Universidad de las Fuerzas Armadas

ESPE Departamento de Ciencias de la Computación

Análisis y Diseño

Arquitectura

Integrantes:

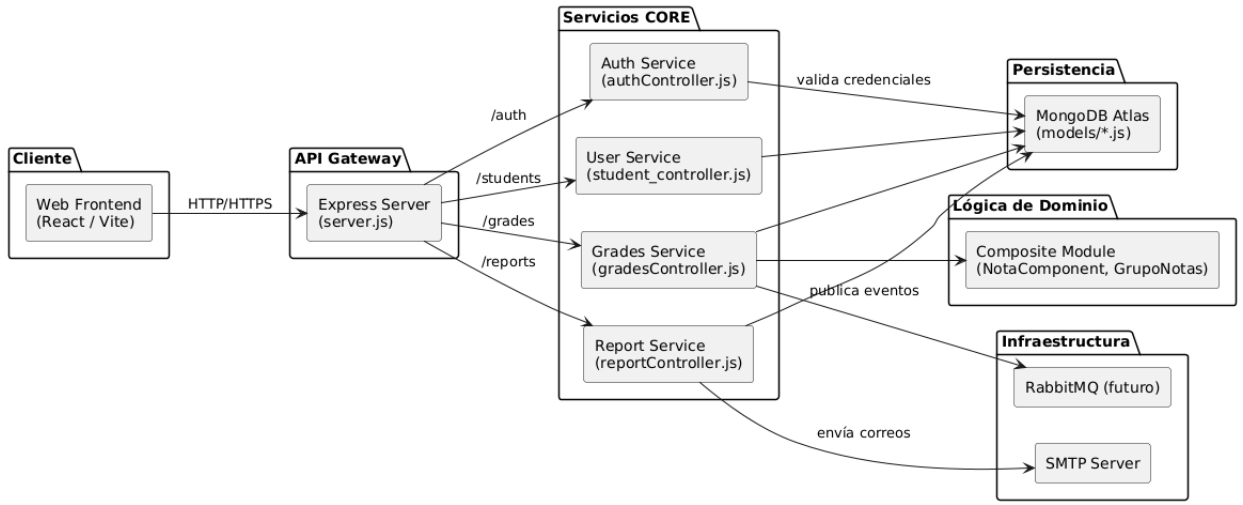
Isaac Escobar

Eduardo Mortensen

Diego Ponce

NRC: 14571

1. Arquitectura



Código

```
@startuml
skinparam componentStyle rectangle
left to right direction

package "Cliente" {
  [Web Frontend\n(React / Vite)]
}

package "API Gateway" {
  [Express Server\n(server.js)]
}

package "Servicios CORE" {
  [Auth Service\n(authController.js)]
  [User Service\n(student_controller.js)]
  [Grades Service\n(gradesController.js)]
  [Report Service\n(reportController.js)]
}

package "Lógica de Dominio" {
  [Composite Module\n(NotaComponent, GrupoNotas)]
}

package "Persistencia" {
  [MongoDB Atlas\n(models/*.js)]
}

package "Infraestructura" {
  [SMTP Server]
  [RabbitMQ (futuro)]
}
```

```
' Flujo de peticiones
[Web Frontend\n(React / Vite)] --> [Express Server\n(server.js)] : HTTP/HTTPS
[Express Server\n(server.js)] --> [Auth Service\n(authController.js)]    : /auth
[Express Server\n(server.js)] --> [User Service\n(student_controller.js)]: /students
[Express Server\n(server.js)] --> [Grades Service\n(gradesController.js)]  : /grades
[Express Server\n(server.js)] --> [Report Service\n(reportController.js)] : /reports

' Dependencias internas
[Grades Service\n(gradesController.js)] --> [Composite Module\n(NotaComponent,
GrupoNotas)]
[User Service\n(student_controller.js)] --> [MongoDB Atlas\n(models/*.js)]
[Grades Service\n(gradesController.js)] --> [MongoDB Atlas\n(models/*.js)]
[Report Service\n(reportController.js)] --> [MongoDB Atlas\n(models/*.js)]

' Servicios externos
[Auth Service\n(authController.js)] --> [MongoDB Atlas\n(models/*.js)]      : valida
credenciales
[Report Service\n(reportController.js)] --> [SMTP Server] : envía correos
[Grades Service\n(gradesController.js)] --> [RabbitMQ (futuro)] : publica eventos
@enduml
```

Descripción de Capas

1. **Cliente (Frontend)**
    - Aplicación web construida con React/Vite.
    - Consume la API REST para autenticación, gestión de usuarios, calificaciones y reportes.
2. **API Gateway (Express Server)**
    - Punto de entrada único (server.js).
    - Enruta las peticiones a los distintos servicios core: Auth, User, Grades y Report.
    - Aplica CORS, parsing de JSON y registro de logs.
3. **Servicios CORE**
    - **Auth Service** (authController.js): Valida credenciales, genera y verifica JWT.
    - **User Service** (student_controller.js): Registra y administra estudiantes.
    - **Grades Service** (gradesController.js): Registra, actualiza y borra calificaciones; se integra con el módulo Composite para agrupar notas.
    - **Report Service** (reportController.js): Genera reportes en PDF y los envía por correo.
4. **Lógica de Dominio**

    - **Composite Module** (NotaComponent, NotaIndividual, GrupoNotas): Implementa el patrón Composite para representar las notas de manera jerárquica (por materia y grupo de evaluaciones).

5. **Persistencia**
    - **MongoDB Atlas**: Base de datos NoSQL gestionada.
    - Modelos Mongoose (User.js, Grade.js) para almacenar usuarios y notas.
6. **Infraestructura y Servicios Externos**
    - **SMTP Server**: Servicio de correo para envío de reportes.
    - **RabbitMQ** (planificado): Canal de mensajería para futuro desacoplamiento de eventos de calificaciones.

Flujo de Datos

1. El **usuario** interactúa en el navegador, realiza login o solicita acciones (registro, calificaciones, reportes).
2. El **Frontend** envía peticiones HTTP al **Express Server**, que verifica JWT y enruta a los controllers correspondientes.
3. Cada **Controller** ejecuta la lógica de negocio y, si corresponde, utiliza el **módulo Composite** para estructurar las notas.
4. Las operaciones de lectura/escritura van a **MongoDB Atlas** a través de los modelos Mongoose.
5. Cuando se generan reportes, el **Report Service** crea un PDF y lo envía vía **SMTP** al destinatario.
6. En futuras versiones, el **Grades Service** podrá publicar eventos en **RabbitMQ** para notificar otros microservicios.

Conclusiones

- La arquitectura en **capas** favorece la **mantenibilidad** y la **escalabilidad**, ya que cada módulo tiene una responsabilidad clara.
- El uso del **API Gateway** centraliza la configuración común (autenticación, CORS, logging) y simplifica el consumo desde cualquier cliente.
- La separación entre **business logic** y **persistencia** mediante Mongoose facilita cambiar de base de datos o migrar datos sin afectar la lógica de aplicación.
- La introducción del **patrón Composite** en el manejo de notas aporta flexibilidad para agregar nuevos niveles de agregación (por ejemplo, secciones, parciales).
- La planificación de la integración de **RabbitMQ** muestra una visión de evolución hacia una arquitectura de microservicios basada en eventos.