:     try {
245:         const user = await User.findById(userId);
246:
247:         if (!user) {
248:             throw new AppError("User not found", 404);
249:         }
250:
251:         // Store old values for audit
252:         const oldValues = {
253:             firstName: user.firstName,
254:             lastName: user.lastName,
255:             email: user.email,
256:         };
257:
258:         // Update allowed fields
259:         const allowedUpdates = ["firstName", "lastName"];
260:         allowedUpdates.forEach((field) => {
261:             if (updates[field] !== undefined) {
262:                 user[field] = updates[field];
263:             }
264:         });
265:
266:         await user.save();
267:
268:         // Log update
269:         await AuditLog.logResourceChange(
270:             AUDIT_ACTIONS.UPDATE,
271:             RESOURCE_TYPES.USER,
272:             user._id,
273:             userId,
274:             { before: oldValues, after: updates },
275:         );
276:
277:         logger.info(`User profile updated: ${user.email}`);
278:
279:         return user;
280:     } catch (error) {
281:         logger.error("Update profile error:", error);
282:         throw error;
283:     }
284: }
285:
286: /**
287:  * Change password
288:  */
289: async changePassword(userId, currentPassword, newPassword) {
290:     try {

```

```

291:         const user = await User.findById(userId).select("+password");
292:
293:         if (!user) {
294:             throw new AppError("User not found", 404);
295:         }
296:
297:         // Verify current password
298:         const isValidPassword = await user.comparePassword(currentPassword);
299:
300:         if (!isValidPassword) {
301:             throw new AppError("Current password is incorrect", 401);
302:         }
303:
304:         // Update password
305:         user.password = newPassword;
306:         await user.save();
307:
308:         // Log password change
309:         await AuditLog.log({
310:             action: AUDIT_ACTIONS.UPDATE,
311:             resourceType: RESOURCE_TYPES.USER,
312:             resourceId: user._id,
313:             userId: user._id,
314:             userEmail: user.email,
315:             success: true,
316:             details: "Password changed successfully",
317:         });
318:
319:         logger.info(`Password changed for user: ${user.email}`);
320:
321:         return { message: "Password changed successfully" };
322:     } catch (error) {
323:         logger.error("Change password error:", error);
324:         throw error;
325:     }
326: }
327: }
328:
329: module.exports = new AuthService();

```

File: src/services/featureToggleService.js

```

1: const FeatureToggle = require("../models/FeatureToggle");
2: const {
3:   AuditLog,
4:   AUDIT_ACTIONS,
5:   RESOURCE_TYPES,
6: } = require("../models/AuditLog");
7: const { AppError } = require("../middleware/errorHandler");
8: const logger = require("../utils/logger");
9: const config = require("../config");
10:
11: /**
12:  * Feature Toggle Service
13:  */
14: class FeatureToggleService {
15:   /**
16:    * Create a new feature toggle
17:    */
18:   async createFeature(featureData, userId) {
19:     try {
20:       // Check if feature already exists
21:       const existing = await FeatureToggle.findOne({
22:         featureName: featureData.featureName,
23:       });
24:
25:       if (existing) {
26:         throw new AppError(
27:           `Feature '${featureData.featureName}' already exists`,
28:           409,
29:         );

```

```

30:     }
31:
32:     // Create feature toggle
33:     const feature = await FeatureToggle.create({
34:       ...featureData,
35:       createdBy: userId,
36:     });
37:
38:     // Log creation
39:     await AuditLog.logResourceChange(
40:       AUDIT_ACTIONS.CREATE,
41:       RESOURCE_TYPES.FEATURE_TOGGLE,
42:       feature._id,
43:       userId,
44:       { after: feature.toJSON() },
45:     );
46:
47:     logger.info(
48:       `Feature toggle created: ${feature.featureName} by user ${userId}`,
49:     );
50:
51:     return feature;
52:   } catch (error) {
53:     logger.error("Create feature toggle error:", error);
54:     throw error;
55:   }
56: }
57:
58: /**
59:  * Get all feature toggles with optional filtering
60:  */
61: async getAllFeatures(filters = {}) {
62:   try {
63:     const query = {};
64:
65:     // Apply filters
66:     if (filters.enabled !== undefined) {
67:       query.enabled = filters.enabled;
68:     }
69:
70:     if (filters.search) {
71:       query.$or = [
72:         { featureName: { $regex: filters.search, $options: "i" } },
73:         { description: { $regex: filters.search, $options: "i" } },
74:       ];
75:     }
76:
77:     const features = await FeatureToggle.find(query)
78:       .populate("createdBy", "email firstName lastName")
79:       .populate("updatedBy", "email firstName lastName")
80:       .sort({ createdAt: -1 });
81:
82:     return features;
83:   } catch (error) {
84:     logger.error("Get all features error:", error);
85:     throw error;
86:   }
87: }
88:
89: /**
90:  * Get enabled features for a specific role
91:  */
92: async getEnabledFeaturesForRole(role, environment = config.env) {
93:   try {
94:     const features = await FeatureToggle.getEnabledFeatures(
95:       role,
96:       environment,
97:     );
98:
99:     return features.map((f) => ({
100:       featureName: f.featureName,
101:       description: f.description,
102:       rolloutPercentage: f.rolloutPercentage,

```

```

103:     });
104:   } catch (error) {
105:     logger.error("Get enabled features error:", error);
106:     throw error;
107:   }
108: }
109:
110: /**
111:  * Get a specific feature by ID
112:  */
113: async getFeatureById(featureId) {
114:   try {
115:     const feature = await FeatureToggle.findById(featureId)
116:       .populate("createdBy", "email firstName lastName")
117:       .populate("updatedBy", "email firstName lastName");
118:
119:     if (!feature) {
120:       throw new AppError("Feature toggle not found", 404);
121:     }
122:
123:     return feature;
124:   } catch (error) {
125:     logger.error("Get feature by ID error:", error);
126:     throw error;
127:   }
128: }
129:
130: /**
131:  * Get a feature by name
132:  */
133: async getFeatureByName(featureName) {
134:   try {
135:     const feature = await FeatureToggle.findOne({ featureName });
136:
137:     if (!feature) {
138:       throw new AppError("Feature toggle not found", 404);
139:     }
140:
141:     return feature;
142:   } catch (error) {
143:     logger.error("Get feature by name error:", error);
144:     throw error;
145:   }
146: }
147:
148: /**
149:  * Update a feature toggle
150:  */
151: async updateFeature(featureId, updates, userId) {
152:   try {
153:     const feature = await FeatureToggle.findById(featureId);
154:
155:     if (!feature) {
156:       throw new AppError("Feature toggle not found", 404);
157:     }
158:
159:     // Store old values for audit
160:     const oldValues = feature.toJSON();
161:
162:     // Update fields
163:     Object.keys(updates).forEach((key) => {
164:       if (updates[key] !== undefined) {
165:         feature[key] = updates[key];
166:       }
167:     });
168:
169:     feature.updatedBy = userId;
170:     await feature.save();
171:
172:     // Log update
173:     await AuditLog.logResourceChange(
174:       AUDIT_ACTIONS.UPDATE,
175:       RESOURCE_TYPES.FEATURE_TOGGLE,

```

