Documentação das APIs – Equipe B (Auth & Profile Services)

UberHub Mentorias – MVP 2025

Capa Institucional

Instituição: UberHub – Curso de Sistemas de Informação
Projeto: Arquitetura de Microsserviços – Aplicativo UberHub Mentorias
Equipe: B – Catálogo e Identidade

Integrantes:

- Artur Machado Soares
- Augusto Cesar Rezende
- Eliezer Ferreira Rocha
- Lucio da Costa Junior
- Victor Hugo Silva

Versão: 1.0
Data: Outubro/2025

Resumo Executivo

O aplicativo UberHub Mentorias conecta mentores e mentorados em um ambiente digital baseado em princípios de microsserviços, segurança de APIs, escalabilidade e documentação OpenAPI.

A Equipe B (Catálogo e Identidade) é responsável pelos seguintes componentes fundamentais:

- **Auth Service:** Gerenciamento de autenticação, tokens JWT e notificações FCM
- **Profile Service:** Gestão de perfis de mentores e taxonomia de especialização

Esses serviços constituem a camada de identidade e catálogo da plataforma, servindo de base para os módulos de matchmaking e administração.

Sumário

---

1. Visão Geral da Equipe B

A Equipe B é responsável pelos componentes de autenticação e catálogo de mentores da plataforma UberHub Mentorias.

1.1 Serviços Desenvolvidos

Serviço	Responsabilidade	Integrações
Auth Service	Cadastro de usuários, autenticação JWT, gerenciamento de roles e status, tokens FCM	Admin Service
Profile Service	Perfis de mentores, taxonomia de tags, workflow de aprovação	Matchmaking Service, Admin Service

1.2 Stack Tecnológica

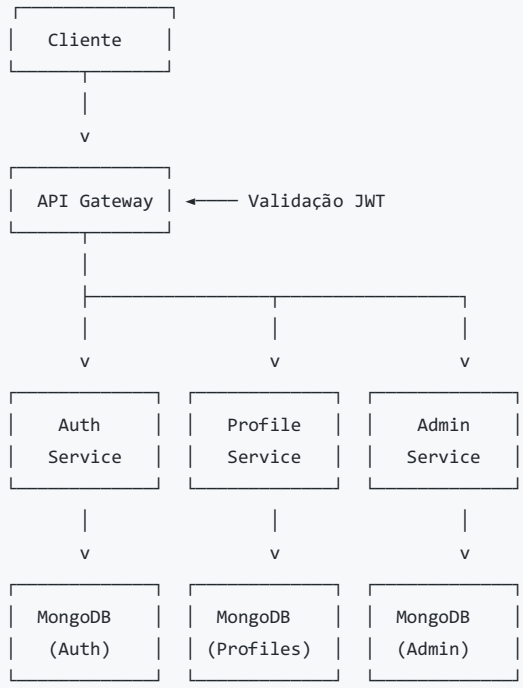
- **Framework:** Spring Boot 3.x
- **Banco de Dados:** MongoDB 6.x
- **API Gateway:** Spring Cloud Gateway

- **Segurança:** Spring Security com JWT
- **Documentação:** OpenAPI 3.0 / Springdoc

---

2. Arquitetura e Integração

2.1 Diagrama de Arquitetura



2.2 Fluxo de Operação

1. **Autenticação:** Usuário registra-se e autentica-se via Auth Service
2. **Validação:** API Gateway valida tokens JWT antes do roteamento
3. **Perfil:** Mentor cria perfil através do Profile Service
4. **Aprovação:** Admin Service aprova perfis pendentes
5. **Matchmaking:** Matchmaking Service consulta perfis aprovados
6. **Notificações:** Auth Service gerencia tokens FCM para push notifications

2.3 Padrões de Comunicação

- Comunicação síncrona via REST/HTTP
- Autenticação via Bearer Token (JWT)
- Validação centralizada no API Gateway
- Isolamento de dados por serviço

---

3. Auth Service

3.1 Especificação

URL Base: `http://localhost:8081/api/v1`
Banco de Dados: MongoDB
Coleções: `users`, `fcmTokens`

3.2 Funcionalidades

- Registro e autenticação de usuários
- Geração e renovação de tokens JWT
- Gerenciamento de roles (MENTOR, MENTEE, ADMIN)
- Controle de status (ACTIVE, SUSPENDED)
- Gerenciamento de tokens FCM para notificações push

