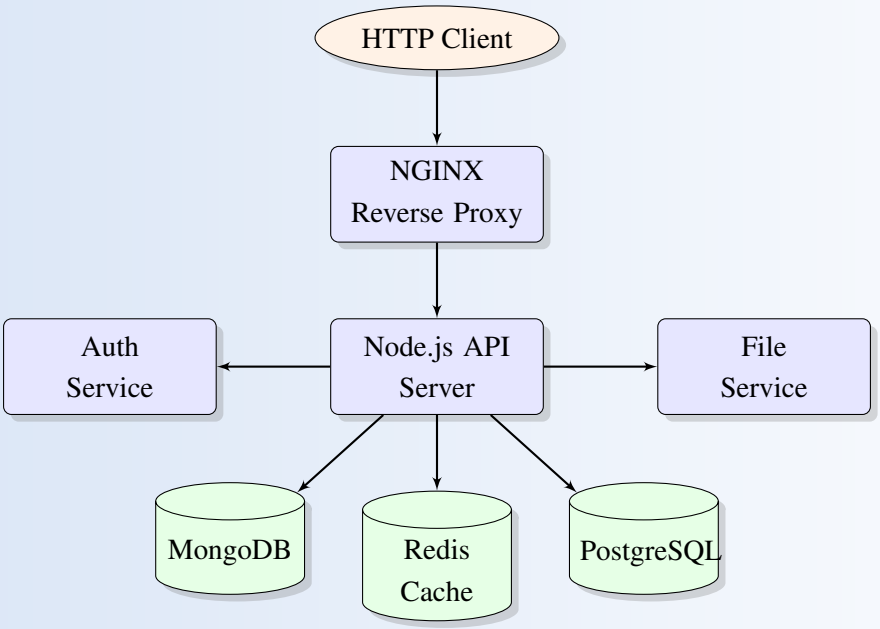
Web API Test Project

Enterprise-Grade API Framework



Version	Status	Classification
1.0.0	Production Ready	Internal

Generated: February 9, 2026

CONTENTS

Executive Summary **2**

1 System Architecture **3**
1.1 High-Level Architecture . . . 3
1.2 Component Interaction Flow . . . 3

2 Project Structure & Organization **4**
2.1 Complete Directory Tree . . . 4
2.2 Key File Descriptions . . . 5

3 API Specification & Documentation **6**
3.1 API Endpoints Reference . . . 6
3.2 Request/Response Examples . . . 7
3.2.1 User Registration Request . . . 7
3.2.2 Successful Registration Response . . . 7
3.2.3 Error Response Example . . . 8

4 Database Configuration & Management **8**
4.1 Environment Configuration . . . 8
4.2 Database Abstraction Layer . . . 9

5 Deployment & Containerization **13**
5.1 Dockerfile (Multi-Stage Production Build) . . . 13
5.2 Docker Compose Configuration . . . 15

6 Go Utilities & Testing Framework **18**
6.1 Port Checker Utility . . . 18

7 Production Deployment Checklist **22**

8 Troubleshooting & Diagnostics **24**

A Appendix A: Environment Variables Reference **25**

B Appendix B: API Error Codes Reference **28**

C Appendix C: Quick Command Reference **29**

EXECUTIVE SUMMARY

Project Overview
The Web API Test Project is a comprehensive, production-ready API framework designed for rapid development, testing, and deployment of scalable web services. This document serves as the complete technical reference for developers, architects, and operations teams.

Key Features:

- **Multi-Database Support**: Seamless switching between MongoDB, MySQL, and PostgreSQL
- **Modern Authentication**: JWT-based auth with refresh tokens and role-based access control
- **Containerized Deployment**: Complete Docker & Docker Compose orchestration
- **Go Utilities**: Performance testing, network validation, and system monitoring tools
- **Enterprise Security**: Built-in protection against common vulnerabilities (OWASP Top 10)
- **Comprehensive Monitoring**: Health checks, metrics collection, and logging

Quick Start Timeline:

1. **5 minutes**: Clone repository and install dependencies
2. **10 minutes**: Configure environment and database connections
3. **2 minutes**: Start services with Docker Compose
4. **1 minute**: Verify deployment with health check endpoints