```

176:         feature._id,
177:         userId,
178:         { before: oldValues, after: updates },
179:     );
180:
181:     logger.info(
182:         `Feature toggle updated: ${feature.featureName} by user ${userId}`,
183:     );
184:
185:     return feature;
186: } catch (error) {
187:     logger.error("Update feature toggle error:", error);
188:     throw error;
189: }
190: }
191:
192: /**
193:  * Toggle feature enable/disable
194:  */
195: async toggleFeature(featureId, enabled, userId) {
196:     try {
197:         const feature = await FeatureToggle.findById(featureId);
198:
199:         if (!feature) {
200:             throw new AppError("Feature toggle not found", 404);
201:         }
202:
203:         const oldValue = feature.enabled;
204:         feature.enabled = enabled;
205:         feature.updatedBy = userId;
206:         await feature.save();
207:
208:         // Log toggle
209:         await AuditLog.logResourceChange(
210:             AUDIT_ACTIONS.UPDATE,
211:             RESOURCE_TYPES.FEATURE_TOGGLE,
212:             feature._id,
213:             userId,
214:             {
215:                 before: { enabled: oldValue },
216:                 after: { enabled },
217:             },
218:         );
219:
220:         logger.info(
221:             `Feature ${enabled ? "enabled" : "disabled"}: ${feature.featureName}`,
222:         );
223:
224:         return feature;
225:     } catch (error) {
226:         logger.error("Toggle feature error:", error);
227:         throw error;
228:     }
229: }
230:
231: /**
232:  * Delete a feature toggle
233:  */
234: async deleteFeature(featureId, userId) {
235:     try {
236:         const feature = await FeatureToggle.findById(featureId);
237:
238:         if (!feature) {
239:             throw new AppError("Feature toggle not found", 404);
240:         }
241:
242:         const featureName = feature.featureName;
243:
244:         await feature.deleteOne();
245:
246:         // Log deletion
247:         await AuditLog.logResourceChange(
248:             AUDIT_ACTIONS.DELETE,

```



```

249:         RESOURCE_TYPES.FEATURE_TOGGLE,
250:         featureId,
251:         userId,
252:         { before: feature.toJSON() },
253:     );
254:
255:     logger.info(`Feature toggle deleted: ${featureName} by user ${userId}`);
256:
257:     return { message: "Feature toggle deleted successfully", featureName };
258: } catch (error) {
259:     logger.error("Delete feature toggle error:", error);
260:     throw error;
261: }
262: }
263:
264: /**
265:  * Check if a feature is enabled for user
266:  */
267: async checkFeatureAccess(featureName, role, userId = null) {
268:     try {
269:         const environment = config.env;
270:         const isEnabled = await FeatureToggle.checkFeature(
271:             featureName,
272:             role,
273:             environment,
274:             userId,
275:         );
276:
277:         return {
278:             featureName,
279:             enabled: isEnabled,
280:             role,
281:             environment,
282:         };
283:     } catch (error) {
284:         logger.error("Check feature access error:", error);
285:         throw error;
286:     }
287: }
288:
289: /**
290:  * Bulk update features
291:  */
292: async bulkUpdateFeatures(updates, userId) {
293:     try {
294:         const results = [];
295:
296:         for (const update of updates) {
297:             try {
298:                 const feature = await this.updateFeature(
299:                     update.id,
300:                     update.data,
301:                     userId,
302:                 );
303:                 results.push({ id: update.id, success: true, feature });
304:             } catch (error) {
305:                 results.push({ id: update.id, success: false, error: error.message });
306:             }
307:         }
308:
309:         return results;
310:     } catch (error) {
311:         logger.error("Bulk update features error:", error);
312:         throw error;
313:     }
314: }
315:
316: /**
317:  * Get feature statistics
318:  */
319: async getFeatureStats() {
320:     try {
321:         const total = await FeatureToggle.countDocuments();

```

```

322:     const enabled = await FeatureToggle.countDocuments({ enabled: true });
323:     const disabled = total - enabled;
324:
325:     const byEnvironment = await FeatureToggle.aggregate([
326:     {
327:         $project: {
328:             devEnabled: "$environments.development.enabled",
329:             stagingEnabled: "$environments.staging.enabled",
330:             prodEnabled: "$environments.production.enabled",
331:         },
332:     },
333:     {
334:         $group: {
335:             _id: null,
336:             development: { $sum: { $cond: ["$devEnabled", 1, 0] } },
337:             staging: { $sum: { $cond: ["$stagingEnabled", 1, 0] } },
338:             production: { $sum: { $cond: ["$prodEnabled", 1, 0] } },
339:         },
340:     },
341:     ]);
342:
343:     return {
344:         total,
345:         enabled,
346:         disabled,
347:         byEnvironment: byEnvironment[0] || {
348:             development: 0,
349:             staging: 0,
350:             production: 0,
351:         },
352:     };
353: } catch (error) {
354:     logger.error("Get feature stats error:", error);
355:     throw error;
356: }
357: }
358: }
359:
360: module.exports = new FeatureToggleService();

```

■ File: src\services\index.js

```

1: /**
2:  * Services Index
3:  * Centralized export of all business logic services
4:  */
5:
6: const authService = require("./authService");
7: const userService = require("./userService");
8: const featureToggleService = require("./featureToggleService");
9: const rateGuardService = require("./rateGuardService");
10: const auditService = require("./auditService");
11:
12: module.exports = {
13:     authService,
14:     userService,
15:     featureToggleService,
16:     rateGuardService,
17:     auditService,
18: };

```

■ File: src\services\rateGuardService.js

```

1: const RateGuard = require("../models/RateGuard");
2: const {
3:     AuditLog,
4:     AUDIT_ACTIONS,
5:     RESOURCE_TYPES,
6: } = require("../models/AuditLog");

```

```

7: const { AppError } = require("../middleware/errorHandler");
8: const { clearRateLimiters } = require("../middleware/rateGuard");
9: const logger = require("../utils/logger");
10:
11: /**
12:  * Rate Guard Service
13:  */
14: class RateGuardService {
15:     /**
16:      * Create a new rate guard rule
17:      */
18:     async createRule(ruleData, userId) {
19:         try {
20:             // Check if rule already exists for this route and method
21:             const existing = await RateGuard.findOne({
22:                 routePath: ruleData.routePath,
23:                 method: ruleData.method || "ALL",
24:             });
25:
26:             if (existing) {
27:                 throw new AppError(
28:                     `Rate guard rule already exists for ${ruleData.method || "ALL"} ${ruleData.routePath}`,
29:                     409,
30:                 );
31:             }
32:
33:             // Create rate guard rule
34:             const rule = await RateGuard.create({
35:                 ...ruleData,
36:                 createdBy: userId,
37:             });
38:
39:             // Clear rate limiter cache to apply new rule
40:             clearRateLimiters();
41:
42:             // Log creation
43:             await AuditLog.logResourceChange(
44:                 AUDIT_ACTIONS.CREATE,
45:                 RESOURCE_TYPES.RATE_GUARD,
46:                 rule._id,
47:                 userId,
48:                 { after: rule.toJSON() },
49:             );
50:
51:             logger.info(
52:                 `Rate guard rule created: ${rule.displayName} by user ${userId}`,
53:             );
54:
55:             return rule;
56:         } catch (error) {
57:             logger.error("Create rate guard rule error:", error);
58:             throw error;
59:         }
60:     }
61:
62:     /**
63:      * Get all rate guard rules with optional filtering
64:      */
65:     async getAllRules(filters = {}) {
66:         try {
67:             const query = {};
68:
69:             // Apply filters
70:             if (filters.enabled !== undefined) {
71:                 query.enabled = filters.enabled;
72:             }
73:
74:             if (filters.method) {
75:                 query.method = filters.method.toUpperCase();
76:             }
77:
78:             if (filters.search) {
79:                 query.$or = [

```