3.3 Autenticação

Requisições autenticadas devem incluir o cabeçalho:

```
Authorization: Bearer {access_token}
```

3.4 Endpoints

Método	Endpoint	Descrição	Autenticação
POST	`/auth/register`	Registra novo usuário	Pública
POST	`/auth/login`	Autentica usuário e retorna JWT	Pública
POST	`/auth/refresh`	Renova access token	Requer refresh token
GET	`/users/me`	Retorna dados do usuário autenticado	Requer autenticação
PUT	`/users/me/fcm-token`	Atualiza token FCM do usuário	Requer autenticação
GET	`/users`	Lista todos os usuários	Requer role ADMIN
PUT	`/users/{id}/status`	Atualiza status do usuário	Requer role ADMIN
GET	`/users/{userId}/fcm-tokens`	Lista tokens FCM do usuário	Uso interno

3.5 Modelos de Dados

User

```
{
  "id": "string",
  "name": "string",
  "email": "string",
  "password": "string (hashed)",
  "role": "MENTOR | MENTEE | ADMIN",
  "status": "ACTIVE | SUSPENDED",
  "createdAt": "datetime",
  "updatedAt": "datetime"
}
```

FCM Token

```
{
  "id": "string",
  "userId": "string",
  "token": "string",
  "deviceId": "string",
  "createdAt": "datetime"
}
```

3.6 Exemplos de Requisições

Registro de Usuário

```
POST /api/v1/auth/register
Content-Type: application/json

{
  "name": "João Silva",
  "email": "joao.silva@email.com",
  "password": "senhaSegura123",
  "role": "MENTOR"
}
```

Resposta (201 Created):

```
{
  "id": "507f1f77bcf86cd799439011",
  "name": "João Silva",
  "email": "joao.silva@email.com",
  "role": "MENTOR",
  "status": "ACTIVE",
  "createdAt": "2025-10-31T14:30:00Z"
}
```

Login

```
POST /api/v1/auth/login
Content-Type: application/json

{
  "email": "joao.silva@email.com",
  "password": "senhaSegura123"
}
```

Resposta (200 OK):

```
{
  "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "expiresIn": 3600,
  "tokenType": "Bearer"
}
```

---

4. Profile Service

4.1 Especificação

URL Base: `http://localhost:8082/api/v1`
Banco de Dados: MongoDB
Coleções: `profiles`, `tags`

4.2 Funcionalidades

- Gerenciamento de perfis de mentores
- CRUD de tags de especialização
- Workflow de aprovação (PENDING → APPROVED)
- Consultas filtradas por área e status
- Integração com Matchmaking Service

4.3 Endpoints

Perfis

Método	Endpoint	Descrição	Autenticação

Método	Endpoint	Descrição	Autenticação
POST	`/profiles/mentor`	Cria perfil de mentor	Requer autenticação
PUT	`/profiles/mentor`	Atualiza perfil do mentor autenticado	Requer autenticação
GET	`/profiles/mentor/{mentorId}`	Retorna perfil específico	Requer autenticação
GET	`/profiles/mentors`	Lista mentores com filtros	Requer autenticação
GET	`/profiles/mentors/pending`	Lista mentores pendentes de aprovação	Requer role ADMIN
PUT	`/profiles/mentor/{mentorId}/status`	Atualiza status do perfil	Requer role ADMIN

Tags

Método	Endpoint	Descrição	Autenticação
GET	`/tags`	Lista todas as tags	Requer autenticação
POST	`/tags`	Cria nova tag	Requer role ADMIN
PUT	`/tags/{tagId}`	Atualiza tag existente	Requer role ADMIN
DELETE	`/tags/{tagId}`	Remove tag	Requer role ADMIN

4.4 Modelos de Dados

Profile

```
{
  "id": "string",
  "mentorId": "string",
  "miniBio": "string",
  "areas": ["string"],
  "schedulingLink": "string (URL)",
  "status": "PENDING | APPROVED | REJECTED",
  "createdAt": "datetime",
  "updatedAt": "datetime",
  "approvedAt": "datetime",
  "approvedBy": "string"
}
```

Tag

```
{
  "id": "string",
  "name": "string",
  "category": "string",
  "description": "string",
  "createdAt": "datetime",
  "updatedAt": "datetime"
}
```