1 SYSTEM ARCHITECTURE

1.1 High-Level Architecture

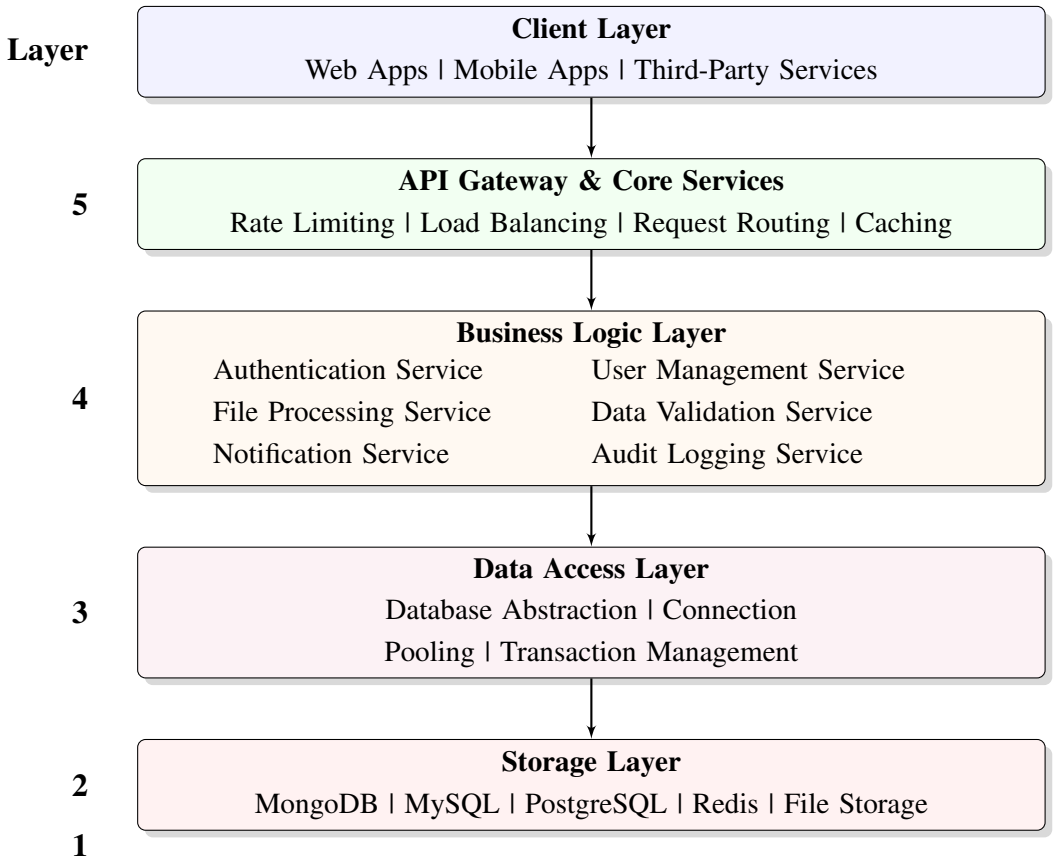


Figure 1: Layered Architecture Overview

1.2 Component Interaction Flow

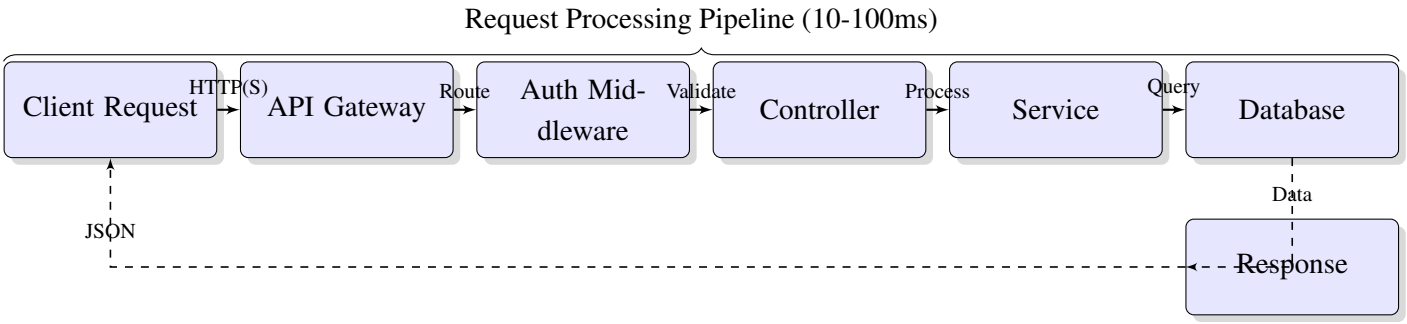


Figure 2: Request Lifecycle Flow

2 PROJECT STRUCTURE & ORGANIZATION

2.1 Complete Directory Tree

Listing 1: Project Directory Structure

```
web_api_test/                          # Root Project Directory
                    .github/                        # GitHub Actions CI/CD
                    .vscode/                        # VS Code settings
                    config/                         # Configuration files
                   environments/                  # Environment-specific configs
                   swagger/                       # OpenAPI specifications
                   nginx/                         # NGINX configurations
                    docs/                          # Documentation
                    src/                           # Application Source Code
                          application/               # Application layer
                                dtos/                 # Data Transfer Objects
                                use-cases/            # Business use cases
                                validators/           # Input validation
                          domain/                   # Domain layer
                                entities/             # Business entities
                                repositories/         # Repository interfaces
                                services/             # Domain services
                          infrastructure/           # Infrastructure layer
                                database/             # Database implementations
                               mongo/               # MongoDB driver
                               mysql/               # MySQL driver
                               postgres/            # PostgreSQL driver
                               db.service.js        # Database abstraction
                                http/                 # HTTP layer
                                      controllers/      # Request handlers
                                      middleware/       # Express middleware
                                      routes/           # Route definitions
                                      serializers/      # Response serializers
                                logging/              # Logging infrastructure
                          interfaces/               # External interfaces
                              web/                  # Web controllers
                              cli/                  # CLI interfaces
                    go/                            # Go utilities and services
                   go.mod                         # Go module definition
                   go.sum                         # Dependency checksum
                          cmd/                      # Go command-line tools
                          pkg/                      # Go libraries
                          testutils/                # Testing utilities
                       config.go                 # Configuration management
                       errors.go                 # Custom error types
                       logger.go                 # Structured logging
                       metrics.go                # Performance metrics
                       retry.go                  # Retry mechanisms
                       testutils.go              # Core utilities
                              examples/             # Usage examples
                    scripts/                       # Build and deployment scripts
                    tests/                         # Test suites
                          unit/                     # Unit tests
                          integration/              # Integration tests
```

```
50                      e2e/                    # End-to-end tests
51                      load/                   # Load tests
52                  uploads/                  # File upload storage
53                      tmp/                    # Temporary uploads
54                      images/                 # Image files
55                      documents/              # Document files
56                  .env.example              # Environment template
57                  .env                      # Environment variables (ignored)
58                  .dockerignore             # Docker ignore rules
59                  .gitignore                # Git ignore rules
60                  package.json              # Node.js dependencies
61                  package-lock.json         # Dependency lockfile
62                  Dockerfile                # Docker build instructions
63                  docker-compose.yml        # Service orchestration
64                  docker-compose.prod.yml   # Production orchestration
65                  index.js                  # Application entry point
66                  app.js                    # Express app configuration
67                  README.md                 # Project documentation
68                  CHANGELOG.md              # Version history
69                  LICENSE                   # License information
```