```

80:         { routePath: { $regex: filters.search, $options: "i" } },
81:         { description: { $regex: filters.search, $options: "i" } },
82:     ];
83:     }
84:
85:     const rules = await RateGuard.find(query)
86:     .populate("createdBy", "email firstName lastName")
87:     .populate("updatedBy", "email firstName lastName")
88:     .sort({ routePath: 1 });
89:
90:     return rules;
91:   } catch (error) {
92:     logger.error("Get all rate guard rules error:", error);
93:     throw error;
94:   }
95: }
96:
97: /**
98:  * Get active rate guard rules
99:  */
100: async getActiveRules() {
101:   try {
102:     return await RateGuard.getActiveRules();
103:   } catch (error) {
104:     logger.error("Get active rules error:", error);
105:     throw error;
106:   }
107: }
108:
109: /**
110:  * Get a specific rule by ID
111:  */
112: async getRuleById(ruleId) {
113:   try {
114:     const rule = await RateGuard.findById(ruleId)
115:     .populate("createdBy", "email firstName lastName")
116:     .populate("updatedBy", "email firstName lastName");
117:
118:     if (!rule) {
119:       throw new AppError("Rate guard rule not found", 404);
120:     }
121:
122:     return rule;
123:   } catch (error) {
124:     logger.error("Get rule by ID error:", error);
125:     throw error;
126:   }
127: }
128:
129: /**
130:  * Find rule for a specific route
131:  */
132: async findRuleForRoute(routePath, method = "ALL") {
133:   try {
134:     const rule = await RateGuard.findRuleForRoute(routePath, method);
135:     return rule;
136:   } catch (error) {
137:     logger.error("Find rule for route error:", error);
138:     throw error;
139:   }
140: }
141:
142: /**
143:  * Update a rate guard rule
144:  */
145: async updateRule(ruleId, updates, userId) {
146:   try {
147:     const rule = await RateGuard.findById(ruleId);
148:
149:     if (!rule) {
150:       throw new AppError("Rate guard rule not found", 404);
151:     }
152:

```

```

153:         // Store old values for audit
154:         const oldValues = rule.toJSON();
155:
156:         // Update fields
157:         Object.keys(updates).forEach((key) => {
158:             if (updates[key] !== undefined) {
159:                 rule[key] = updates[key];
160:             }
161:         });
162:
163:         rule.updatedBy = userId;
164:         await rule.save();
165:
166:         // Clear rate limiter cache to apply changes
167:         clearRateLimiters();
168:
169:         // Log update
170:         await AuditLog.logResourceChange(
171:             AUDIT_ACTIONS.UPDATE,
172:             RESOURCE_TYPES.RATE_GUARD,
173:             rule._id,
174:             userId,
175:             { before: oldValues, after: updates },
176:         );
177:
178:         logger.info(
179:             `Rate guard rule updated: ${rule.displayName} by user ${userId}`,
180:         );
181:
182:         return rule;
183:     } catch (error) {
184:         logger.error("Update rate guard rule error:", error);
185:         throw error;
186:     }
187: }
188:
189: /**
190:  * Toggle rule enable/disable
191:  */
192: async toggleRule(ruleId, enabled, userId) {
193:     try {
194:         const rule = await RateGuard.findById(ruleId);
195:
196:         if (!rule) {
197:             throw new AppError("Rate guard rule not found", 404);
198:         }
199:
200:         const oldValue = rule.enabled;
201:         rule.enabled = enabled;
202:         rule.updatedBy = userId;
203:         await rule.save();
204:
205:         // Clear rate limiter cache
206:         clearRateLimiters();
207:
208:         // Log toggle
209:         await AuditLog.logResourceChange(
210:             AUDIT_ACTIONS.UPDATE,
211:             RESOURCE_TYPES.RATE_GUARD,
212:             rule._id,
213:             userId,
214:             {
215:                 before: { enabled: oldValue },
216:                 after: { enabled },
217:             },
218:         );
219:
220:         logger.info(
221:             `Rate guard rule ${enabled ? "enabled" : "disabled"}: ${rule.displayName}`,
222:         );
223:
224:         return rule;
225:     } catch (error) {

```

```

226:     logger.error("Toggle rate guard rule error:", error);
227:     throw error;
228:   }
229: }
230:
231: /**
232:  * Delete a rate guard rule
233:  */
234: async deleteRule(ruleId, userId) {
235:   try {
236:     const rule = await RateGuard.findById(ruleId);
237:
238:     if (!rule) {
239:       throw new AppError("Rate guard rule not found", 404);
240:     }
241:
242:     const displayName = rule.displayName;
243:
244:     await rule.deleteOne();
245:
246:     // Clear rate limiter cache
247:     clearRateLimiters();
248:
249:     // Log deletion
250:     await AuditLog.logResourceChange(
251:       AUDIT_ACTIONS.DELETE,
252:       RESOURCE_TYPES.RATE_GUARD,
253:       ruleId,
254:       userId,
255:       { before: rule.toJSON() },
256:     );
257:
258:     logger.info(`Rate guard rule deleted: ${displayName} by user ${userId}`);
259:
260:     return { message: "Rate guard rule deleted successfully", displayName };
261:   } catch (error) {
262:     logger.error("Delete rate guard rule error:", error);
263:     throw error;
264:   }
265: }
266:
267: /**
268:  * Add user/IP to whitelist
269:  */
270: async addToWhitelist(ruleId, identifier, userId) {
271:   try {
272:     const rule = await RateGuard.findById(ruleId);
273:
274:     if (!rule) {
275:       throw new AppError("Rate guard rule not found", 404);
276:     }
277:
278:     if (rule.whitelist.includes(identifier)) {
279:       throw new AppError("Identifier already in whitelist", 409);
280:     }
281:
282:     rule.whitelist.push(identifier);
283:     rule.updatedBy = userId;
284:     await rule.save();
285:
286:     // Clear rate limiter cache
287:     clearRateLimiters();
288:
289:     logger.info(
290:       `Added to whitelist: ${identifier} for rule ${rule.displayName}`,
291:     );
292:
293:     return rule;
294:   } catch (error) {
295:     logger.error("Add to whitelist error:", error);
296:     throw error;
297:   }
298: }

```

```

299:
300: /**
301:  * Remove user/IP from whitelist
302:  */
303: async removeFromWhitelist(ruleId, identifier, userId) {
304:   try {
305:     const rule = await RateGuard.findById(ruleId);
306:
307:     if (!rule) {
308:       throw new AppError("Rate guard rule not found", 404);
309:     }
310:
311:     rule.whitelist = rule.whitelist.filter((item) => item !== identifier);
312:     rule.updatedBy = userId;
313:     await rule.save();
314:
315:     // Clear rate limiter cache
316:     clearRateLimiters();
317:
318:     logger.info(
319:       `Removed from whitelist: ${identifier} for rule ${rule.displayName}`,
320:     );
321:
322:     return rule;
323:   } catch (error) {
324:     logger.error("Remove from whitelist error:", error);
325:     throw error;
326:   }
327: }
328:
329: /**
330:  * Get rate guard statistics
331:  */
332: async getRateGuardStats() {
333:   try {
334:     const total = await RateGuard.countDocuments();
335:     const enabled = await RateGuard.countDocuments({ enabled: true });
336:     const disabled = total - enabled;
337:
338:     const byMethod = await RateGuard.aggregate([
339:       {
340:         $group: {
341:           _id: "$method",
342:           count: { $sum: 1 },
343:         },
344:       },
345:     ]);
346:
347:     const ipBased = await RateGuard.countDocuments({ ipBased: true });
348:
349:     return {
350:       total,
351:       enabled,
352:       disabled,
353:       ipBased,
354:       byMethod: byMethod.reduce((acc, item) => {
355:         acc[item._id] = item.count;
356:         return acc;
357:       }, {}),
358:     };
359:   } catch (error) {
360:     logger.error("Get rate guard stats error:", error);
361:     throw error;
362:   }
363: }
364:
365: /**
366:  * Test rate limit for a route
367:  */
368: async testRateLimit(routePath, method, role) {
369:   try {
370:     const rule = await this.findRuleForRoute(routePath, method);
371:

```

```

372:         if (!rule) {
373:             return {
374:                 hasRule: false,
375:                 message: "No rate limit rule found for this route",
376:             };
377:         }
378:
379:         const limit = rule.getLimitForRole(role);
380:
381:         return {
382:             hasRule: true,
383:             enabled: rule.enabled,
384:             limit,
385:             displayName: rule.displayName,
386:         };
387:     } catch (error) {
388:         logger.error("Test rate limit error:", error);
389:         throw error;
390:     }
391: }
392: }
393:
394: module.exports = new RateGuardService();

```

File: src\services\userService.js

