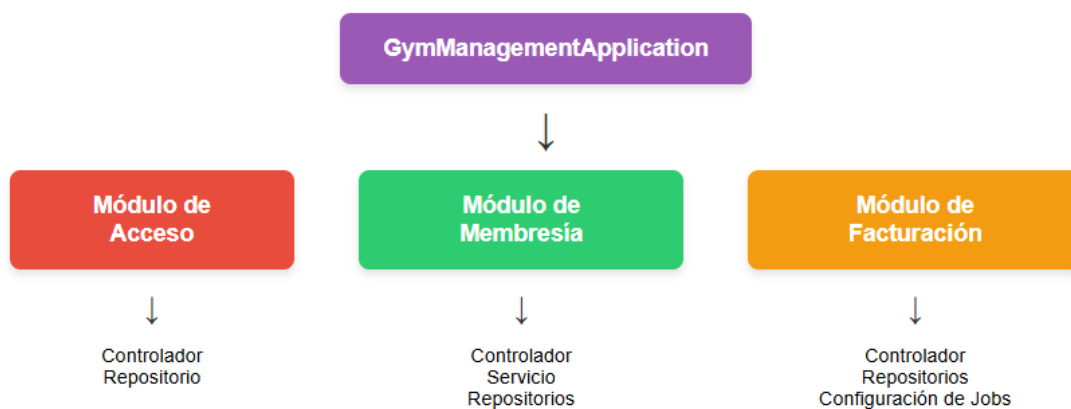


Sistema de Gestión de Gimnasio

Arquitectura General del Sistema

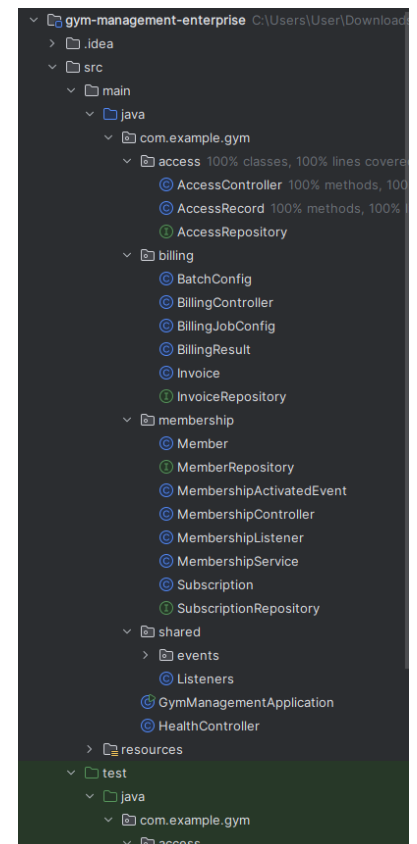
El sistema es un Sistema de Gestión de Gimnasio/Fitness, modular y desacoplado, desarrollado con Spring Boot 2.7.x y Java 11, usando Spring Modulith y spring Batch para organizar la aplicación en módulos independientes y comunicados mediante eventos.

Arquitectura del Proyecto



Tecnologías utilizadas

- Java 11 + Spring Boot 2.7.x
- Spring Modulith → módulos desacoplados
- Spring Batch → jobs de facturación
- MySQL → persistencia estructurada
- MongoDB → persistencia flexible
- Lombok
- Eventos desacoplados → comunicación entre módulos
- Docker Compose → levanta MySQL + MongoDB
- Tests unitarios e integrados → cobertura mínima 80%
- Jacoco → medición de cobertura
- README.md → documentación de instalación y uso



Infraestructura con Docker Compose

El proyecto utiliza un archivo `docker-compose.yml` para levantar los servicios necesarios, como MySQL, MongoDB.

Contenido del archivo `docker-compose.yml`:

```
version: '3.8'
services:
  mysql:
    image: mysql:8.0
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: gymdb
      MYSQL_USER: gym
      MYSQL_PASSWORD: gympass
    ports:
      - "3306:3306"
    volumes:
      - mysql_data:/var/lib/mysql
  mongo:
    image: mongo:6.0
    environment:
      MONGO_INITDB_ROOT_USERNAME: admin
      MONGO_INITDB_ROOT_PASSWORD: xideral4321
      MONGO_INITDB_DATABASE: gym_mongo
    ports:
      - "27017:27017"
    volumes:
      - mongo_data:/data/db
volumes:
  mysql_data:
  mongo_data:
```