2.2 Key File Descriptions

File	Purpose
`.env.example`	Template for environment configuration
`docker-compose.yml`	Development environment orchestration
`docker-compose.prod.yml`	Production environment orchestration
`src/infrastructure/database/db.service.js`	Database abstraction layer
`go/testutils/port_checker.go`	Network connectivity validation
`config/swagger/openapi.yaml`	API documentation specification
`scripts/deploy.sh`	Automated deployment script
`tests/load/locustfile.py`	Load testing configuration

Table 1: Critical Project Files

3 API SPECIFICATION & DOCUMENTATION

3.1 API Endpoints Reference

Method	Endpoint	Auth	Description
Authentication Endpoints			
POST	`/api/auth/register`	Public	Register new user account
POST	`/api/auth/login`	Public	Authenticate user credentials
POST	`/api/auth/refresh`	Public	Refresh access token
POST	`/api/auth/logout`	Required	Invalidate user session
GET	`/api/auth/profile`	Required	Get current user profile
PUT	`/api/auth/profile`	Required	Update user profile
User Management Endpoints			
GET	`/api/users`	Admin	List all users (paginated)
GET	`/api/users/:id`	Required	Get user by ID
POST	`/api/users`	Admin	Create new user
PUT	`/api/users/:id`	Required	Update user details
DELETE	`/api/users/:id`	Admin	Delete user account
GET	`/api/users/search`	Required	Search users by criteria
File Operations Endpoints			
POST	`/api/uploads`	Required	Upload single file
POST	`/api/uploads/multiple`	Required	Upload multiple files
GET	`/api/uploads`	Required	List uploaded files
GET	`/api/uploads/:id`	Required	Get file metadata
GET	`/api/uploads/:id/download`	Required	Download file
DELETE	`/api/uploads/:id`	Required	Delete uploaded file
System Endpoints			
GET	`/health`	Public	System health check
GET	`/metrics`	Admin	System metrics (Prometheus)
GET	`/version`	Public	API version information
GET	`/api-docs`	Public	Swagger UI documentation
GET	`/api-docs.json`	Public	OpenAPI specification

3.2 Request/Response Examples

3.2.1 User Registration Request

Listing 2: Registration Request Example

```
POST /api/auth/register HTTP/1.1
Content-Type: application/json
Accept: application/json

{
    "email": "user@example.com",
    "password": "SecurePass123!",
    "firstName": "John",
    "lastName": "Doe",
    "role": "user",
    "metadata": {
        "department": "Engineering",
        "phone": "+1234567890"
    }
}
```

3.2.2 Successful Registration Response

Listing 3: Registration Response Example

```
HTTP/1.1 201 Created
Content-Type: application/json
X-RateLimit-Limit: 100
X-RateLimit-Remaining: 99

{
    "success": true,
    "message": "User registered successfully",
    "data": {
        "user": {
            "id": "507f1f77bcf86cd799439011",
            "email": "user@example.com",
            "firstName": "John",
            "lastName": "Doe",
            "role": "user",
            "status": "active",
            "createdAt": "2024-02-09T10:30:00Z",
            "updatedAt": "2024-02-09T10:30:00Z"
        },
        "tokens": {
            "access": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
            "refresh": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
            "expiresIn": 3600,
            "type": "Bearer"
        }
    },
    "metadata": {
        "timestamp": "2024-02-09T10:30:00Z",
        "version": "1.0.0",
        "requestId": "req_abc123def456"}}
```

3.2.3 Error Response Example

Listing 4: Error Response Example

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "success": false,
    "error": {
        "code": "VALIDATION_ERROR",
        "message": "Invalid input parameters",
        "details": [
            {
                "field": "email",
                "message": "Email must be a valid email address"
            },
            {
                "field": "password",
                "message": "Password must be at least 8 characters"
            }
        ],
        "timestamp": "2024-02-09T10:30:00Z",
        "requestId": "req_abc123def456"
    }
}
```

4 DATABASE CONFIGURATION & MANAGEMENT

4.1 Environment Configuration

Listing 5: Complete Database Configuration (.env)

```
# ============================================
# DATABASE CONFIGURATION
# ============================================

# Database Selection (mongo, mysql, postgres)
DB_TYPE=mongo

# ========== MONGODB CONFIGURATION ==========
MONGO_URI=mongodb://localhost:27017/web_api
MONGO_DB_NAME=web_api
MONGO_USERNAME=admin
MONGO_PASSWORD=securepassword123
MONGO_AUTH_SOURCE=admin
MONGO_REPLICA_SET=rs0
MONGO_READ_PREFERENCE=primary
MONGO_MAX_POOL_SIZE=10
MONGO_SOCKET_TIMEOUT=30000
MONGO_CONNECT_TIMEOUT=30000

```