```

=====
1: const { User, ROLES } = require("../models");
2: const {
3:   AuditLog,
4:   AUDIT_ACTIONS,
5:   RESOURCE_TYPES,
6: } = require("../models/AuditLog");
7: const { AppError } = require("../middleware/errorHandler");
8: const logger = require("../utils/logger");
9:
10: /**
11:  * User Service
12:  */
13: class UserService {
14:   /**
15:    * Get all users with filtering and pagination
16:    */
17:   async getAllUsers(filters = {}, pagination = {}) {
18:     try {
19:       const {
20:         page = 1,
21:         limit = 10,
22:         sort = "-createdAt",
23:         search = "",
24:       } = pagination;
25:
26:       const query = {};
27:
28:       // Apply role filter
29:       if (filters.role) {
30:         query.role = filters.role;
31:       }
32:
33:       // Apply active status filter
34:       if (filters.isActive !== undefined) {
35:         query.isActive = filters.isActive;
36:       }
37:
38:       // Apply search
39:       if (search) {
40:         query.$or = [
41:           { email: { $regex: search, $options: "i" } },
42:           { firstName: { $regex: search, $options: "i" } },
43:           { lastName: { $regex: search, $options: "i" } },
44:         ];
45:       }

```



```

46:
47:     const skip = (page - 1) * limit;
48:
49:     const [users, total] = await Promise.all([
50:       User.find(query).sort(sort).skip(skip).limit(limit).lean(),
51:       User.countDocuments(query),
52:     ]);
53:
54:     return {
55:       users,
56:       pagination: {
57:         page,
58:         limit,
59:         total,
60:         pages: Math.ceil(total / limit),
61:       },
62:     };
63:   } catch (error) {
64:     logger.error("Get all users error:", error);
65:     throw error;
66:   }
67: }
68:
69: /**
70:  * Get user by ID
71:  */
72: async getUserById(userId) {
73:   try {
74:     const user = await User.findById(userId);
75:
76:     if (!user) {
77:       throw new AppError("User not found", 404);
78:     }
79:
80:     return user;
81:   } catch (error) {
82:     logger.error("Get user by ID error:", error);
83:     throw error;
84:   }
85: }
86:
87: /**
88:  * Update user (Admin function)
89:  */
90: async updateUser(userId, updates, adminId) {
91:   try {
92:     const user = await User.findById(userId);
93:
94:     if (!user) {
95:       throw new AppError("User not found", 404);
96:     }
97:
98:     // Store old values for audit
99:     const oldValues = {
100:       firstName: user.firstName,
101:       lastName: user.lastName,
102:       email: user.email,
103:       role: user.role,
104:       isActive: user.isActive,
105:     };
106:
107:     // Update allowed fields
108:     const allowedUpdates = [
109:       "firstName",
110:       "lastName",
111:       "email",
112:       "role",
113:       "isActive",
114:     ];
115:     allowedUpdates.forEach((field) => {
116:       if (updates[field] !== undefined) {
117:         user[field] = updates[field];
118:       }

```

```

119:     });
120:
121:     await user.save();
122:
123:     // Log update
124:     await AuditLog.logResourceChange(
125:         AUDIT_ACTIONS.UPDATE,
126:         RESOURCE_TYPES.USER,
127:         user._id,
128:         adminId,
129:         { before: oldValues, after: updates },
130:     );
131:
132:     logger.info(`User updated: ${user.email} by admin ${adminId}`);
133:
134:     return user;
135: } catch (error) {
136:     logger.error("Update user error:", error);
137:     throw error;
138: }
139: }
140:
141: /**
142:  * Deactivate user
143:  */
144: async deactivateUser(userId, adminId) {
145:     try {
146:         const user = await User.findById(userId);
147:
148:         if (!user) {
149:             throw new AppError("User not found", 404);
150:         }
151:
152:         if (!user.isActive) {
153:             throw new AppError("User is already deactivated", 400);
154:         }
155:
156:         user.isActive = false;
157:         await user.save();
158:
159:         // Log deactivation
160:         await AuditLog.log({
161:             action: AUDIT_ACTIONS.UPDATE,
162:             resourceType: RESOURCE_TYPES.USER,
163:             resourceId: user._id,
164:             userId: adminId,
165:             success: true,
166:             details: `User deactivated: ${user.email}`,
167:         });
168:
169:         logger.info(`User deactivated: ${user.email} by admin ${adminId}`);
170:
171:         return user;
172:     } catch (error) {
173:         logger.error("Deactivate user error:", error);
174:         throw error;
175:     }
176: }
177:
178: /**
179:  * Activate user
180:  */
181: async activateUser(userId, adminId) {
182:     try {
183:         const user = await User.findById(userId);
184:
185:         if (!user) {
186:             throw new AppError("User not found", 404);
187:         }
188:
189:         if (user.isActive) {
190:             throw new AppError("User is already active", 400);
191:         }

```

```

192:
193:     user.isActive = true;
194:     await user.save();
195:
196:     // Log activation
197:     await AuditLog.log({
198:         action: AUDIT_ACTIONS.UPDATE,
199:         resourceType: RESOURCE_TYPES.USER,
200:         resourceId: user._id,
201:         userId: adminId,
202:         success: true,
203:         details: `User activated: ${user.email}`,
204:     });
205:
206:     logger.info(`User activated: ${user.email} by admin ${adminId}`);
207:
208:     return user;
209: } catch (error) {
210:     logger.error("Activate user error:", error);
211:     throw error;
212: }
213: }
214:
215: /**
216:  * Delete user
217:  */
218: async deleteUser(userId, adminId) {
219:     try {
220:         const user = await User.findById(userId);
221:
222:         if (!user) {
223:             throw new AppError("User not found", 404);
224:         }
225:
226:         // Prevent self-deletion
227:         if (userId === adminId) {
228:             throw new AppError("You cannot delete your own account", 400);
229:         }
230:
231:         const userEmail = user.email;
232:
233:         await user.deleteOne();
234:
235:         // Log deletion
236:         await AuditLog.logResourceChange(
237:             AUDIT_ACTIONS.DELETE,
238:             RESOURCE_TYPES.USER,
239:             userId,
240:             adminId,
241:             { before: user.toJSON() },
242:         );
243:
244:         logger.info(`User deleted: ${userEmail} by admin ${adminId}`);
245:
246:         return { message: "User deleted successfully", email: userEmail };
247:     } catch (error) {
248:         logger.error("Delete user error:", error);
249:         throw error;
250:     }
251: }
252:
253: /**
254:  * Change user role
255:  */
256: async changeUserRole(userId, newRole, adminId) {
257:     try {
258:         const user = await User.findById(userId);
259:
260:         if (!user) {
261:             throw new AppError("User not found", 404);
262:         }
263:
264:         const oldRole = user.role;

```

```

265:
266:     if (oldRole === newRole) {
267:         throw new AppError("User already has this role", 400);
268:     }
269:
270:     user.role = newRole;
271:     await user.save();
272:
273:     // Log role change
274:     await AuditLog.logResourceChange(
275:         AUDIT_ACTIONS.UPDATE,
276:         RESOURCE_TYPES.USER,
277:         user._id,
278:         adminId,
279:         {
280:             before: { role: oldRole },
281:             after: { role: newRole },
282:         },
283:     );
284:
285:     logger.info(
286:         `User role changed: ${user.email} from ${oldRole} to ${newRole}`,
287:     );
288:
289:     return user;
290: } catch (error) {
291:     logger.error("Change user role error:", error);
292:     throw error;
293: }
294: }
295:
296: /**
297:  * Get user statistics
298:  */
299: async getUserStats() {
300:     try {
301:         const total = await User.countDocuments();
302:         const active = await User.countDocuments({ isActive: true });
303:         const inactive = total - active;
304:
305:         const byRole = await User.aggregate([
306:             {
307:                 $group: {
308:                     _id: "$role",
309:                     count: { $sum: 1 },
310:                 },
311:             },
312:         ]);
313:
314:         const recentLogins = await User.find({ lastLogin: { $exists: true } })
315:             .sort({ lastLogin: -1 })
316:             .limit(10)
317:             .select("email lastLogin role");
318:
319:         return {
320:             total,
321:             active,
322:             inactive,
323:             byRole: byRole.reduce((acc, item) => {
324:                 acc[item._id] = item.count;
325:                 return acc;
326:             }, {}),
327:             recentLogins,
328:         };
329:     } catch (error) {
330:         logger.error("Get user stats error:", error);
331:         throw error;
332:     }
333: }
334:
335: /**
336:  * Unlock user account
337:  */

```