Application.properties

```
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.hibernate.naming.physical-
strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardIm
pl
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialec
t
spring.jpa.properties.hibernate.jdbc.time_zone=UTC

# =====
# MONGODB CONFIG
# =====
spring.data.mongodb.uri=mongodb://admin:xideral234@localhost:27017/gymdb?
authSource=admin
spring.data.mongodb.database=gym_mongo

# =====
# SPRING BATCH CONFIG
# =====
spring.batch.jdbc.schema=classpath:org/springframework/batch/core/schema-
mysql.sql
spring.batch.initialize-schema=always
spring.batch.job.enabled=true
spring.datasource.platform=mysql

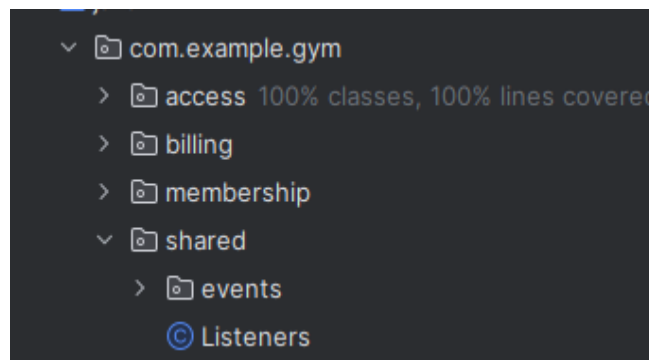
# =====
# LOGGING
# =====
logging.level.org.springframework=INFO
logging.level.com.example.gym=DEBUG

# =====
# OPENAPI / SWAGGER
# =====
springdoc.api-docs.path=/v3/api-docs
springdoc.swagger-ui.path=/swagger-ui.html
```

Módulos principales:

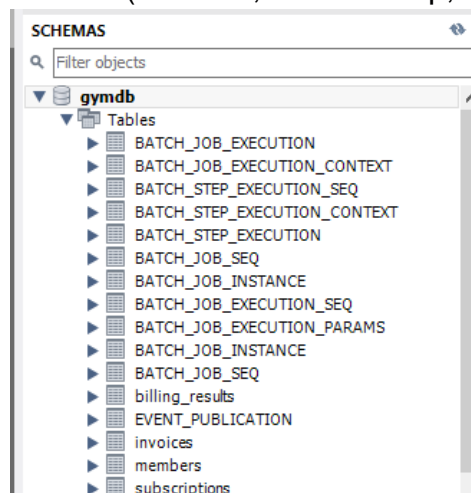
Módulo	Funcionalidad principal
Membership	Gestión de usuarios, membresías, planes y promociones. Emite MembershipActivatedEvent.
Billing	Facturación mensual automatizada mediante Spring Batch. Emite InvoiceGeneratedEvent.
Access	Control de accesos al gimnasio. Registra accesos en MongoDB. Emite AccessGrantedEvent.
Shared	Contiene eventos de dominio, DTOs y contratos comunes entre módulos.

Imagen ilustrativa con los modulos principales

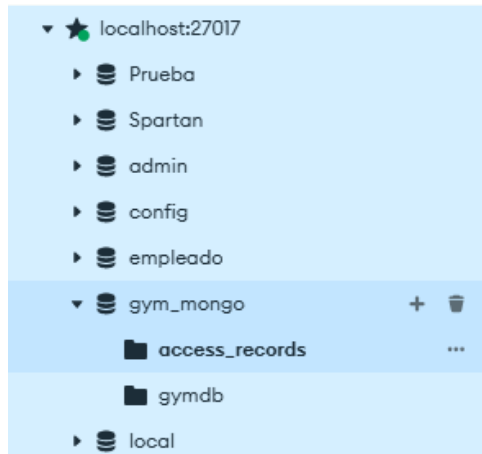


Bases de datos:

- MySQL: Datos estructurados (Member, Membership, Invoice, etc.)



- MongoDB: Datos no estructurados o masivos (registros de accesos, auditoría, logs)



Ambos se levantan con Docker Compose, facilitando el despliegue local.

Con el siguiente commando podras levantar docker compose para un despliegue local.

```
docker compose up -d
```

Spring Modulith

- Cada módulo tiene responsabilidad clara y mínima dependencia de otros.
- Comunicación entre módulos mediante eventos.

Ejemplo de flujo de eventos:



1. Usuario activa su membresía → MembershipActivatedEvent.
2. Billing escucha el evento → genera la primera factura → InvoiceGeneratedEvent.
3. Access registra la entrada del socio → AccessGrantedEvent.

Spring Batch

El módulo Billing utiliza Spring Batch para automatizar la facturación.

