Backend Architecture

Backend Architecture Document

> **Project:** {{PROJECT_NAME}} **Version:** {{VERSION}} **Date:** {{DATE}}
> **Author:** {{AUTHOR}} **Status:** Draft | In Review | Approved **Reviewers:** {{REVIEWERS}}

Document History

Version	Date	Author	Changes
0.1	{{DATE}}	{{AUTHOR}}	Initial draft

---

1. Architecture Pattern

Pattern: `{{Modular Monolith | Microservices | Monolith | Event-Driven Microservices}}`

Pattern Considered	Pros	Cons	Decision
Monolith	Simple deploy, low latency	Scaling bottleneck, team coupling	`{{Selected/Rejected}}`
Modular Monolith	Organized, single deploy, module isolation	Shared DB risk	`{{Selected/Rejected}}`
Microservices	Independent scaling, team autonomy	Operational complexity	`{{Selected/Rejected}}`

Rationale:

> TODO: 3-5 sentences explaining the decision considering team size, scale requirements, and operational maturity.

---

2. Technology Stack

Layer	Technology	Version	Notes
Runtime	`{{Node.js}}`	`{{20.x LTS}}`	
Framework	`{{NestJS / Express / Fastify / Hono}}`	`{{10.x}}`	
ORM	`{{Prisma / TypeORM / Drizzle}}`	`{{5.x}}`	
Primary DB	`{{PostgreSQL}}`	`{{16.x}}`	Managed: `{{RDS / Supabase}}`
Cache	`{{Redis}}`	`{{7.x}}`	Managed: `{{ElastiCache / Upstash}}`
Queue	`{{BullMQ / SQS / RabbitMQ}}`	`{{5.x}}`	
Search	`{{Elasticsearch / MeiliSearch / Typesense}}`	`{{8.x}}`	Optional
File storage	`{{AWS S3 / Cloudflare R2}}`	API	
Auth	`{{Custom JWT / Auth0 / Supabase Auth}}`		
Logging	`{{Pino / Winston}}`	`{{8.x}}`	→ `{{Datadog / Loki}}`
APM	`{{Datadog / Sentry / Elastic APM}}`		
API docs	`{{Swagger / OpenAPI 3.1}}`	`{{3.1}}`	

---

3. Project Structure

```
src/
├── modules/                # Feature modules (NestJS) / route handlers (Express)
│   ├── users/
│   │   ├── users.module.ts
│   │   ├── users.controller.ts
│   │   ├── users.service.ts
│   │   ├── users.repository.ts
│   │   ├── dto/
│   │   │   ├── create-user.dto.ts
│   │   │   └── update-user.dto.ts
│   │   └── entities/
│   │       └── user.entity.ts
```

```
│   ├── auth/
│   ├── notifications/
│   └── {{FEATURE}}/
├── common/
│   ├── decorators/         # Custom decorators
│   ├── filters/            # Exception filters
│   ├── guards/             # Auth / role guards
│   ├── interceptors/       # Logging, transform interceptors
│   ├── pipes/              # Validation pipes
│   └── middleware/         # Request middleware
├── database/
│   ├── migrations/
│   └── seeds/
├── config/
│   ├── app.config.ts
│   ├── database.config.ts
│   └── redis.config.ts
├── jobs/                   # Background job definitions
└── main.ts                 # Application entry point

test/
├── unit/
├── integration/
└── e2e/
```

---

4. Request Processing Pipeline

```
sequenceDiagram
    participant Client
    participant Gateway as API Gateway / LB
    participant Middleware as Middleware Stack
    participant Guard as Guards
    participant Pipe as Validation Pipe
    participant Controller
    participant Service
    participant Repository
    participant DB as Database
```

```
    Client->>Gateway: HTTP Request
    Gateway->>Middleware: Forward (with tracing headers)
    Middleware->>Middleware: Request ID, CORS, Security Headers, Rate Limit
    Middleware->>Guard: Authenticated request
    Guard->>Guard: JWT verification, Role check
    Guard->>Pipe: Authorized request
    Pipe->>Pipe: Schema validation (Zod/class-validator)
    Pipe->>Controller: Validated DTO
    Controller->>Service: Business method call
    Service->>Repository: Data access call
    Repository->>DB: Query
    DB-->>Repository: Result
    Repository-->>Service: Domain entity
    Service-->>Controller: Response data
    Controller-->>Client: HTTP Response (transformed)
```

---

5. Middleware Stack Configuration

Execution order (applied left-to-right):