4.5 Exemplos de Requisições

Criar Perfil de Mentor

```
POST /api/v1/profiles/mentor
Authorization: Bearer {token}
Content-Type: application/json

{
  "mentorId": "507f1f77bcf86cd799439011",
  "miniBio": "Engenheiro de Software com 10 anos de experiência em desenvolvimento backend, especializado em arquitetura de microsse
  "areas": ["Backend", "Cloud Computing", "Microsserviços"],
  "schedulingLink": "https://calendly.com/joao-silva"
}
```

Resposta (201 Created):

```
{
  "id": "507f191e810c19729de860ea",
  "mentorId": "507f1f77bcf86cd799439011",
  "miniBio": "Engenheiro de Software com 10 anos de experiência...",
  "areas": ["Backend", "Cloud Computing", "Microsserviços"],
  "schedulingLink": "https://calendly.com/joao-silva",
  "status": "PENDING",
  "createdAt": "2025-10-31T14:35:00Z"
}
```

Listar Mentores Aprovados

```
GET /api/v1/profiles/mentors?status=APPROVED&area=Backend&limit=10&offset=0
Authorization: Bearer {token}
```

Resposta (200 OK):

```
{
  "data": [
    {
      "id": "507f191e810c19729de860ea",
      "mentorId": "507f1f77bcf86cd799439011",
      "miniBio": "Engenheiro de Software com 10 anos...",
      "areas": ["Backend", "Cloud Computing"],
      "schedulingLink": "https://calendly.com/joao-silva",
      "status": "APPROVED",
      "approvedAt": "2025-10-31T15:00:00Z"
    }
  ],
  "pagination": {
    "total": 45,
    "limit": 10,
    "offset": 0,
    "hasNext": true
  }
}
```

Criar Tag

```
POST /api/v1/tags
Authorization: Bearer {token}
Content-Type: application/json

{
  "name": "Spring Boot",
  "category": "Framework",
  "description": "Framework Java para desenvolvimento de aplicações empresariais"
}
```

Resposta (201 Created):

```
{
  "id": "507f191e810c19729de860eb",
  "name": "Spring Boot",
  "category": "Framework",
  "description": "Framework Java para desenvolvimento de aplicações empresariais",
  "createdAt": "2025-10-31T14:40:00Z"
}
```

---

5. Banco de Dados

5.1 Estratégia de Persistência

Cada microsserviço mantém seu próprio banco de dados MongoDB, garantindo:

- Isolamento de dados e independência operacional
- Escalabilidade horizontal independente
- Resiliência a falhas isoladas
- Flexibilidade para evolução de schemas

5.2 Estrutura de Coleções

Auth Service

Coleção: users

```
{
  _id: ObjectId,
  name: String,
  email: String (unique, indexed),
  password: String (bcrypt hashed),
  role: String (enum),
  status: String (enum, indexed),
  createdAt: ISODate,
  updatedAt: ISODate
}
```

Coleção: fcmTokens

```
{
  _id: ObjectId,
  userId: ObjectId (indexed),
  token: String (unique, indexed),
  deviceId: String,
  createdAt: ISODate,
  lastUsedAt: ISODate
}
```

Profile Service

Coleção: profiles

```
{
  _id: ObjectId,
  mentorId: ObjectId (unique, indexed),
  miniBio: String,
  areas: [String] (indexed),
  schedulingLink: String,
  status: String (enum, indexed),
  createdAt: ISODate,
  updatedAt: ISODate,
  approvedAt: ISODate,
  approvedBy: ObjectId
}
```

Coleção: tags

```
{
  _id: ObjectId,
  name: String (unique, indexed),
  category: String (indexed),
  description: String,
  createdAt: ISODate,
  updatedAt: ISODate
}
```

5.3 Índices

Auth Service

```
// users
db.users.createIndex({ email: 1 }, { unique: true });
db.users.createIndex({ status: 1 });
db.users.createIndex({ role: 1 });
db.users.createIndex({ createdAt: -1 });

// fcmTokens
db.fcmTokens.createIndex({ userId: 1 });
db.fcmTokens.createIndex({ token: 1 }, { unique: true });
db.fcmTokens.createIndex({ createdAt: -1 });
```