Configuración principal:

- BatchConfig.java → configuración general del batch.

```
// Lector: obtiene los miembros activos
@Bean 1 usage
public ItemReader<Member> memberItemReader() {
    final List<Member> actives = memberRepository.findByActiveTrue();
    final Iterator<Member> it = actives.iterator();
    return () -> it.hasNext() ? it.next() : null;
}

// Procesador: convierte Member → Invoice
@Bean 1 usage
public ItemProcessor<Member, Invoice> memberToInvoiceProcessor() {
    return Member member -> {
        BigDecimal amount = member.getMonthlyFee() != null
            ? member.getMonthlyFee()
            : BigDecimal.valueOf(29.99);
        return new Invoice(member.getId(), amount);
    };
}
```

- BillingJobConfig.java → define Jobs y Steps.

```
public class BillingJobConfig {

    @Bean no usages
    public JpaPagingItemReader<Member> reader(EntityManagerFactory emf) {
        return new JpaPagingItemReaderBuilder<Member>()
            .name("memberReader")
            .entityManagerFactory(emf)
            .queryString("SELECT m FROM Member m WHERE m.active = true")
            .pageSize(5)
            .build();
    }

    // Procesador: aplica descuento y calcula mensualidad final
    @Bean no usages
    public ItemProcessor<Member, BillingResult> processor() {
        return Member member -> {
            BillingResult result = new BillingResult();
            result.setMemberId(member.getId());
            result.setMemberName(member.getName());
            result.setMonthlyFee(member.getMonthlyFee());

            // Lógica de descuentos
            BigDecimal discount = BigDecimal.ZERO;
            if (member.getMonthlyFee().compareTo(new BigDecimal("700")) > 0) {
                discount = member.getMonthlyFee().multiply(new BigDecimal("0.15")); // 15%
            } else if (member.getMonthlyFee().compareTo(new BigDecimal("600")) > 0) {
                discount = member.getMonthlyFee().multiply(new BigDecimal("0.10")); // 10%
            }

            result.setDiscount(discount);
            result.setFinalFee(member.getMonthlyFee().subtract(discount));
            result.setInvoiceDate(LocalDate.now());
            result.setStatus(Math.random() > 0.5 ? "PAID" : "PENDING");

            System.out.println("Factura: " + result.getMemberName() +
                " | Total: " + result.getFinalFee() +
                " | Descuento: " + result.getDiscount() +
                " | Estado: " + result.getStatus());

            return result;
        };
    }
}
```

Flujo del Job billingJob:

Flujo del Job billingJob



1. Lectura: miembros activos desde MySQL.
2. Procesamiento: calcula mensualidades, aplica descuentos/promociones.
3. Escritura: genera facturas (Invoice) y las persiste en MySQL.
4. Evento: publica InvoiceGeneratedEvent.

Notas de implementación:

- JOB_EXECUTION_ID identifica cada ejecución.
- Parámetros de job: nombre, tipo, valor.
- Tabla BATCH_JOB_EXECUTION_PARAMS con primary key combinada y foreign key para evitar duplicados.

4. Eventos del Sistema

Evento	Emisor	Impacto / Receptor
MembershipActivatedEvent	Membership	Billing, Access
ClassBookedEvent	Classes	Billing
EquipmentMaintenanceEvent	Equipment	Classes, Billing
AccessGrantedEvent	Access	Membership, Analytics
InvoiceGeneratedEvent	Billing	Contabilidad / sistemas externos

Persistencia de Datos

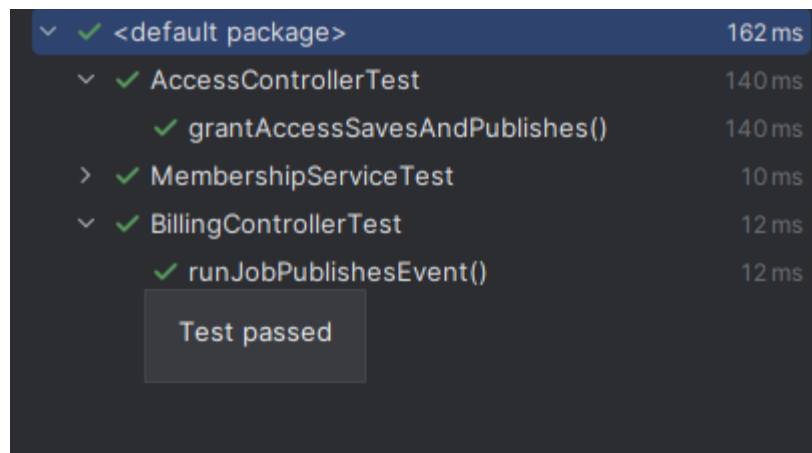
Tipo	Uso	Ejemplo de entidades/repositorios
MySQL	Datos estructurados	Member, Membership, Invoice, MemberRepository, InvoiceRepository
MongoDB	Datos históricos o no estructurados	Accesos, auditoría, logs, AccessRecord

Lombok

- @Data → getters, setters, toString, equals, hashCode.
- @NoArgsConstructor / @AllArgsConstructor → constructores automáticos.
- @Builder → construcción fluida de objetos.