```
# ========== MYSQL CONFIGURATION ============
MYSQL_HOST=localhost
MYSQL_PORT=3306
MYSQL_DATABASE=web_api
MYSQL_USERNAME=root
MYSQL_PASSWORD=securepassword123
MYSQL_CHARSET=utf8mb4
MYSQL_TIMEZONE=+00:00
MYSQL_CONNECTION_LIMIT=10
MYSQL_QUEUE_LIMIT=0
MYSQL_SSL_ENABLED=false

# ========== POSTGRESQL CONFIGURATION =======
POSTGRES_HOST=localhost
POSTGRES_PORT=5432
POSTGRES_DATABASE=web_api
POSTGRES_USERNAME=postgres
POSTGRES_PASSWORD=securepassword123
POSTGRES_SSL_MODE=disable
POSTGRES_MAX_CONNECTIONS=20
POSTGRES_IDLE_TIMEOUT=30000
POSTGRES_CONNECTION_TIMEOUT=2000

# ========== REDIS CONFIGURATION ============
REDIS_HOST=localhost
REDIS_PORT=6379
REDIS_PASSWORD=
REDIS_DB=0
REDIS_KEY_PREFIX=web_api:
REDIS_TTL=3600

# ========== DATABASE POOL SETTINGS =========
DB_POOL_MIN=2
DB_POOL_MAX=10
DB_IDLE_TIMEOUT=10000
DB_CONNECTION_TIMEOUT=2000
DB_ACQUIRE_TIMEOUT=60000

# ========== MIGRATION SETTINGS =============
DB_MIGRATIONS_TABLE=migrations
DB_MIGRATIONS_PATH=./migrations
DB_SEEDS_PATH=./seeds
```

4.2 Database Abstraction Layer

Listing 6: Enhanced Database Service (src/infrastructure/database/db.service.js)

```
/**
 * Database Abstraction Layer
 * Provides unified interface for multiple database engines
 */
class DatabaseService {
    constructor(config) {
        this.config = config;
```

```
        this.connections = new Map();
        this.strategies = {
            mongo: require('./strategies/mongo.strategy'),
            mysql: require('./strategies/mysql.strategy'),
            postgres: require('./strategies/postgres.strategy')
        };

        this.initialize();
    }

    /**
     * Initialize database connections
     */
    async initialize() {
        try {
            const { type, ...options } = this.config;

            if (!this.strategies[type]) {
                throw new Error(`Unsupported database type: ${type}`);
            }

            this.strategy = new this.strategies[type](options);
            await this.strategy.connect();

            console.log(`    Database connected: ${type.toUpperCase()}`);

            // Register cleanup on process exit
            this.registerCleanup();

        } catch (error) {
            console.error(`    Database connection failed:`, error);
            throw error;
        }
    }

    /**
     * Execute query with connection pooling
     */
    async query(collection, operation, data, options = {}) {
        const startTime = Date.now();

        try {
            const result = await this.strategy.query(
                collection,
                operation,
                data,
                options
            );

            const duration = Date.now() - startTime;

            // Log slow queries
            if (duration > this.config.slowQueryThreshold || 1000) {
                console.warn(`       Slow query detected: ${duration}ms`, {
                    collection,
```

```
                    operation,
                    duration
                });
            }

            return result;

        } catch (error) {
            console.error(`    Query failed:`, {
                collection,
                operation,
                error: error.message
            });

            // Implement retry logic for transient errors
            if (this.isTransientError(error)) {
                return await this.retryQuery(collection, operation, data,
    options);
            }

            throw error;
        }
    }

    /**
     * Transaction support
     */
    async transaction(callback) {
        const session = await this.strategy.startSession();

        try {
            await this.strategy.startTransaction(session);
            const result = await callback(session);
            await this.strategy.commitTransaction(session);
            return result;

        } catch (error) {
            await this.strategy.abortTransaction(session);
            throw error;
        } finally {
            await this.strategy.endSession(session);
        }
    }

    /**
     * Health check
     */
    async healthCheck() {
        try {
            const result = await this.strategy.ping();
            return {
                status: 'healthy',
                database: this.config.type,
                latency: result.latency,
                timestamp: new Date().toISOString()
```

```
            };
        } catch (error) {
            return {
                status: 'unhealthy',
                database: this.config.type,
                error: error.message,
                timestamp: new Date().toISOString()
            };
        }
    }

    /**
     * Graceful shutdown
     */
    async disconnect() {
        try {
            await this.strategy.disconnect();
            console.log('    Database connections closed');
        } catch (error) {
            console.error('    Error closing database connections:', error);
        }
    }

    /**
     * Register cleanup handlers
     */
    registerCleanup() {
        const cleanup = async () => {
            console.log('\  n     Shutting down database connections...');
            await this.disconnect();
            process.exit(0);
        };

        process.on('SIGTERM', cleanup);
        process.on('SIGINT', cleanup);
        process.on('uncaughtException', cleanup);
    }

    // Helper methods...
    isTransientError(error) {
        const transientErrors = [
            'ECONNRESET',
            'ETIMEDOUT',
            'ECONNREFUSED',
            'EPIPE'
        ];
        return transientErrors.some(code => error.code === code);
    }

    async retryQuery(collection, operation, data, options, maxRetries = 3) {
        for (let attempt = 1; attempt <= maxRetries; attempt++) {
            try {
                await new Promise(resolve =>
                    setTimeout(resolve, Math.pow(2, attempt) * 100)
                );
```