```

338:   async unlockUser(userId, adminId) {
339:     try {
340:       const user = await User.findById(userId);
341:
342:       if (!user) {
343:         throw new AppError("User not found", 404);
344:       }
345:
346:       user.loginAttempts = 0;
347:       user.lockUntil = undefined;
348:       await user.save();
349:
350:       // Log unlock
351:       await AuditLog.log({
352:         action: AUDIT_ACTIONS.UPDATE,
353:         resourceType: RESOURCE_TYPES.USER,
354:         resourceId: user._id,
355:         userId: adminId,
356:         success: true,
357:         details: `User account unlocked: ${user.email}`,
358:       });
359:
360:       logger.info(`User unlocked: ${user.email} by admin ${adminId}`);
361:
362:       return user;
363:     } catch (error) {
364:       logger.error("Unlock user error:", error);
365:       throw error;
366:     }
367:   }
368: }
369:
370: module.exports = new UserService();

```

File: src\utils\logger.js

```

1: const winston = require("winston");
2: const config = require("../config");
3: const path = require("path");
4: const fs = require("fs");
5:
6: // Ensure logs directory exists
7: const logDir = path.dirname(config.logging.file);
8: if (!fs.existsSync(logDir)) {
9:   fs.mkdirSync(logDir, { recursive: true });
10: }
11:
12: /**
13:  * Custom log format with timestamp and colorization
14:  */
15: const logFormat = winston.format.combine(
16:   winston.format.timestamp({ format: "YYYY-MM-DD HH:mm:ss" }),
17:   winston.format.errors({ stack: true }),
18:   winston.format.splat(),
19:   winston.format.json(),
20: );
21:
22: /**
23:  * Console format with colors for development
24:  */
25: const consoleFormat = winston.format.combine(
26:   winston.format.colorize(),
27:   winston.format.timestamp({ format: "YYYY-MM-DD HH:mm:ss" }),
28:   winston.format.printf(({ timestamp, level, message, ...meta }) => {
29:     let msg = `${timestamp} [${level}]: ${message}`;
30:     if (Object.keys(meta).length > 0) {
31:       msg += ` ${JSON.stringify(meta)}`;
32:     }
33:     return msg;
34:   }),
35: );

```

```

36:
37: /**
38:  * Winston Logger Instance
39:  */
40: const logger = winston.createLogger({
41:   level: config.logging.level,
42:   format: logFormat,
43:   defaultMeta: { service: "policy-toggle-service" },
44:   transports: [
45:     // Write all logs to file
46:     new winston.transports.File({
47:       filename: config.logging.file,
48:       maxsize: 5242880, // 5MB
49:       maxFiles: 5,
50:     }),
51:     // Write errors to separate file
52:     new winston.transports.File({
53:       filename: path.join(logDir, "error.log"),
54:       level: "error",
55:       maxsize: 5242880,
56:       maxFiles: 5,
57:     }),
58:   ],
59:   // Handle uncaught exceptions
60:   exceptionHandlers: [
61:     new winston.transports.File({
62:       filename: path.join(logDir, "exceptions.log"),
63:     }),
64:   ],
65:   rejectionHandlers: [
66:     new winston.transports.File({
67:       filename: path.join(logDir, "rejections.log"),
68:     }),
69:   ],
70: });
71:
72: /**
73:  * Add console output in development
74:  */
75: if (config.env !== "production") {
76:   logger.add(
77:     new winston.transports.Console({
78:       format: consoleFormat,
79:     }),
80:   );
81: }
82:
83: /**
84:  * Stream for Morgan HTTP logging
85:  */
86: logger.stream = {
87:   write: (message) => {
88:     logger.info(message.trim());
89:   },
90: };
91:
92: module.exports = logger;

```

■ File: tests\helpers.js

```

=====
1: const { User, FeatureToggle, RateGuard, ROLES } = require("../src/models");
2: const { generateToken } = require("../src/middleware/auth");
3:
4: /**
5:  * Create test user
6:  */
7: const createTestUser = async (overrides = {}) => {
8:   const defaultUser = {
9:     email: `test-${Date.now()}@example.com`,
10:    password: "password123",
11:    firstName: "Test",

```

```

12:     lastName: "User",
13:     role: ROLES.USER,
14:     isActive: true,
15:   };
16:
17:   const user = await User.create({ ...defaultUser, ...overrides });
18:   return user;
19: };
20:
21: /**
22:  * Create test admin
23:  */
24: const createTestAdmin = async (overrides = {}) => {
25:   return createTestUser({ ...overrides, role: ROLES.ADMIN });
26: };
27:
28: /**
29:  * Create test guest
30:  */
31: const createTestGuest = async (overrides = {}) => {
32:   return createTestUser({ ...overrides, role: ROLES.GUEST });
33: };
34:
35: /**
36:  * Generate auth token for user
37:  */
38: const getAuthToken = (userId) => {
39:   return generateToken(userId);
40: };
41:
42: /**
43:  * Create authenticated request header
44:  */
45: const getAuthHeader = (token) => {
46:   return { Authorization: `Bearer ${token}` };
47: };
48:
49: /**
50:  * Create test feature toggle
51:  */
52: const createTestFeature = async (userId, overrides = {}) => {
53:   const defaultFeature = {
54:     featureName: `test-feature-${Date.now()}`,
55:     description: "Test feature",
56:     enabled: true,
57:     allowedRoles: [ROLES.ADMIN, ROLES.USER],
58:     rolloutPercentage: 100,
59:     environments: {
60:       development: { enabled: true },
61:       staging: { enabled: true },
62:       production: { enabled: false },
63:     },
64:     createdBy: userId,
65:   };
66:
67:   const feature = await FeatureToggle.create({
68:     ...defaultFeature,
69:     ...overrides,
70:   });
71:   return feature;
72: };
73:
74: /**
75:  * Create test rate guard rule
76:  */
77: const createTestRateGuard = async (userId, overrides = {}) => {
78:   const defaultRule = {
79:     routePath: `/test/${Date.now()}`,
80:     method: "ALL",
81:     description: "Test rate guard rule",
82:     enabled: true,
83:     limits: {
84:       admin: {

```

```

85:         maxRequests: 1000,
86:         windowMs: 60000,
87:     },
88:     user: {
89:         maxRequests: 100,
90:         windowMs: 60000,
91:     },
92:     guest: {
93:         maxRequests: 10,
94:         windowMs: 60000,
95:     },
96: },
97: createdBy: userId,
98: };
99:
100: const rule = await RateGuard.create({ ...defaultRule, ...overrides });
101: return rule;
102: };
103:
104: /**
105:  * Wait for async operations
106:  */
107: const wait = (ms) => new Promise((resolve) => setTimeout(resolve, ms));
108:
109: /**
110:  * Extract error message from response
111:  */
112: const getErrorMessage = (response) => {
113:     return response.body.message || response.body.error || "Unknown error";
114: };
115:
116: module.exports = {
117:     createTestUser,
118:     createTestAdmin,
119:     createTestGuest,
120:     getAuthToken,
121:     getAuthHeader,
122:     createTestFeature,
123:     createTestRateGuard,
124:     wait,
125:     getErrorMessage,
126: };

```

File: tests\integration\auth.routes.test.js