Order	Middleware	Purpose	Global?
1	`helmet`	Security headers (CSP, HSTS, etc.)	Yes
2	`cors`	CORS policy enforcement	Yes
3	`request-id`	Inject `X-Request-ID` header	Yes
4	`compression`	gzip response compression	Yes
5	`body-parser`	Parse JSON/urlencoded bodies	Yes
6	`rate-limiter`	IP-based rate limiting (Redis)	Yes
7	`request-logger`	Structured request logging	Yes
8	Route-specific middleware	Auth, validation per route	No

Security headers configured via Helmet:

```
app.use(helmet({
  contentSecurityPolicy: { /* ... */ },
  hsts: { maxAge: 31536000, includeSubDomains: true, preload: true },
```

```
  referrerPolicy: { policy: 'same-origin' },
}));
```

---

6. Authentication & Authorization Flow

```
flowchart TD
    Login["POST /auth/login\n{email, password}"] --> ValidateCreds["Validate credentials\n(bcrypt compare)"]
    ValidateCreds -->|Invalid| Reject["401 Unauthorized"]
    ValidateCreds -->|Valid| IssuePair["Issue token pair\naccess (15m) + refresh (30d)"]
    IssuePair --> StoreRefresh["Store refresh token\n(hashed, Redis or DB)"]
    IssuePair --> ReturnTokens["Return tokens to client"]

    AuthReq["Authenticated Request"] --> ExtractJWT["Extract Bearer token"]
    ExtractJWT --> VerifyJWT["Verify signature + expiry"]
    VerifyJWT -->|Invalid/Expired| RefreshFlow["POST /auth/refresh"]
    VerifyJWT -->|Valid| CheckRoles["Role/permission check"]
    CheckRoles -->|Unauthorized| Forbidden["403 Forbidden"]
    CheckRoles -->|Authorized| Handler["Route Handler"]

    RefreshFlow --> VerifyRefresh["Verify refresh token\n(hash match, not revoked)"]
    VerifyRefresh -->|Invalid| Logout["Force logout → 401"]
    VerifyRefresh -->|Valid| RotateToken["Rotate tokens\n(old token revoked)"]
```

RBAC / ABAC:

- Roles: `{{admin | manager | user | viewer}}`
- Permissions: `{{resource:action}}` e.g. `users:delete`
- Role-permission mapping: `{{database table | config file | code}}`

---

7. Database Access Patterns

Pattern: `{{Repository Pattern}}`

```
// Repository interface — decouples business logic from storage
interface UserRepository {
  findById(id: string): Promise<User | null>;
```

```
  findByEmail(email: string): Promise<User | null>;
  findMany(filters: UserFilters): Promise<PaginatedResult<User>>;
  create(data: CreateUserData): Promise<User>;
  update(id: string, data: UpdateUserData): Promise<User>;
  delete(id: string): Promise<void>;
}
```

Query performance rules:

- All queries accessing > 1000 rows must have paginator
- All filterable fields must have DB indexes (document in migration)
- N+1 queries forbidden — use `include`/JOIN or `dataloader` pattern
- Raw SQL allowed only when ORM cannot express the query efficiently

---

8. Caching Architecture

```
graph LR
    Request --> L1["L1: In-Memory\n(node-cache)"]
    L1 -->|Cache miss| L2["L2: Redis\n(shared, distributed)"]
    L2 -->|Cache miss| DB["Database"]

    DB --> L2
    L2 --> L1
    L1 --> Response
```

Layer	Technology	TTL	What's Cached
L1 (in-process)	`node-cache` / Map	30 sec	Config, feature flags
L2 (distributed)	Redis	Per resource	User sessions, API responses
L3 (CDN edge)	Cloudflare / CloudFront	Per route	Public API responses

Cache-aside pattern (L2):

```
async function getCachedUser(id: string): Promise<User> {
  const cacheKey = `user:${id}`;
  const cached = await redis.get(cacheKey);
  if (cached) return JSON.parse(cached);

  const user = await userRepository.findById(id);
```

```
  await redis.setex(cacheKey, 300, JSON.stringify(user)); // 5 min TTL
  return user;
}
```

Cache invalidation strategy:

- On update/delete: `await redis.del(cacheKey)`
- Pattern-based: `await redis.del('user:*')` (use sparingly — expensive)
- Tag-based: `{{ioredis-tag | custom tagging}}`

---

9. Background Job Processing

Library: `{{BullMQ | Agenda | pg-boss}}` **Queue storage:** `{{Redis | PostgreSQL}}`