Pruebas

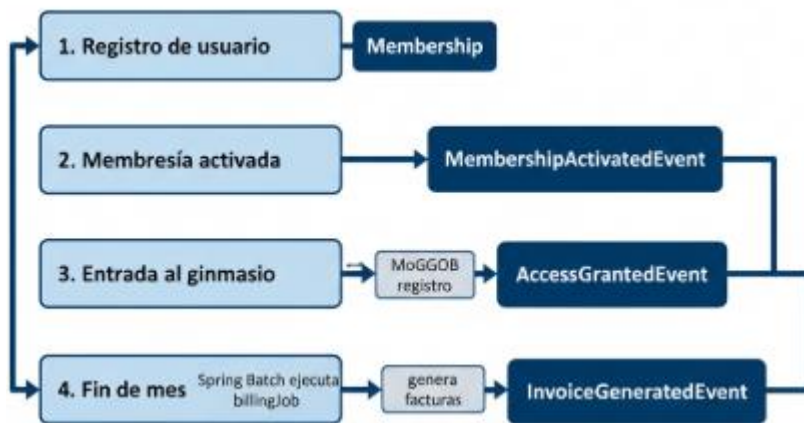
- Unitarias: lógica de negocio y servicios individuales.
- Integradas: interacción entre módulos con bases en memoria o Testcontainers.



- Cobertura mínima: 80% obteniendo un 90%

com.example.gym	90% (9/20)	19% (16/80)	90% (15/17...)	0% (0/2)
> access	100% (2/2)	100% (0/0)	100% (1/7)	100% (0/0)
> billing	10% (0/6)	3% (1/33)	4% (3/93)	0% (0/10)
BatchConfig	0% (0/9)	0% (0/9)	0% (0/23)	0% (0/4)
BillingController	100% (1/1)	100% (1/1)	100% (3/4)	100% (0/0)
BillingJobConfig	0% (0/1)	0% (0/2)	0% (0/26)	0% (0/6)
BillingResult	0% (0/1)	0% (0/14)	0% (0/14)	100% (0/0)
Invoice	0% (0/1)	0% (0/2)	0% (0/6)	100% (0/0)
InvoiceRepository	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
> membership	36% (2/6)	26% (10/38)	25% (17/86)	0% (0/10)
Member	100% (1/1)	70% (4/10)	69% (8/13)	100% (0/0)
MemberRepository	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
MembershipActivated	0% (0/1)	0% (0/2)	0% (0/5)	100% (0/0)
MembershipController	0% (0/1)	0% (0/7)	0% (0/13)	0% (0/8)
MembershipListener	0% (0/1)	0% (0/1)	0% (0/4)	100% (0/0)
MembershipService	100% (1/1)	30% (3/10)	36% (8/22)	0% (0/2)
Subscription	0% (0/1)	0% (0/1)	100% (10)	100% (0/0)
SubscriptionRepository	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
> shared	35% (3/4)	42% (3/0)	65% (7/10)	100% (0/0)
> events	100% (3/3)	75% (3/0)	87% (8/0)	100% (0/0)
Listeners	0% (0/1)	0% (0/3)	0% (0/2)	100% (0/0)

Flujo de Negocio General



1. Registro de usuario → Membership.
2. Membresía activada → MembershipActivatedEvent.
3. Entrada al gimnasio → MongoDB registro → AccessGrantedEvent.
4. Fin de mes → Spring Batch ejecuta billingJob → genera facturas → InvoiceGeneratedEvent.