```

=====
1: const request = require("supertest");
2: const app = require("../src/app");
3: const { User, ROLES } = require("../src/models");
4: const { createTestUser, getAuthHeader, getAuthToken } = require("../helpers");
5:
6: describe("Auth Routes", () => {
7:     describe("POST /api/auth/register", () => {
8:         it("should register a new user successfully", async () => {
9:             const userData = {
10:                 email: "newuser@example.com",
11:                 password: "password123",
12:                 firstName: "New",
13:                 lastName: "User",
14:             };
15:
16:             const response = await request(app)
17:                 .post("/api/auth/register")
18:                 .send(userData)
19:                 .expect(201);
20:
21:             expect(response.body.success).toBe(true);
22:             expect(response.body.data.user.email).toBe(userData.email);
23:             expect(response.body.data.tokens.accessToken).toBeDefined();
24:             expect(response.body.data.tokens.refreshToken).toBeDefined();
25:         });
26:     });

```



```

27:     it("should not register user with existing email", async () => {
28:         const email = "existing@example.com";
29:         await createTestUser({ email });
30:
31:         const response = await request(app)
32:             .post("/api/auth/register")
33:             .send({
34:                 email,
35:                 password: "password123",
36:             })
37:             .expect(409);
38:
39:         expect(response.body.success).toBe(false);
40:     });
41:
42:     it("should validate required fields", async () => {
43:         const response = await request(app)
44:             .post("/api/auth/register")
45:             .send({
46:                 email: "test@example.com",
47:                 // Missing password
48:             })
49:             .expect(400);
50:
51:         expect(response.body.success).toBe(false);
52:     });
53:
54:     it("should validate email format", async () => {
55:         const response = await request(app)
56:             .post("/api/auth/register")
57:             .send({
58:                 email: "invalid-email",
59:                 password: "password123",
60:             })
61:             .expect(400);
62:
63:         expect(response.body.success).toBe(false);
64:     });
65: });
66:
67: describe("POST /api/auth/login", () => {
68:     let testUser;
69:     const password = "password123";
70:
71:     beforeEach(async () => {
72:         testUser = await createTestUser({ password });
73:     });
74:
75:     it("should login successfully with correct credentials", async () => {
76:         const response = await request(app)
77:             .post("/api/auth/login")
78:             .send({
79:                 email: testUser.email,
80:                 password,
81:             })
82:             .expect(200);
83:
84:         expect(response.body.success).toBe(true);
85:         expect(response.body.data.user.email).toBe(testUser.email);
86:         expect(response.body.data.tokens.accessToken).toBeDefined();
87:         expect(response.body.data.tokens.refreshToken).toBeDefined();
88:     });
89:
90:     it("should not login with incorrect password", async () => {
91:         const response = await request(app)
92:             .post("/api/auth/login")
93:             .send({
94:                 email: testUser.email,
95:                 password: "wrongpassword",
96:             })
97:             .expect(401);
98:
99:         expect(response.body.success).toBe(false);

```

```

100:     });
101:
102:     it("should not login with non-existent email", async () => {
103:         const response = await request(app)
104:             .post("/api/auth/login")
105:             .send({
106:                 email: "nonexistent@example.com",
107:                 password,
108:             })
109:             .expect(401);
110:
111:         expect(response.body.success).toBe(false);
112:     });
113:
114:     it("should not login with inactive account", async () => {
115:         testUser.isActive = false;
116:         await testUser.save();
117:
118:         const response = await request(app)
119:             .post("/api/auth/login")
120:             .send({
121:                 email: testUser.email,
122:                 password,
123:             })
124:             .expect(403);
125:
126:         expect(response.body.success).toBe(false);
127:     });
128:
129:     it("should increment login attempts on failed login", async () => {
130:         await request(app).post("/api/auth/login").send({
131:             email: testUser.email,
132:             password: "wrongpassword",
133:         });
134:
135:         const updatedUser = await User.findById(testUser._id);
136:         expect(updatedUser.loginAttempts).toBe(1);
137:     });
138: });
139:
140: describe("GET /api/auth/me", () => {
141:     let testUser;
142:     let authToken;
143:
144:     beforeEach(async () => {
145:         testUser = await createTestUser();
146:         authToken = getAuthToken(testUser._id);
147:     });
148:
149:     it("should return current user profile", async () => {
150:         const response = await request(app)
151:             .get("/api/auth/me")
152:             .set(getAuthHeader(authToken))
153:             .expect(200);
154:
155:         expect(response.body.success).toBe(true);
156:         expect(response.body.data.user.email).toBe(testUser.email);
157:         expect(response.body.data.user._id.toString()).toBe(
158:             testUser._id.toString(),
159:         );
160:     });
161:
162:     it("should require authentication", async () => {
163:         const response = await request(app).get("/api/auth/me").expect(401);
164:
165:         expect(response.body.success).toBe(false);
166:     });
167:
168:     it("should reject invalid token", async () => {
169:         const response = await request(app)
170:             .get("/api/auth/me")
171:             .set(getAuthHeader("invalid-token"))
172:             .expect(401);

```

```

173:
174:     expect(response.body.success).toBe(false);
175:   });
176: });
177:
178: describe("PUT /api/auth/profile", () => {
179:   let testUser;
180:   let authToken;
181:
182:   beforeEach(async () => {
183:     testUser = await createTestUser();
184:     authToken = getAuthToken(testUser._id);
185:   });
186:
187:   it("should update user profile", async () => {
188:     const updates = {
189:       firstName: "Updated",
190:       lastName: "Name",
191:     };
192:
193:     const response = await request(app)
194:       .put("/api/auth/profile")
195:       .set(getAuthHeader(authToken))
196:       .send(updates)
197:       .expect(200);
198:
199:     expect(response.body.success).toBe(true);
200:     expect(response.body.data.user.firstName).toBe(updates.firstName);
201:     expect(response.body.data.user.lastName).toBe(updates.lastName);
202:   });
203:
204:   it("should require authentication", async () => {
205:     const response = await request(app)
206:       .put("/api/auth/profile")
207:       .send({ firstName: "Test" })
208:       .expect(401);
209:
210:     expect(response.body.success).toBe(false);
211:   });
212: });
213:
214: describe("PUT /api/auth/change-password", () => {
215:   let testUser;
216:   let authToken;
217:   const currentPassword = "password123";
218:
219:   beforeEach(async () => {
220:     testUser = await createTestUser({ password: currentPassword });
221:     authToken = getAuthToken(testUser._id);
222:   });
223:
224:   it("should change password successfully", async () => {
225:     const response = await request(app)
226:       .put("/api/auth/change-password")
227:       .set(getAuthHeader(authToken))
228:       .send({
229:         currentPassword,
230:         newPassword: "newpassword123",
231:       })
232:       .expect(200);
233:
234:     expect(response.body.success).toBe(true);
235:   });
236:
237:   it("should reject incorrect current password", async () => {
238:     const response = await request(app)
239:       .put("/api/auth/change-password")
240:       .set(getAuthHeader(authToken))
241:       .send({
242:         currentPassword: "wrongpassword",
243:         newPassword: "newpassword123",
244:       })
245:       .expect(401);

```