```
                return await this.strategy.query(collection, operation, data,
    options);
            } catch (retryError) {
                if (attempt === maxRetries) throw retryError;
            }
        }
    }
}

module.exports = DatabaseService;
```

5 DEPLOYMENT & CONTAINERIZATION

5.1 Dockerfile (Multi-Stage Production Build)

Listing 7: Production Dockerfile

```
# ============================================
# BUILD STAGE
# ============================================
FROM node:20-alpine AS builder

WORKDIR /app

# Install build dependencies
RUN apk add --no-cache \
    python3 \
    make \
    g++ \
    git \
    openssl

# Copy package files
COPY package*.json ./
COPY go/go.mod go/go.sum ./go/

# Install Node.js dependencies
RUN npm ci --only=production --ignore-scripts

# Install Go dependencies
RUN cd go && go mod download

# ============================================
# GO BUILD STAGE
# ============================================
FROM golang:1.21-alpine AS go-builder

WORKDIR /go/src/app

# Copy Go source
COPY go/ ./

# Build Go utilities
```

```
RUN CGO_ENABLED=0 GOOS=linux go build \
    -ldflags="-w -s" \
    -o /go/bin/testutils ./cmd/testutils

# ==========================================
# FINAL STAGE
# ==========================================
FROM node:20-alpine

# Install system dependencies
RUN apk add --no-cache \
    dumb-init \
    tzdata \
    curl \
    bash \
    && cp /usr/share/zoneinfo/UTC /etc/localtime \
    && echo "UTC" > /etc/timezone \
    && rm -rf /var/cache/apk/*

WORKDIR /app

# Create non-root user
RUN addgroup -g 1001 -S nodejs \
    && adduser -S nodejs -u 1001

# Copy built artifacts
COPY --from=builder --chown=nodejs:nodejs /app/node_modules ./node_modules
COPY --from=go-builder --chown=nodejs:nodejs /go/bin/testutils ./go/bin/
COPY --chown=nodejs:nodejs . .

# Create necessary directories
RUN mkdir -p ./uploads ./logs ./tmp \
    && chown -R nodejs:nodejs ./uploads ./logs ./tmp

# Switch to non-root user
USER nodejs

# Health check
HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \
    CMD node -e "require('http').get('http://localhost:${PORT:-3000}/health',
    (r) => { \
        if (r.statusCode === 200) process.exit(0); \
        else process.exit(1); \
    }).on('error', () => process.exit(1))"

# Expose port
EXPOSE 3000

# Start application
CMD ["dumb-init", "node", "index.js"]
```

5.2 Docker Compose Configuration

Listing 8: docker-compose.yml (Development)

```yaml
version: '3.8'

x-logging: &default-logging
  driver: "json-file"
  options:
    max-size: "10m"
    max-file: "3"

services:
  # API Service
  api:
    build:
      context: .
      dockerfile: Dockerfile
      target: builder  # Use builder stage for development
    ports:
      - "3000:3000"
      - "9229:9229"  # Node.js debug port
    environment:
      - NODE_ENV=development
      - DB_TYPE=mongo
      - DEBUG=app:*
    volumes:
      - .:/app
      - /app/node_modules
      - ./uploads:/app/uploads
      - ./logs:/app/logs
    depends_on:
      - mongodb
      - redis
    networks:
      - app-network
    logging: *default-logging
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:3000/health"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 40s

  # MongoDB Service
  mongodb:
    image: mongo:7.0
    ports:
      - "27017:27017"
    environment:
      - MONGO_INITDB_ROOT_USERNAME=admin
      - MONGO_INITDB_ROOT_PASSWORD=admin123
      - MONGO_INITDB_DATABASE=web_api
    volumes:
      - mongodb_data:/data/db
```

```
      - ./config/mongo/init.js:/docker-entrypoint-initdb.d/init.js:ro
    networks:
      - app-network
    command: ["--replSet", "rs0", "--bind_ip_all"]
    logging: *default-logging
    healthcheck:
      test: echo 'db.runCommand("ping").ok' | mongosh localhost:27017/test --
  quiet
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 40s

  # Redis Service
  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"
    command: redis-server --appendonly yes --requirepass redis123
    volumes:
      - redis_data:/data
    networks:
      - app-network
    logging: *default-logging
    healthcheck:
      test: ["CMD", "redis-cli", "--raw", "incr", "ping"]
      interval: 30s
      timeout: 10s
      retries: 3

  # PostgreSQL Service (Optional)
  postgres:
    image: postgres:16-alpine
    ports:
      - "5432:5432"
    environment:
      - POSTGRES_DB=web_api
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres123
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./config/postgres/init.sql:/docker-entrypoint-initdb.d/init.sql:ro
    networks:
      - app-network
    logging: *default-logging
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 30s
      timeout: 10s
      retries: 3

  # NGINX Reverse Proxy
  nginx:
    image: nginx:alpine
    ports:
```

```
      - "80:80"
      - "443:443"
    volumes:
      - ./config/nginx/nginx.conf:/etc/nginx/nginx.conf:ro
      - ./config/nginx/ssl:/etc/nginx/ssl:ro
      - ./logs/nginx:/var/log/nginx
    depends_on:
      - api
    networks:
      - app-network
    logging: *default-logging

  # Adminer (Database Management)
  adminer:
    image: adminer
    ports:
      - "8080:8080"
    environment:
      - ADMINER_DEFAULT_SERVER=mongodb
    networks:
      - app-network
    logging: *default-logging