Pruebas con postman

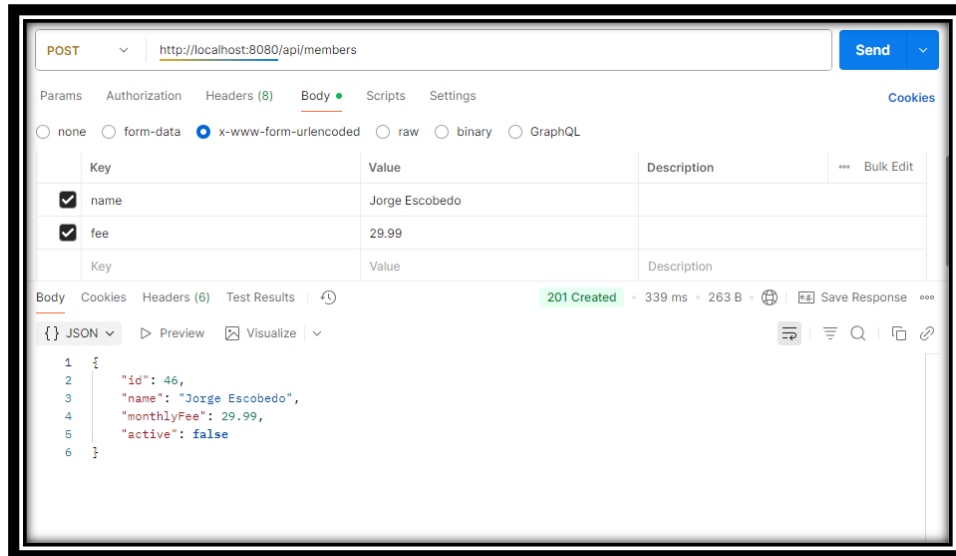
Endpoints del sistema Gym Management

Módulo de Socios

1. Crear Socio

Método: POST

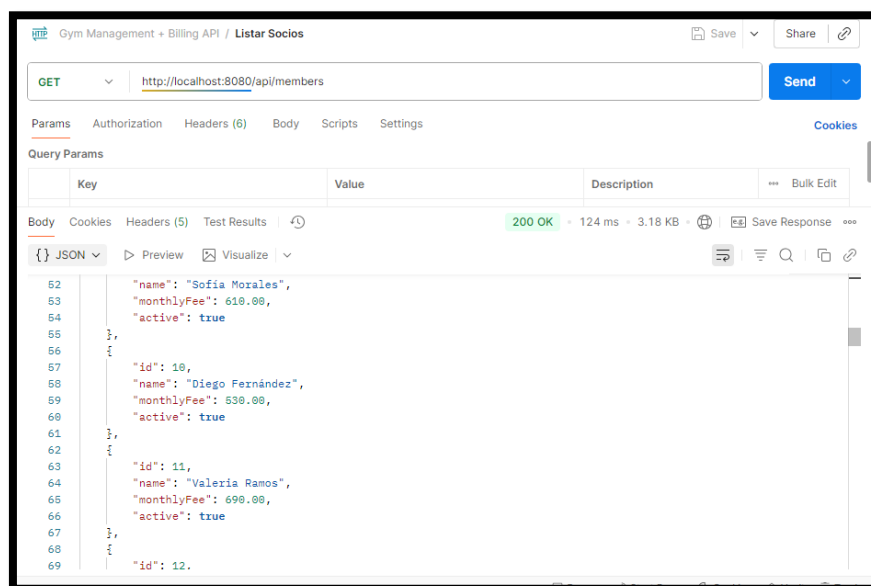
URL: `http://localhost:8080/api/members`



2. Listar Socios

Método: GET

URL: `http://localhost:8080/api/members`



3. Activar Socio

Método: POST

URL: `http://localhost:8080/api/members/{id}/activate`

The screenshot shows a REST client interface for a project named "Gym Management + Billing API / Activar Socio". The request is a POST to `http://localhost:8080/api/members/2/activate`. The response is a 200 OK status with a response time of 36 ms and a body size of 229 B. The response body is displayed in JSON format:

```
1 {
2   "id": 2,
3   "name": "Jorge Escobedo",
4   "monthlyFee": 29.99,
5   "active": true
6 }
```

4. Actualizar Socio

Método: PUT

URL: `http://localhost:8080/api/members/{id}`

Parámetros:

- o name = *Jorge E.*
- o fee = *39.99*

The screenshot shows a REST client interface for a PUT request to `http://localhost:8080/api/members/1`. The request body is selected as "x-www-form-urlencoded". The response is a 200 OK status with a response time of 45 ms and a body size of 221 B. The response body is displayed in JSON format:

```
1 {
2   "id": 1,
3   "name": "Jorge E.",
4   "monthlyFee": 559,
5   "active": true
6 }
```

5. Eliminar Socio

Método: DELETE

URL: `http://localhost:8080/api/members/{id}`

DELETE ▼ `http://localhost:8080/api/members/1` Send ▼

Params Authorization Headers (6) Body Scripts Settings Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

Body Cookies Headers (3) Test Results ↺ 204 No Content • 43 ms • 112 B • 🌐 | 📄 Save Response ...

📄 Raw ▶ Preview 🖼️ Visualize ▼ 🔍 🔗 📄

1

Módulo de Membresías

6. Listar Membresías

Método: GET

URL: `http://localhost:8080/api/members/subscriptions`

GET ▼ `http://localhost:8080/api/members/subscriptions` Send ▼

Params Authorization Headers (6) Body Scripts Settings Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

Body Cookies Headers (5) Test Results ↺ 200 OK • 25 ms • 293 B • 🌐 | 📄 Save Response ...