```

246:
247:     expect(response.body.success).toBe(false);
248:   });
249:
250:   it("should validate new password length", async () => {
251:     const response = await request(app)
252:       .put("/api/auth/change-password")
253:       .set(getAuthHeader(authToken))
254:       .send({
255:         currentPassword,
256:         newPassword: "123", // Too short
257:       })
258:       .expect(400);
259:
260:     expect(response.body.success).toBe(false);
261:   });
262:
263:   it("should require authentication", async () => {
264:     const response = await request(app)
265:       .put("/api/auth/change-password")
266:       .send({
267:         currentPassword,
268:         newPassword: "newpassword123",
269:       })
270:       .expect(401);
271:
272:     expect(response.body.success).toBe(false);
273:   });
274: });
275:
276: describe("POST /api/auth/logout", () => {
277:   let testUser;
278:   let authToken;
279:
280:   beforeEach(async () => {
281:     testUser = await createTestUser();
282:     authToken = getAuthToken(testUser._id);
283:   });
284:
285:   it("should logout successfully", async () => {
286:     const response = await request(app)
287:       .post("/api/auth/logout")
288:       .set(getAuthHeader(authToken))
289:       .expect(200);
290:
291:     expect(response.body.success).toBe(true);
292:   });
293:
294:   it("should require authentication", async () => {
295:     const response = await request(app).post("/api/auth/logout").expect(401);
296:
297:     expect(response.body.success).toBe(false);
298:   });
299: });
300: });

```

■ File: tests/setup.js

```

=====
1: const mongoose = require("mongoose");
2:
3: /**
4:  * Setup test environment before all tests
5:  */
6: beforeEach(async () => {
7:   try {
8:     // Use MongoDB from docker-compose.test.yml or fallback to local
9:     const mongoUri =
10:       process.env.MONGODB_TEST_URI ||
11:       "mongodb://localhost:27017/policy-toggle-service-test";
12:
13:     // Connect to the containerized MongoDB

```

```

14:     await mongoose.connect(mongoUri, {
15:       useNewUrlParser: true,
16:       useUnifiedTopology: true,
17:     });
18:
19:     console.log("? Test database connected");
20:   } catch (error) {
21:     console.error("Failed to connect to test database:", error);
22:     throw error;
23:   }
24: });
25:
26: /**
27:  * Clear database between tests
28:  */
29: afterEach(async () => {
30:   try {
31:     const collections = mongoose.connection.collections;
32:     for (const key in collections) {
33:       await collections[key].deleteMany({});
34:     }
35:   } catch (error) {
36:     console.error("Failed to clear database:", error);
37:   }
38: });
39:
40: /**
41:  * Cleanup after all tests
42:  */
43: afterAll(async () => {
44:   try {
45:     // Close mongoose connection
46:     await mongoose.connection.close();
47:     console.log("? Test database closed");
48:   } catch (error) {
49:     console.error("Failed to close database connection:", error);
50:   }
51: });
52:
53: /**
54:  * Global test timeout
55:  */
56: jest.setTimeout(30000);
57:
58: /**
59:  * Suppress console logs during tests (optional)
60:  */
61: if (process.env.SUPPRESS_LOGS === "true") {
62:   global.console = {
63:     ...console,
64:     log: jest.fn(),
65:     debug: jest.fn(),
66:     info: jest.fn(),
67:     warn: jest.fn(),
68:     error: jest.fn(),
69:   };
70: }

```

■ File: tests\system\featureToggle.flow.test.js

```

1: const request = require("supertest");
2: const app = require("../src/app");
3: const {
4:   createTestAdmin,
5:   createTestUser,
6:   getAuthHeader,
7:   getAuthToken,
8: } = require("../helpers");
9:
10: describe("Feature Toggle System Flow", () => {
11:   let admin;

```

```

12: let adminToken;
13: let user;
14: let userToken;
15: let featureId;
16:
17: beforeEach(async () => {
18:   // Create test users
19:   admin = await createTestAdmin();
20:   adminToken = getAuthToken(admin._id);
21:
22:   user = await createTestUser();
23:   userToken = getAuthToken(user._id);
24: });
25:
26: it("should complete full feature toggle lifecycle", async () => {
27:   // Step 1: Admin creates a new feature toggle
28:   const createResponse = await request(app)
29:     .post("/api/features")
30:     .set(getAuthHeader(adminToken))
31:     .send({
32:       featureName: "premium-features",
33:       description: "Premium features for paid users",
34:       enabled: true,
35:       allowedRoles: ["admin", "user"],
36:       rolloutPercentage: 100,
37:       environments: {
38:         development: { enabled: true },
39:         staging: { enabled: true },
40:         production: { enabled: false },
41:       },
42:     })
43:     .expect(201);
44:
45:   expect(createResponse.body.success).toBe(true);
46:   featureId = createResponse.body.data.feature._id;
47:
48:   // Step 2: User checks enabled features
49:   const enabledResponse = await request(app)
50:     .get("/api/features/enabled")
51:     .set(getAuthHeader(userToken))
52:     .expect(200);
53:
54:   expect(enabledResponse.body.success).toBe(true);
55:   const hasFeature = enabledResponse.body.data.features.some(
56:     (f) => f.featureName === "premium-features",
57:   );
58:   expect(hasFeature).toBe(true);
59:
60:   // Step 3: User checks specific feature access
61:   const checkResponse = await request(app)
62:     .post("/api/features/check")
63:     .set(getAuthHeader(userToken))
64:     .send({ featureName: "premium-features" })
65:     .expect(200);
66:
67:   expect(checkResponse.body.data.enabled).toBe(true);
68:
69:   // Step 4: Admin disables the feature
70:   const toggleResponse = await request(app)
71:     .put(`/api/features/${featureId}/toggle`)
72:     .set(getAuthHeader(adminToken))
73:     .send({ enabled: false })
74:     .expect(200);
75:
76:   expect(toggleResponse.body.data.feature.enabled).toBe(false);
77:
78:   // Step 5: User checks feature access again (should be disabled)
79:   const recheckResponse = await request(app)
80:     .post("/api/features/check")
81:     .set(getAuthHeader(userToken))
82:     .send({ featureName: "premium-features" })
83:     .expect(200);
84:

```

```

85:     expect(recheckResponse.body.data.enabled).toBe(false);
86:
87:     // Step 6: Admin deletes the feature
88:     const deleteResponse = await request(app)
89:       .delete(`/api/features/${featureId}`)
90:       .set(getAuthHeader(adminToken))
91:       .expect(200);
92:
93:     expect(deleteResponse.body.success).toBe(true);
94:
95:     // Step 7: Verify feature is deleted
96:     await request(app)
97:       .get(`/api/features/${featureId}`)
98:       .set(getAuthHeader(adminToken))
99:       .expect(404);
100:   });
101:
102:   it("should enforce role-based feature access", async () => {
103:     // Create admin-only feature
104:     const createResponse = await request(app)
105:       .post("/api/features")
106:       .set(getAuthHeader(adminToken))
107:       .send({
108:         featureName: "admin-dashboard",
109:         description: "Admin-only dashboard",
110:         enabled: true,
111:         allowedRoles: ["admin"],
112:         rolloutPercentage: 100,
113:       });
114:     .expect(201);
115:
116:     featureId = createResponse.body.data.feature._id;
117:
118:     // Admin can access
119:     const adminCheckResponse = await request(app)
120:       .post("/api/features/check")
121:       .set(getAuthHeader(adminToken))
122:       .send({ featureName: "admin-dashboard" });
123:     .expect(200);
124:
125:     expect(adminCheckResponse.body.data.enabled).toBe(true);
126:
127:     // Regular user cannot access
128:     const userCheckResponse = await request(app)
129:       .post("/api/features/check")
130:       .set(getAuthHeader(userToken))
131:       .send({ featureName: "admin-dashboard" });
132:     .expect(200);
133:
134:     expect(userCheckResponse.body.data.enabled).toBe(false);
135:   });
136:
137:   it("should prevent non-admin users from creating features", async () => {
138:     const response = await request(app)
139:       .post("/api/features")
140:       .set(getAuthHeader(userToken))
141:       .send({
142:         featureName: "unauthorized-feature",
143:         description: "Should not be created",
144:         enabled: true,
145:       });
146:     .expect(403);
147:
148:     expect(response.body.success).toBe(false);
149:   });
150:
151:   it("should handle feature statistics correctly", async () => {
152:     // Create multiple features
153:     await request(app)
154:       .post("/api/features")
155:       .set(getAuthHeader(adminToken))
156:       .send({
157:         featureName: "feature-1",

```