Profile Service

```
// profiles
db.profiles.createIndex({ mentorId: 1 }, { unique: true });
db.profiles.createIndex({ status: 1 });
db.profiles.createIndex({ areas: 1 });
db.profiles.createIndex({ createdAt: -1 });
db.profiles.createIndex({ status: 1, areas: 1 }); // compound index

// tags
db.tags.createIndex({ name: 1 }, { unique: true });
db.tags.createIndex({ category: 1 });
```

---

6. Comunicação entre Serviços

6.1 Padrão de Integração

A arquitetura utiliza comunicação REST síncrona através do Spring Cloud Gateway, implementando os seguintes padrões:

- **Service Discovery**: Registro e descoberta de serviços
- **Load Balancing**: Distribuição de carga entre instâncias
- **Circuit Breaker**: Proteção contra falhas em cascata

- **Retry Pattern:** Tentativas automáticas em caso de falha temporária

6.2 Cliente HTTP com Feign

```
@FeignClient(
    name = "profile-service",
    url = "${services.profile.url}",
    configuration = FeignConfig.class
)
public interface ProfileServiceClient {

    @GetMapping("/api/v1/profiles/mentor/{mentorId}")
    ResponseEntity<MentorProfileDTO> getMentorProfile(
        @PathVariable("mentorId") String mentorId,
        @RequestHeader("Authorization") String token
    );

    @GetMapping("/api/v1/profiles/mentors")
    ResponseEntity<List<MentorProfileDTO>> getApprovedMentors(
        @RequestParam(required = false) String area,
        @RequestParam(defaultValue = "APPROVED") String status,
        @RequestHeader("Authorization") String token
    );
}
```

6.3 Fluxo de Autenticação

```
1. Cliente → POST /auth/login → Auth Service
2. Auth Service → Retorna JWT → Cliente
3. Cliente → GET /profiles/mentors (com JWT) → Gateway
4. Gateway → Valida JWT → Roteia para Profile Service
5. Profile Service → Processa requisição → Retorna dados
6. Gateway → Retorna resposta → Cliente
```

6.4 Propagação de Contexto

O contexto de autenticação é propagado entre serviços através de:

- Header `Authorization` com Bearer Token
- Header `X-User-Id` com identificador do usuário
- Header `X-User-Role` com role do usuário

---

7. Padrões e Boas Práticas

7.1 Segurança

Autenticação e Autorização

- Implementação de Spring Security com JWT
- Tokens com expiração de 1 hora (access) e 7 dias (refresh)
- Algoritmo de assinatura: HMAC SHA-256
- Validação de tokens centralizada no Gateway
- Suporte a múltiplos roles (MENTOR, MENTEE, ADMIN)

Proteção de Dados

- Senhas hasheadas com BCrypt (cost factor: 12)
- Dados sensíveis não retornados em respostas de API
- HTTPS obrigatório em produção
- CORS configurado com whitelist de origens

Rate Limiting

- 100 requisições por minuto por IP
- 1000 requisições por hora por usuário autenticado
- Limitações específicas para endpoints sensíveis

7.2 Versionamento

- Versionamento via URL: `/api/v1/`, `/api/v2/`
- Manutenção de versões anteriores por no mínimo 6 meses
- Documentação clara de breaking changes
- Header `X-API-Version` para negociação de versão

7.3 Tratamento de Erros

Estrutura Padrão de Erro

```
{
  "timestamp": "2025-10-31T14:45:00Z",
  "status": 400,
  "error": "Bad Request",
  "message": "Validation failed for field 'email'",
  "path": "/api/v1/auth/register",
  "errors": [
    {
      "field": "email",
      "message": "Email já cadastrado no sistema"
    }
  ]
}
```

Códigos de Status HTTP

- 200 OK: Requisição bem-sucedida
- 201 Created: Recurso criado com sucesso
- 400 Bad Request: Erro de validação
- 401 Unauthorized: Autenticação necessária
- 403 Forbidden: Permissão negada
- 404 Not Found: Recurso não encontrado
- 409 Conflict: Conflito de dados (ex: email duplicado)
- 500 Internal Server Error: Erro interno do servidor

7.4 Logging

Níveis de Log

- ERROR: Erros críticos que exigem atenção imediata
- WARN: Situações anormais que não impedem operação
- INFO: Eventos importantes do sistema
- DEBUG: Informações detalhadas para debugging