{ } JSON ▶ Preview 🖼️ Visualize ▼ 🔍 🔗 📄

```
1  [
2    {
3      "id": 1,
4      "name": "Plan plata",
5      "price": 499.00
6    },
7    {
8      "id": 2,
9      "name": "Plan oro",
10     "price": 499.00
11  },
12 ]
```

7. Crear Membresía

Método: POST

URL: http://localhost:8080/api/members/subscriptions

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:8080/api/members/subscriptions
- Body:** x-www-form-urlencoded
- Headers:** 8
- Response:** 201 Created, 24 ms, 252 B
- Body Content (JSON):**

```
{  "id": 4,  "name": "Plan diamante",  "price": 499}
```

Módulo de Facturación

8. Ejecutar Job de Facturación (Spring Batch)

Método: POST

URL: http://localhost:8080/api/billing/run

Cuerpo: {}

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:8080/api/billing/run
- Body:** {}
- Headers:** 8
- Response:** 200 OK, 1.04 s, 185 B
- Body Content (Text):**

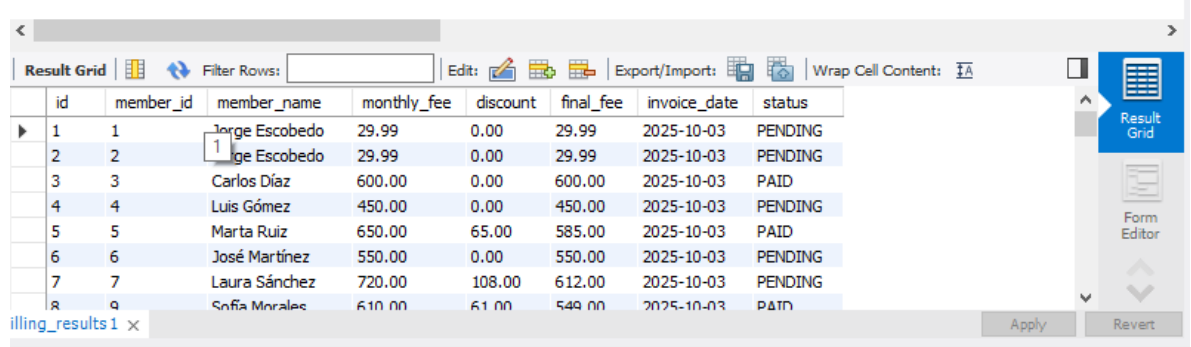
```
1 Job status: COMPLETED
```

Descripción: Lanza el proceso batch que genera las facturas de los miembros activos.

Resultado en los logs

```
Factura: Jorge Escobedo | Total: 29.99 | Descuento: 0 | Estado: PAID
Factura: Carlos Díaz | Total: 600.00 | Descuento: 0 | Estado: PENDING
Factura: Luis Gómez | Total: 450.00 | Descuento: 0 | Estado: PAID
Factura: Marta Ruiz | Total: 585.0000 | Descuento: 65.0000 | Estado: PENDING
Factura: José Martínez | Total: 550.00 | Descuento: 0 | Estado: PENDING
Hibernate: select m1_0.id,m1_0.active,m1_0.monthlyFee,m1_0.name from members m1_0 where m1_0.active=1 limit ?,?
Factura: Laura Sánchez | Total: 612.0000 | Descuento: 108.0000 | Estado: PAID
Factura: Sofía Morales | Total: 549.0000 | Descuento: 61.0000 | Estado: PENDING
Factura: Diego Fernández | Total: 530.00 | Descuento: 0 | Estado: PAID
Factura: Valeria Ramos | Total: 621.0000 | Descuento: 69.0000 | Estado: PENDING
Factura: Paula Castro | Total: 560.00 | Descuento: 0 | Estado: PENDING
Hibernate: select m1_0.id,m1_0.active,m1_0.monthlyFee,m1_0.name from members m1_0 where m1_0.active=1 limit ?,?
Factura: Ricardo Jiménez | Total: 576.0000 | Descuento: 64.0000 | Estado: PENDING
Factura: Carolina Vega | Total: 580.00 | Descuento: 0 | Estado: PENDING
Factura: Fernando Ruiz | Total: 558.0000 | Descuento: 62.0000 | Estado: PENDING
Factura: Jorge Medina | Total: 603.5000 | Descuento: 106.5000 | Estado: PAID
```

