ChatGPT

Compendio de Instrucciones para Desarrollador A

Leer junto con `Taller 2.pdf` *y* `Taller 2 - Rubrica.xlsx`.

---

1. Contexto y Alcance

Responsable de los microservicios de:

- **Autenticación**
- **Usuarios**
- **Listas de reproducción**
- **Envío de correos**
- **API Gateway**

Objetivos:

- Entregar todos los componentes con **máxima calidad**, minimizando riesgos de descuentos.
- Preparar mocks y contratos claros para que el Desarrollador B se acople sin bloqueos.
- Optimizar el esfuerzo mediante scaffolding con LLMs y buenas prácticas de validación.

---

2. Dependencias y Mocks para Integración con B

Servicio	Depende de	Qué dejar mockeado para B
AuthService	UsersService	- Publica eventos `user.updated` en RabbitMQ
UsersService	Ninguno	- N/A
PlaylistService	UsersService, VideosService	- gRPC/HTTP stub de Videos (`GetVideo`)
EmailService	BillingService, MonitoringService	- Consumidor RabbitMQ que suscribe `invoice.paid` (aio-pika) en vez de gRPC

- Implementación con Node.js + nodemailer + plantillas Pug
- En desarrollo usar Mailtrap como SMTP dummy
- Publica `email.sent` en `email.events` para monitoreo | BillingService, MonitoringService | - Mock de eventos `invoice.paid` en RabbitMQ- Publica eventos `email.sent` a `email.events` para monitoreo | | **API Gateway** | Todos los anteriores| - Endpoints de Videos y Facturas apuntan a mocks B |

Con esta tabla, cada flujo que requiera datos de B usa **stubs ligeros**:

- FastAPI mocks en contenedores separados (`/mocks/video` , `/mocks/billing`).
- Productor de eventos periódicos en RabbitMQ (aio-pika) para facturas.

---

3. Riesgos de Descuentos y Mitigación Rápida

Elemento crítico	Riesgo de descuento	Mitigación exprés
Repositorio público	–10 décimas si no está accesible	Configurar visibilidad al crear el repo
README y .env.example	Nota mínima si no arranca	Documentar pasos Docker, seeder y credenciales Admin
Seeder usuarios	–30 décimas si falta o no funciona	Script Faker + commit de ejemplo de salida
Base de datos Users	Máx 4.0 si usan PostgreSQL	Elegir MariaDB/MySQL desde el scaffold inicial
Colección Postman	–30 décimas si incompleta	Exportar directamente desde `openapi.json`
Diagramas insuficientes	Nota mínima	Generar PlantUML C4 + secuencia de validación
Despliegue Docker/ CI	Penalización indirecta	Pipeline mínimo: build, tests, `compose up` healthchecks

Checklist express:

-

---

4. Scaffolding Acelerado con LLMs

1. **Boilerplate FastAPI**
2. Prompt LLM: "FastAPI CRUD Users + JWT auth + Alembic migrations + Dockerfile + tests Pytest".
3. **Generar mocks B**
4. Stub FastAPI para `/videos/{id}` con JSON dummy.
5. Consumidor RabbitMQ simulado que publique `invoice.paid` .
6. **OpenAPI → Postman**
7. Extraer `/openapi.json` , importar en Postman.
8. **PlantUML**
9. Prompt: "PlantUML C4 contenedores y secuencia: crear playlist (validar user/video)".
10. **Validación manual**
11. Revisar código generado: tipado explícito, pruebas unitarias, versiones fijas.