Informações Registradas

- Timestamp e nível de log
- Identificador de requisição (correlation ID)
- Identificador de usuário (quando autenticado)
- Endpoint e método HTTP
- Tempo de resposta
- Status code da resposta

7.5 Documentação

OpenAPI/Swagger

- Documentação gerada automaticamente via Springdoc
- Disponível em `/api/docs` e `/api/swagger-ui.html`
- Schemas de requisição e resposta detalhados
- Exemplos de uso para cada endpoint
- Códigos de status e descrições de erro

Documentação de Código

- Javadoc para classes e métodos públicos
- README.md em cada repositório
- Diagramas de arquitetura atualizados
- Guias de configuração e deployment

7.6 Testes

Cobertura Mínima

- Testes unitários: 80% de cobertura
- Testes de integração para endpoints críticos
- Testes de contrato entre serviços

Ferramentas

- JUnit 5 para testes unitários
- MockMvc para testes de API
- Testcontainers para testes de integração com MongoDB
- Postman/Newman para testes de API automatizados

---

8. Conclusão

A Equipe B entrega componentes fundamentais para a plataforma UberHub Mentorias, estabelecendo uma base sólida de identidade e catálogo.

8.1 Entregas

Auth Service

- Sistema completo de autenticação e autorização
- Gerenciamento de usuários e roles
- Suporte a notificações push via FCM
- Segurança com JWT e refresh tokens

Profile Service

- Gestão de perfis de mentores
- Sistema de tags de especialização
- Workflow de aprovação de mentores
- APIs otimizadas para matchmaking

8.2 Integração com Ecossistema

Os serviços desenvolvidos integram-se com:

- **Admin Service:** Gestão de usuários e aprovação de mentores
- **Matchmaking Service:** Consulta de mentores e tags
- **Notification Service:** Envio de notificações push

8.3 Próximos Passos

- Implementação de cache distribuído (Redis)
- Métricas e observabilidade (Prometheus/Grafana)
- CI/CD completo com testes automatizados
- Deployment em ambiente Kubernetes

---

9. Referências Técnicas

9.1 Frameworks e Bibliotecas

Spring Framework

- Spring Boot 3.x - Framework base para microsserviços
- Spring Security 6.x - Segurança e autenticação
- Spring Data MongoDB - Integração com MongoDB
- Spring Cloud Gateway - API Gateway

Segurança

- JWT (RFC 7519) - JSON Web Tokens
- BCrypt - Hashing de senhas

Comunicação

- OpenFeign - Cliente HTTP declarativo
- WebClient - Cliente HTTP reativo

Documentação

- OpenAPI 3.0 - Especificação de APIs
- Springdoc OpenAPI - Geração de documentação

9.2 Banco de Dados

- MongoDB 6.x - Banco de dados NoSQL
- MongoDB Indexes - Otimização de queries

9.3 Padrões e Arquitetura

- Microservices Pattern - Padrões de microsserviços
- 12 Factor App - Metodologia para aplicações cloud-native
- REST API Design - Melhores práticas REST

10. Anexos

10.1 Repositórios

- **Auth Service:** github.com/uberhub/auth-service
- **Profile Service:** github.com/uberhub/profile-service
- **API Gateway:** github.com/uberhub/api-gateway

10.2 Equipe

Nome	Função	E-mail
Artur Machado Soares	Desenvolvedor Backend	artur.soares@uberhub.edu.br
Augusto Cesar Rezende	Desenvolvedor Backend	augusto.rezende@uberhub.edu.br
Eliezer Ferreira Rocha	Desenvolvedor Full Stack	eliezer.rocha@uberhub.edu.br
Lucio da Costa Junior	Analista de Sistemas	lucio.junior@uberhub.edu.br
Victor Hugo Silva	Arquiteto de Software	victor.silva@uberhub.edu.br

10.3 Ambientes

Desenvolvimento

- Auth Service: http://localhost:8081
- Profile Service: http://localhost:8082
- API Gateway: http://localhost:8080

Homologação

- API Gateway: https://staging-api.uberhub.edu.br

Produção

- API Gateway: https://api.uberhub.edu.br

10.4 Suporte

- **E-mail:** equipe-b@uberhub.edu.br
- **Slack:** #equipe-b-catalogo-identidade
- **Documentação:** https://docs.uberhub.edu.br

UberHub Mentorias – MVP 2025
Equipe B – Catálogo e Identidade
Versão 1.0 – Outubro/2025