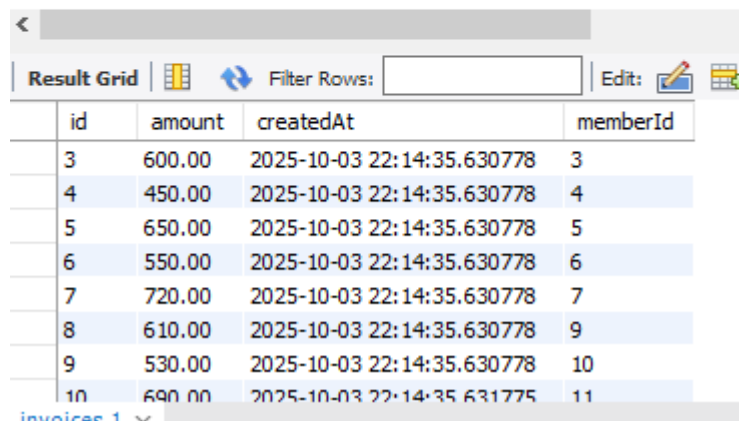
Resultado en la base de datos



	id	member_id	member_name	monthly_fee	discount	final_fee	invoice_date	status
1	1	1	Jorge Escobedo	29.99	0.00	29.99	2025-10-03	PENDING
2	2	2	Jorge Escobedo	29.99	0.00	29.99	2025-10-03	PENDING
3	3	3	Carlos Díaz	600.00	0.00	600.00	2025-10-03	PAID
4	4	4	Luis Gómez	450.00	0.00	450.00	2025-10-03	PENDING
5	5	5	Marta Ruiz	650.00	65.00	585.00	2025-10-03	PAID
6	6	6	José Martínez	550.00	0.00	550.00	2025-10-03	PENDING
7	7	7	Laura Sánchez	720.00	108.00	612.00	2025-10-03	PENDING
8	8	9	Sofía Morales	610.00	61.00	549.00	2025-10-03	PAID

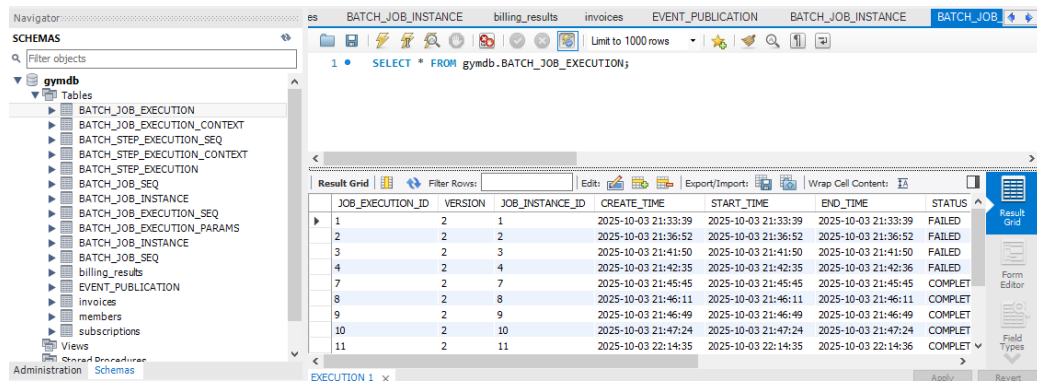
Seleccionamos todos los registros para verificar si se estan guardando con exito cada facture generada

```
1 • SELECT * FROM gymdb.invoices;
```



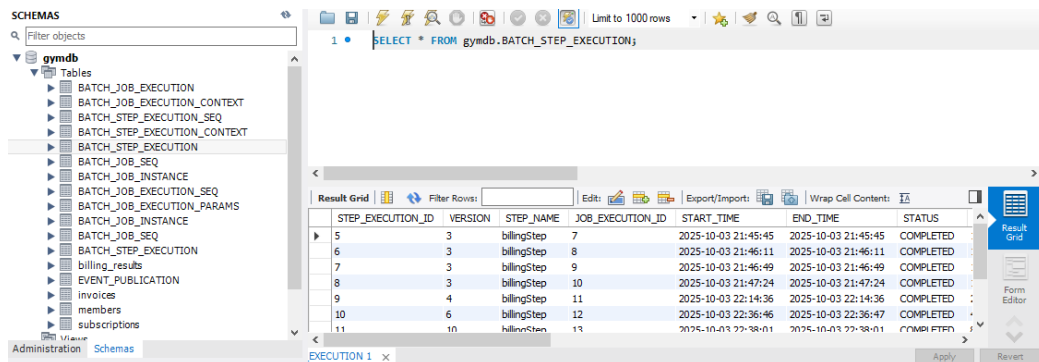
	id	amount	createdAt	memberId
	3	600.00	2025-10-03 22:14:35.630778	3
	4	450.00	2025-10-03 22:14:35.630778	4
	5	650.00	2025-10-03 22:14:35.630778	5
	6	550.00	2025-10-03 22:14:35.630778	6
	7	720.00	2025-10-03 22:14:35.630778	7
	8	610.00	2025-10-03 22:14:35.630778	9
	9	530.00	2025-10-03 22:14:35.630778	10
	10	690.00	2025-10-03 22:14:35.631775	11