  # Portainer (Container Management)
  portainer:
    image: portainer/portainer-ce:latest
    ports:
      - "9000:9000"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - portainer_data:/data
    networks:
      - app-network
    logging: *default-logging

networks:
  app-network:
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet: 172.20.0.0/16

volumes:
  mongodb_data:
    driver: local
  redis_data:
    driver: local
  postgres_data:
    driver: local
  portainer_data:
    driver: local
```

6 GO UTILITIES & TESTING FRAMEWORK

6.1 Port Checker Utility

Listing 9: Enhanced Port Checker (go/pkg/utils/port$_c$*hecker.go*)

```go
package utils

import (
    "context"
    "fmt"
    "net"
    "sync"
    "time"
)

// PortStatus represents detailed status of a port check
type PortStatus struct {
    Host        string        `json:"host"`
    Port        int           `json:"port"`
    Protocol    string        `json:"protocol"`
    Open        bool          `json:"open"`
    Latency     time.Duration `json:"latency_ms"`
    Error       string        `json:"error,omitempty"`
    CheckedAt   time.Time     `json:"checked_at"`
    ServiceType string        `json:"service_type,omitempty"`
}

// PortChecker provides comprehensive port checking capabilities
type PortChecker struct {
    Timeout      time.Duration
    MaxWorkers   int
    RetryCount   int
    RetryDelay   time.Duration
}

// NewPortChecker creates a new PortChecker with default settings
func NewPortChecker() *PortChecker {
    return &PortChecker{
        Timeout:    5 * time.Second,
        MaxWorkers: 10,
        RetryCount: 2,
        RetryDelay: 100 * time.Millisecond,
    }
}

// CheckPort performs a single port check with retry logic
func (pc *PortChecker) CheckPort(host string, port int) (PortStatus, error) {
    var lastErr error
    var result PortStatus

    for attempt := 0; attempt <= pc.RetryCount; attempt++ {
        if attempt > 0 {
            time.Sleep(pc.RetryDelay)
        }
```

```
50
51          status := pc.checkSinglePort(host, port, attempt)
52          if status.Open || attempt == pc.RetryCount {
53              result = status
54              break
55          }
56          lastErr = fmt.Errorf(status.Error)
57      }
58
59      // Detect service type
60      if result.Open {
61          result.ServiceType = pc.detectServiceType(host, port)
62      }
63
64      return result, lastErr
65  }
66
67  // checkSinglePort performs a single port check attempt
68  func (pc *PortChecker) checkSinglePort(host string, port int, attempt int)
        PortStatus {
69      start := time.Now()
70      address := fmt.Sprintf("%s:%d", host, port)
71
72      ctx, cancel := context.WithTimeout(context.Background(), pc.Timeout)
73      defer cancel()
74
75      var d net.Dialer
76      conn, err := d.DialContext(ctx, "tcp", address)
77
78      status := PortStatus{
79          Host:      host,
80          Port:      port,
81          Protocol:  "tcp",
82          CheckedAt: time.Now(),
83      }
84
85      if err != nil {
86          status.Open = false
87          status.Error = fmt.Sprintf("Attempt %d: %v", attempt+1, err)
88          status.Latency = time.Since(start)
89          return status
90      }
91
92      defer conn.Close()
93      status.Open = true
94      status.Latency = time.Since(start)
95
96      return status
97  }
98
99  // ScanPortRange scans multiple ports concurrently
100 func (pc *PortChecker) ScanPortRange(host string, startPort, endPort int) (map
        [int]PortStatus, error) {
101     results := make(map[int]PortStatus)
102     var mu sync.Mutex
```

```
    var wg sync.WaitGroup

    semaphore := make(chan struct{}, pc.MaxWorkers)

    for port := startPort; port <= endPort; port++ {
        wg.Add(1)
        go func(p int) {
            defer wg.Done()

            semaphore <- struct{}{}
            defer func() { <-semaphore }()

            status, err := pc.CheckPort(host, p)
            if err != nil {
                status.Error = err.Error()
            }

            mu.Lock()
            results[p] = status
            mu.Unlock()
        }(port)
    }

    wg.Wait()
    return results, nil
}

// CheckCommonPorts checks commonly used service ports
func (pc *PortChecker) CheckCommonPorts(host string) map[int]PortStatus {
    commonPorts := map[int]string{
        21:    "FTP",
        22:    "SSH",
        23:    "Telnet",
        25:    "SMTP",
        53:    "DNS",
        80:    "HTTP",
        443:   "HTTPS",
        3306:  "MySQL",
        5432:  "PostgreSQL",
        27017: "MongoDB",
        6379:  "Redis",
        9200:  "Elasticsearch",
    }

    results := make(map[int]PortStatus)
    var wg sync.WaitGroup
    var mu sync.Mutex

    for port, serviceName := range commonPorts {
        wg.Add(1)
        go func(p int, name string) {
            defer wg.Done()