Queue	Job Types	Concurrency	Retry Policy
`emails`	Welcome, reset password, notifications	10	3 retries, exp. backoff
`uploads`	Image processing, file conversion	5	2 retries
`sync`	External API sync, data aggregation	3	5 retries
`reports`	PDF generation, exports	2	1 retry

Job schema:

```
interface EmailJob {
  type: 'welcome' | 'password_reset' | 'notification';
  to: string;
  templateId: string;
  data: Record<string, unknown>;
}
```

Monitoring: `{{Bull Board | BullMQ Metrics API}}` — admin UI at `{{/admin/queues}}`

---

10. File Storage & Media Handling

Storage provider: `{{AWS S3 | Cloudflare R2 | MinIO}}` **Bucket naming:** `{{company-project-env}}` (e.g., `alai-app-production`)

Upload flow:

1. Client requests pre-signed URL from API (`POST /uploads/presigned`)
2. API validates file type, size, generates pre-signed URL (expiry: 15 min)
3. Client uploads directly to storage (bypasses API server)
4. Client notifies API of upload completion (`POST /uploads/confirm`)
5. API validates file exists, creates database record, triggers processing job

File size limits:

Type	Max Size
Profile images	5 MB
Documents	25 MB
Videos	500 MB

---

11. Logging & Observability

Logger: `{{Pino}}` — structured JSON logs **Log aggregation:** `{{Datadog / Loki / CloudWatch}}`

Log levels policy:

Level	When to use
`error`	Exceptions, failures requiring attention
`warn`	Unexpected but handled situations
`info`	Significant business events (user created, order placed)
`debug`	Detailed diagnostic info — dev/staging only
`trace`	Verbose request tracing — never in production

Always log:

- Request: method, path, request ID, user ID (hashed), status code, duration
- Errors: full stack trace, request context
- Background jobs: job ID, queue, start/end, duration, outcome

Never log:

- Passwords, tokens, API keys
- Full request/response bodies with PII
- Payment card data

---

12. Configuration Management

Pattern: Typed config module with validation on startup

```
// config/app.config.ts
const schema = z.object({
  NODE_ENV: z.enum(['development', 'staging', 'production']),
  PORT: z.coerce.number().default(4000),
  DATABASE_URL: z.string().url(),
  REDIS_URL: z.string().url(),
  JWT_SECRET: z.string().min(32),
  // ...
});

export const config = schema.parse(process.env);
// App fails FAST at startup if any required variable is missing/invalid
```

Secrets management: `{{HashiCorp Vault | AWS Secrets Manager | Doppler}}` **NO secrets in environment files committed to git.**

---

13. Health Check Endpoints

Endpoint	Type	Checks
`GET /health/live`	Liveness	Process is running
`GET /health/ready`	Readiness	DB connected, Redis connected, app ready
`GET /health/startup`	Startup	Migrations run, config valid

Readiness check response:

```
{
  "status": "ok",
  "checks": {
    "database": { "status": "ok", "latency": 3 },
    "redis": { "status": "ok", "latency": 1 },
    "queue": { "status": "ok", "pendingJobs": 12 }
  },
  "version": "1.2.3",
```

```
  "uptime": 3600
}
```

---

14. Architecture Diagram

```
graph TB
    subgraph "Clients"
        Web["Web App"]
        Mobile["Mobile App"]
    end

    subgraph "Infrastructure"
        LB["Load Balancer\n(Nginx / ALB)"]
        API["API Server\n({{FRAMEWORK}})"]
        Workers["Background Workers\n(BullMQ)"]
    end

    subgraph "Data"
        DB["PostgreSQL\n(Primary + Replicas)"]
        Cache["Redis\n(Cache + Queue)"]
        Storage["Object Storage\n(S3 / R2)"]
    end

    subgraph "Observability"
        Logs["Log Aggregation\n(Datadog / Loki)"]
        APM["APM / Tracing"]
    end

    Web --> LB
    Mobile --> LB
    LB --> API
    API --> DB
    API --> Cache
    API --> Storage
    API --> Cache
    Workers --> Cache
    Workers --> DB
    API --> Logs
```

```
    Workers --> Logs
    API --> APM
```

Approval

Role	Name	Date	Signature
Author			
Backend Lead			
Tech Lead / Architect			
Security Reviewer			

Revision #3
Created 2026-02-24 14:53:23 UTC by John
Updated 2026-05-25 07:33:24 UTC by John