Pruebas de que el proceso Batch se ejecuta exitosamente



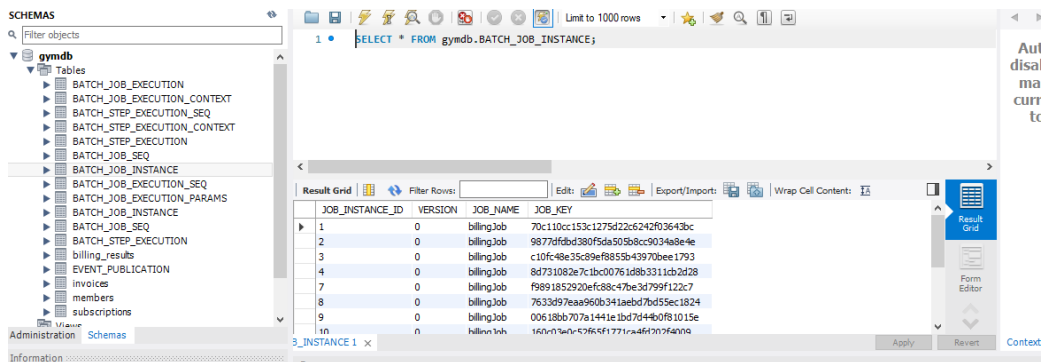
JOB_EXECUTION_ID	VERSION	JOB_INSTANCE_ID	CREATE_TIME	START_TIME	END_TIME	STATUS
1	2	1	2025-10-03 21:33:39	2025-10-03 21:33:39	2025-10-03 21:33:39	FAILED
2	2	2	2025-10-03 21:36:52	2025-10-03 21:36:52	2025-10-03 21:36:52	FAILED
3	2	3	2025-10-03 21:41:50	2025-10-03 21:41:50	2025-10-03 21:41:50	FAILED
4	2	4	2025-10-03 21:42:35	2025-10-03 21:42:35	2025-10-03 21:42:35	FAILED
7	2	7	2025-10-03 21:45:45	2025-10-03 21:45:45	2025-10-03 21:45:45	COMPLETED
8	2	8	2025-10-03 21:46:11	2025-10-03 21:46:11	2025-10-03 21:46:11	COMPLETED
9	2	9	2025-10-03 21:46:49	2025-10-03 21:46:49	2025-10-03 21:46:49	COMPLETED
10	2	10	2025-10-03 21:47:24	2025-10-03 21:47:24	2025-10-03 21:47:24	COMPLETED
11	2	11	2025-10-03 22:14:35	2025-10-03 22:14:35	2025-10-03 22:14:35	COMPLETED

Aquí se ve que algunos jobs fallaron y otros terminaron correctamente (COMPLETED), lo cual indica que se están lanzando las ejecuciones y quedan registradas.



STEP_EXECUTION_ID	VERSION	STEP_NAME	JOB_EXECUTION_ID	START_TIME	END_TIME	STATUS
5	3	billingStep	7	2025-10-03 21:45:45	2025-10-03 21:45:45	COMPLETED
6	3	billingStep	8	2025-10-03 21:46:11	2025-10-03 21:46:11	COMPLETED
7	3	billingStep	9	2025-10-03 21:46:49	2025-10-03 21:46:49	COMPLETED
8	3	billingStep	10	2025-10-03 21:47:24	2025-10-03 21:47:24	COMPLETED
9	4	billingStep	11	2025-10-03 22:14:36	2025-10-03 22:14:36	COMPLETED
10	6	billingStep	12	2025-10-03 22:36:46	2025-10-03 22:36:46	COMPLETED
11	10	billingStep	13	2025-10-03 22:38:01	2025-10-03 22:38:01	COMPLETED

El step billingStep se está ejecutando varias veces y termina con COMPLETED. Eso significa que el flujo reader → processor → writer se completó.



JOB_INSTANCE_ID	VERSION	JOB_NAME	JOB_KEY
1	0	billingJob	70c110cc153c1275d22c6242f03643bc
2	0	billingJob	9877dfdd380f5da505b9cc9034a8e4e
3	0	billingJob	c10fc48e35c89ef8853b43970bee1793
4	0	billingJob	8d731082e7c1bc00761d8b3311cb2d28
7	0	billingJob	f9891852920efc88c47be3d799f122c7
8	0	billingJob	7633d97eaa960b341aebd7bd55ec1824
9	0	billingJob	00618bb707a1441e1bd7d44b0f81015e
10	0	billingJob	160c110cc153c1275d22c6242f03643bc
11	0	billingJob	9877dfdd380f5da505b9cc9034a8e4e

Se han creado varias instancias del job billingJob, lo cual confirma que el proceso batch se ejecutó en distintas ocasiones con sus propios parámetros de entrada.