            status, _ := pc.CheckPort(host, p)
            status.ServiceType = name
```

```
158
159             mu.Lock()
160             results[p] = status
161             mu.Unlock()
162         }(port, serviceName)
163     }
164
165     wg.Wait()
166     return results
167 }
168
169 // detectServiceType attempts to identify the service running on a port
170 func (pc *PortChecker) detectServiceType(host string, port int) string {
171     address := fmt.Sprintf("%s:%d", host, port)
172
173     // Try to read banner
174     conn, err := net.DialTimeout("tcp", address, 2*time.Second)
175     if err != nil {
176         return "unknown"
177     }
178     defer conn.Close()
179
180     // Set read timeout
181     conn.SetReadDeadline(time.Now().Add(2 * time.Second))
182
183     buffer := make([]byte, 1024)
184     n, _ := conn.Read(buffer)
185     if n > 0 {
186         banner := string(buffer[:n])
187
188         // Simple banner detection
189         switch {
190         case contains(banner, "HTTP"):
191             return "HTTP Server"
192         case contains(banner, "SSH"):
193             return "SSH Server"
194         case contains(banner, "FTP"):
195             return "FTP Server"
196         case contains(banner, "SMTP"):
197             return "SMTP Server"
198         case contains(banner, "MySQL"):
199             return "MySQL Database"
200         case contains(banner, "PostgreSQL"):
201             return "PostgreSQL Database"
202         case contains(banner, "MongoDB"):
203             return "MongoDB Database"
204         case contains(banner, "Redis"):
205             return "Redis Server"
206         }
207     }
208
209     // Port-based detection
210     switch port {
211     case 80, 8080, 3000, 5000:
212         return "Web Server"
```

```
    case 443:
        return "HTTPS Server"
    case 22:
        return "SSH Server"
    case 21:
        return "FTP Server"
    case 25, 587, 465:
        return "Email Server"
    case 3306:
        return "MySQL"
    case 5432:
        return "PostgreSQL"
    case 27017:
        return "MongoDB"
    case 6379:
        return "Redis"
    case 9200, 9300:
        return "Elasticsearch"
    default:
        return "unknown"
    }
}

func contains(s, substr string) bool {
    return len(s) >= len(substr) && (s == substr ||
           len(s) > len(substr) && (s[:len(substr)] == substr ||
           contains(s[1:], substr)))
}
```

7 PRODUCTION DEPLOYMENT CHECKLIST

Important Note
Before deploying to production, ensure all items in this checklist are completed and verified.

Category	Checklist Item	Status	Owner
	SSL/TLS certificates configured and validated		
	Environment variables secured (no hardcoded secrets)		
	Firewall rules configured for database access		
	API rate limiting enabled and tested		
	Security headers (CSP, HSTS) properly configured		
	Database backups configured and tested		
	Connection pooling limits set appropriately		
	Indexes created for frequently queried fields		
	Database migration scripts tested		
	Read replicas configured (if applicable)		
	Load balancer configured and tested		
	Auto-scaling rules defined		
	CDN configured for static assets		
	DNS records properly configured		
	Application metrics (Prometheus) configured		
	Log aggregation (ELK/DataDog) set up		
	Alerting rules configured for critical metrics		
	Uptime monitoring enabled		
	Performance baseline established		
	Load testing completed with expected traffic patterns		
	Security penetration testing completed		
	Disaster recovery plan tested		
	Rollback procedure documented and tested		
	Runbooks for common operational tasks created		
	API documentation updated and published		
	Contact information for on-call team updated		

Table 3: Production Deployment Checklist

8 TROUBLESHOOTING & DIAGNOSTICS

Symptom	Possible Cause	Solution
Database connection failures	Network issues, credentials incorrect, database down	• Verify database service is running: `docker ps` • Check connection string in `.env` • Test network connectivity: `telnet host port` • Verify firewall rules
High API response times	Database queries unoptimized, no caching, insufficient resources	• Check slow query logs • Enable Redis caching • Monitor resource usage (CPU, memory) • Implement database indexes
File upload failures	Storage permissions, disk space, file size limits	• Check uploads directory permissions • Verify disk space: `df -h` • Review Multer configuration • Check file size limits in `.env`
Authentication errors	JWT configuration, token expiration, invalid credentials	• Verify JWT secret matches • Check token expiration settings • Validate token generation logic • Test with Postman/curl
Docker container crashes	Memory limits, application errors, dependency issues	• Check container logs: `docker logs <container>` • Monitor memory usage • Verify Docker resource limits • Check for unhandled exceptions
Go utilities not working	Go version mismatch, dependency issues, permission problems	• Verify Go version: `go version` • Update dependencies: `go mod tidy` • Check file permissions • Review build process

Table 4: Troubleshooting Common Issues

A APPENDIX A: ENVIRONMENT VARIABLES REFERENCE

Listing 10: Complete Environment Variables Specification

```
# ==========================================
# APPLICATION CONFIGURATION
# ==========================================
NODE_ENV=development                    # development|production|test
PORT=3000                               # Application port
HOST=0.0.0.0                            # Bind address
APP_NAME=Web API Test Project           # Application name
APP_VERSION=1.0.0                       # Version
BASE_URL=http://localhost:3000          # Base URL

# ==========================================
# DATABASE CONFIGURATION
# ==========================================
DB_TYPE=mongo                           # mongo|mysql|postgres
DB_POOL_MIN=2                           # Minimum connections
DB_POOL_MAX=10                          # Maximum connections
DB_IDLE_TIMEOUT=10000                   # Connection idle timeout (ms)
DB_CONNECTION_TIMEOUT=2000              # Connection timeout (ms)

# MongoDB
MONGO_URI=mongodb://localhost:27017/web_api
MONGO_DB=web_api
MONGO_USER=admin
MONGO_PASSWORD=password

# MySQL
MYSQL_HOST=localhost
MYSQL_PORT=3306
MYSQL_DATABASE=web_api
MYSQL_USER=root
MYSQL_PASSWORD=password
MYSQL_CHARSET=utf8mb4
MYSQL_TIMEZONE=+00:00