---

5. Infraestructura y Despliegue

- **docker-compose.yml** con servicios A, mocks B, RabbitMQ, MariaDB, Postgres.
- **Red**: `streamflow_net`. **Healthchecks** en `/healthz`.
- **CI básico** (GitHub Actions):
- Linter (flake8, isort)
- Build de imágenes
- Levantar Compose + ejecutar tests y seeder

---

6. API Gateway en FastAPI

- **Proxy**: `httpx.AsyncClient` para enrutar a cada servicio.
- **Middleware JWT**: extraer y validar token, inyectar `user_id`.
- **Rutas**:
- `/auth/*` → AuthService
- `/usuarios/*` → UsersService
- `/playlists/*` → PlaylistService
- `/email/*` → EmailService
- `/videos/*`, `/facturas/*` → mocks B

---

7. Plan de Trabajo por Fases (Esfuerzo Óptimo)

1. **Setup Inicial y Contratos**
2. Crear estructura de carpetas y repos.
3. Definir protos gRPC y exchanges RabbitMQ.
4. Configurar `docker-compose.yml`.
5. **Auth & Users**
6. Scaffold AuthService (FastAPI + JWT + Postgres/Alembic).
7. Scaffold UsersService (FastAPI + SQLAlchemy + MariaDB/Alembic).
8. Seeder de usuarios y publicación `user.created`.
9. **Playlists & Mocks B**
10. Scaffold PlaylistService (FastAPI + Postgres).
11. Integrar stub VideoService (FastAPI mock o gRPC stub).
12. **EmailService**
13. Scaffold FastAPI consumer RabbitMQ + envío SMTP.
14. Probar flujo con mock de `invoice.paid`.
15. **Gateway & Postman**
16. Implementar proxy y middleware JWT.
17. Generar colección Postman desde OpenAPI.
18. **Diagramas y CI**
19. Añadir PlantUML y diagrama de secuencia.
20. Ajustar CI: build, tests, `compose up` healthchecks.
21. **Revisión y Merge**

22. Verificar checklist de riesgos.
23. Hacer merge de repos o mono-repo.
24. Publicar y etiquetar release.

---

8. Modelos de Datos

AuthService (PostgreSQL)

```
CREATE TABLE blacklisted_tokens (
  token TEXT PRIMARY KEY,
  user_id UUID NOT NULL,
  expires_at TIMESTAMPTZ NOT NULL,
  created_at TIMESTAMPTZ DEFAULT now()
);
```

UsersService (MariaDB)

```
CREATE TABLE users (
  id CHAR(36) PRIMARY KEY,
  email VARCHAR(255) UNIQUE NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  full_name VARCHAR(100),
  role ENUM('free','premium','admin') NOT NULL,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
  updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

PlaylistService (PostgreSQL)

```
CREATE TABLE playlists (
  id SERIAL PRIMARY KEY,
  user_id CHAR(36) NOT NULL,
  name VARCHAR(100) NOT NULL,
  created_at TIMESTAMP DEFAULT now(),
  updated_at TIMESTAMP DEFAULT now()
);
CREATE TABLE playlist_videos (
  playlist_id INT,
  video_id CHAR(24),
  added_at TIMESTAMP DEFAULT now(),
  PRIMARY KEY (playlist_id, video_id),
```

```
  FOREIGN KEY (playlist_id) REFERENCES playlists(id)
);
```

EmailService

- **Comunicación**: asíncrona vía RabbitMQ; **no** expone gRPC.
- **Implementación**: Node.js con `nodemailer`, plantillas `pug` para HTML, configuración SMTP apuntando a Mailtrap en staging
- **Configuración**: valores SMTP en variables de entorno (`SMTP_HOST`, `SMTP_USER`, `SMTP_PASS`), `.env.example` actualizado
- **Nginx (puerta de entrada)**: opcional proxy inverso que maneje TLS y redirija `/email/*` a EmailService

Modelo de datos (opcional):

```
CREATE TABLE email_logs (
  id           UUID PRIMARY KEY,
  user_id      CHAR(36)      NOT NULL,
  to_email     VARCHAR(255)  NOT NULL,
  subject      VARCHAR(255)  NOT NULL,
  body         TEXT          NOT NULL,
  status       ENUM('pending','sent','failed') DEFAULT 'pending',
  error_msg    TEXT,
  template     VARCHAR(100),
  created_at   TIMESTAMPTZ   DEFAULT now(),
  sent_at      TIMESTAMPTZ
);
```