```

158:         enabled: true,
159:     });
160:
161:     await request(app)
162:       .post("/api/features")
163:       .set(getAuthHeader(adminToken))
164:       .send({
165:         featureName: "feature-2",
166:         enabled: false,
167:       });
168:
169:     // Get statistics
170:     const statsResponse = await request(app)
171:       .get("/api/features/stats")
172:       .set(getAuthHeader(adminToken))
173:       .expect(200);
174:
175:     expect(statsResponse.body.data.total).toBeGreaterThanOrEqual(2);
176:     expect(statsResponse.body.data.enabled).toBeGreaterThanOrEqual(1);
177:     expect(statsResponse.body.data.disabled).toBeGreaterThanOrEqual(1);
178:   });
179: });

```

■ File: tests/unit/user.model.test.js

```

=====
1: const { User, ROLES } = require("../src/models");
2:
3: describe("User Model", () => {
4:   describe("User Creation", () => {
5:     it("should create a new user successfully", async () => {
6:       const userData = {
7:         email: "test@example.com",
8:         password: "password123",
9:         firstName: "John",
10:        lastName: "Doe",
11:        role: ROLES.USER,
12:      };
13:
14:      const user = await User.create(userData);
15:
16:      expect(user.email).toBe(userData.email);
17:      expect(user.firstName).toBe(userData.firstName);
18:      expect(user.lastName).toBe(userData.lastName);
19:      expect(user.role).toBe(ROLES.USER);
20:      expect(user.isActive).toBe(true);
21:      expect(user.password).not.toBe(userData.password); // Should be hashed
22:    });
23:
24:    it("should hash password before saving", async () => {
25:      const plainPassword = "password123";
26:      const user = await User.create({
27:        email: "test@example.com",
28:        password: plainPassword,
29:        role: ROLES.USER,
30:      });
31:
32:      expect(user.password).not.toBe(plainPassword);
33:      expect(user.password.length).toBeGreaterThan(20); // Bcrypt hash length
34:    });
35:
36:    it("should require email", async () => {
37:      const userData = {
38:        password: "password123",
39:        role: ROLES.USER,
40:      };
41:
42:      await expect(User.create(userData)).rejects.toThrow();
43:    });
44:
45:    it("should require password", async () => {
46:      const userData = {

```



```

47:         email: "test@example.com",
48:         role: ROLES.USER,
49:     };
50:
51:     await expect(User.create(userData)).rejects.toThrow();
52: });
53:
54: it("should enforce unique email", async () => {
55:     const email = "duplicate@example.com";
56:
57:     await User.create({
58:         email,
59:         password: "password123",
60:         role: ROLES.USER,
61:     });
62:
63:     await expect(
64:         User.create({
65:             email,
66:             password: "password456",
67:             role: ROLES.USER,
68:         }),
69:     ).rejects.toThrow();
70: });
71:
72: it("should validate email format", async () => {
73:     const userData = {
74:         email: "invalid-email",
75:         password: "password123",
76:         role: ROLES.USER,
77:     };
78:
79:     await expect(User.create(userData)).rejects.toThrow();
80: });
81:
82: it("should set default role to USER", async () => {
83:     const user = await User.create({
84:         email: "test@example.com",
85:         password: "password123",
86:     });
87:
88:     expect(user.role).toBe(ROLES.USER);
89: });
90: });
91:
92: describe("Password Methods", () => {
93:     let user;
94:     const plainPassword = "password123";
95:
96:     beforeEach(async () => {
97:         user = await User.create({
98:             email: "test@example.com",
99:             password: plainPassword,
100:             role: ROLES.USER,
101:         });
102:     });
103:
104:     it("should compare password correctly", async () => {
105:         const isMatch = await user.comparePassword(plainPassword);
106:         expect(isMatch).toBe(true);
107:     });
108:
109:     it("should reject incorrect password", async () => {
110:         const isMatch = await user.comparePassword("wrongpassword");
111:         expect(isMatch).toBe(false);
112:     });
113:
114:     it("should update password hash when password changes", async () => {
115:         const oldPassword = user.password;
116:         user.password = "newpassword123";
117:         await user.save();
118:
119:         expect(user.password).not.toBe(oldPassword);

```

```

120:     const isMatch = await user.comparePassword("newpassword123");
121:     expect(isMatch).toBe(true);
122:   });
123: });
124:
125: describe("Account Locking", () => {
126:   let user;
127:
128:   beforeEach(async () => {
129:     user = await User.create({
130:       email: "test@example.com",
131:       password: "password123",
132:       role: ROLES.USER,
133:     });
134:   });
135:
136:   it("should not be locked initially", () => {
137:     expect(user.isLocked()).toBe(false);
138:   });
139:
140:   it("should increment login attempts", async () => {
141:     await user.incLoginAttempts();
142:     const updatedUser = await User.findById(user._id);
143:     expect(updatedUser.loginAttempts).toBe(1);
144:   });
145:
146:   it("should lock account after 5 failed attempts", async () => {
147:     for (let i = 0; i < 5; i++) {
148:       await user.incLoginAttempts();
149:       user = await User.findById(user._id);
150:     }
151:
152:     expect(user.isLocked()).toBe(true);
153:     expect(user.lockUntil).toBeDefined();
154:   });
155:
156:   it("should reset login attempts on successful login", async () => {
157:     await user.incLoginAttempts();
158:     await user.incLoginAttempts();
159:     await user.resetLoginAttempts();
160:
161:     const updatedUser = await User.findById(user._id);
162:     expect(updatedUser.loginAttempts).toBe(0);
163:     expect(updatedUser.lockUntil).toBeUndefined();
164:   });
165: });
166:
167: describe("Role Methods", () => {
168:   it("should check if user has specific role", async () => {
169:     const adminUser = await User.create({
170:       email: "admin@example.com",
171:       password: "password123",
172:       role: ROLES.ADMIN,
173:     });
174:
175:     expect(adminUser.hasRole(ROLES.ADMIN)).toBe(true);
176:     expect(adminUser.hasRole(ROLES.USER)).toBe(false);
177:   });
178:
179:   it("should check if user is admin", async () => {
180:     const adminUser = await User.create({
181:       email: "admin@example.com",
182:       password: "password123",
183:       role: ROLES.ADMIN,
184:     });
185:
186:     const regularUser = await User.create({
187:       email: "user@example.com",
188:       password: "password123",
189:       role: ROLES.USER,
190:     });
191:
192:     expect(adminUser.isAdmin()).toBe(true);

```

```

193:         expect(regularUser.isAdmin()).toBe(false);
194:     });
195: });
196:
197: describe("Virtual Properties", () => {
198:     it("should return full name", async () => {
199:         const user = await User.create({
200:             email: "test@example.com",
201:             password: "password123",
202:             firstName: "John",
203:             lastName: "Doe",
204:             role: ROLES.USER,
205:         });
206:
207:         expect(user.fullName).toBe("John Doe");
208:     });
209:
210:     it("should return Anonymous if no name provided", async () => {
211:         const user = await User.create({
212:             email: "test@example.com",
213:             password: "password123",
214:             role: ROLES.USER,
215:         });
216:
217:         expect(user.fullName).toBe("Anonymous");
218:     });
219: });
220:
221: describe("Static Methods", () => {
222:     it("should find user by email", async () => {
223:         const email = "test@example.com";
224:         await User.create({
225:             email,
226:             password: "password123",
227:             role: ROLES.USER,
228:         });
229:
230:         const user = await User.findByEmail(email);
231:         expect(user).toBeDefined();
232:         expect(user.email).toBe(email);
233:     });
234:
235:     it("should return null for non-existent email", async () => {
236:         const user = await User.findByEmail("nonexistent@example.com");
237:         expect(user).toBeNull();
238:     });
239: });
240: });

```