# PostgreSQL
POSTGRES_HOST=localhost
POSTGRES_PORT=5432
POSTGRES_DATABASE=web_api
POSTGRES_USER=postgres
POSTGRES_PASSWORD=password
POSTGRES_SSL=false




# ==========================================
```

```
# REDIS CONFIGURATION
# ============================================
REDIS_HOST=localhost
REDIS_PORT=6379
REDIS_PASSWORD=
REDIS_DB=0
REDIS_KEY_PREFIX=web_api:
REDIS_TTL=3600                          # Default TTL in seconds

# ============================================
# JWT CONFIGURATION
# ============================================
JWT_SECRET=your-super-secret-jwt-key-change-in-production
JWT_ALGORITHM=HS256
JWT_EXPIRES_IN=15m                      # Access token expiration
JWT_REFRESH_SECRET=your-refresh-secret
JWT_REFRESH_EXPIRES_IN=7d               # Refresh token expiration

# ============================================
# SECURITY CONFIGURATION
# ============================================
CORS_ORIGIN=*
RATE_LIMIT_WINDOW=15                    # Minutes
RATE_LIMIT_MAX=100                      # Requests per window
BCRYPT_SALT_ROUNDS=12                   # Password hashing rounds
SESSION_SECRET=your-session-secret
CSRF_SECRET=your-csrf-secret

# ============================================
# FILE UPLOAD CONFIGURATION
# ============================================
UPLOAD_PATH=./uploads
MAX_FILE_SIZE=10485760                  # 10MB in bytes
ALLOWED_FILE_TYPES=jpg,jpeg,png,gif,pdf,doc,docx,txt
MAX_FILES_PER_REQUEST=5
ENABLE_FILE_VALIDATION=true

# ============================================
# LOGGING CONFIGURATION
# ============================================
LOG_LEVEL=info                          # error|warn|info|debug
LOG_FORMAT=json                         # json|text
LOG_FILE=./logs/app.log
LOG_MAX_SIZE=10485760                   # 10MB
LOG_MAX_FILES=5
ENABLE_ACCESS_LOGS=true

# ============================================
# EMAIL CONFIGURATION (Optional)
# ============================================
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_USER=your-email@gmail.com
SMTP_PASSWORD=your-password
EMAIL_FROM=noreply@example.com
```

```
EMAIL_TEMPLATES_PATH=./email-templates

# ==============================================
# THIRD-PARTY INTEGRATIONS
# ==============================================
SENTRY_DSN=                                  # Sentry error tracking
NEW_RELIC_LICENSE_KEY=                       # New Relic monitoring
AWS_ACCESS_KEY_ID=
AWS_SECRET_ACCESS_KEY=
AWS_REGION=us-east-1
GOOGLE_CLIENT_ID=
GOOGLE_CLIENT_SECRET=

# ==============================================
# FEATURE FLAGS
# ==============================================
ENABLE_CACHE=true
ENABLE_COMPRESSION=true
ENABLE_HELMET=true
ENABLE_RATE_LIMIT=true
ENABLE_SWAGGER=true
ENABLE_METRICS=true
```

B APPENDIX B: API ERROR CODES REFERENCE

Error Code	HTTP Status	Description	Example Scenario
VALIDATION_ERROR	400	Input validation failed	Invalid email format, missing required fields
AUTHENTICATION_FAILED	401	Invalid credentials	Wrong password, expired token
TOKEN_EXPIRED	401	JWT token expired	Access token past expiration time
INSUFFICIENT_PERMISSIONS	403	User lacks required permissions	User tries to access admin endpoint
RESOURCE_NOT_FOUND	404	Requested resource doesn't exist	User ID not found in database
METHOD_NOT_ALLOWED	405	HTTP method not supported for endpoint	Using GET on POST-only endpoint
CONFLICT	409	Resource conflict	Duplicate email during registration
PAYLOAD_TOO_LARGE	413	Request body exceeds limits	File upload exceeds size limit
UNSUPPORTED_MEDIA_TYPE	415	Invalid Content-Type header	Sending XML when JSON expected
RATE_LIMIT_EXCEEDED	429	Too many requests	User exceeds API rate limits
INTERNAL_SERVER_ERROR	500	Unexpected server error	Database connection failure
SERVICE_UNAVAILABLE	503	Service temporarily unavailable	Maintenance mode, database down
DATABASE_CONNECTION_ERROR	503	Cannot connect to database	Database server unreachable

Table 5: Complete API Error Codes Reference

C APPENDIX C: QUICK COMMAND REFERENCE

Command	Purpose
`npm start`	Start the application
`npm run dev`	Start with hot-reload (development)
`npm test`	Run all tests
`npm run test:coverage`	Run tests with coverage report
`npm run lint`	Run ESLint code analysis
`npm run format`	Format code with Prettier
`docker-compose up`	Start all services (development)
`docker-compose -f docker-compose.prod.yml up`	Start production services
`docker-compose down`	Stop all services
`docker-compose logs -f api`	Follow API service logs
`curl http://localhost:3000/health`	Check API health
`curl http://localhost:3000/metrics`	View Prometheus metrics
`go test ./...`	Run all Go tests
`go run go/cmd/testutils/main.go`	Run Go test utilities
`node scripts/create-user.js`	Create test user via script
`node scripts/seed-database.js`	Seed database with test data
`npm run migration:run`	Run database migrations
`npm run migration:revert`	Revert last migration
`npm run generate:docs`	Generate API documentation
`npm run security:audit`	Run security audit

Table 6: Useful Development & Deployment Commands