Ecole Normale Supérieure de l'Enseignement Technique Mohammedia

Université Hassan II de Casablanca

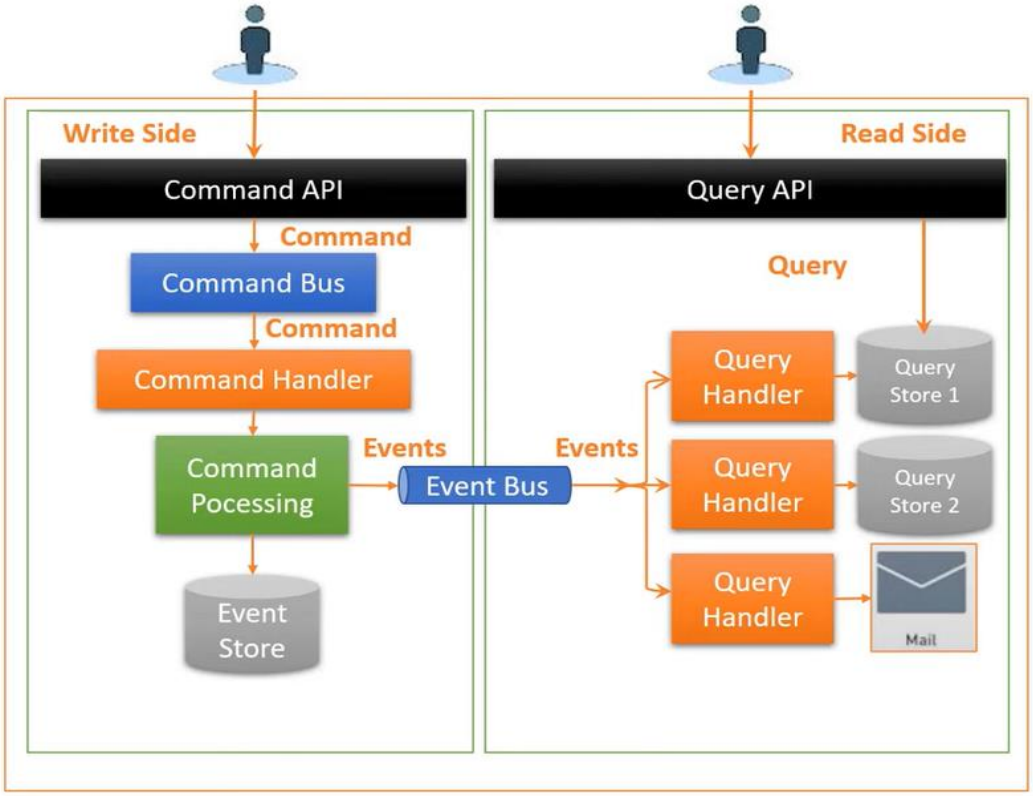
ENSET

ا لمدرسة العليا لأساتذة التعليم التقني المحمدية

جامعة الحسن الثاني بالدار البيضاء

COMPTE-RENDU : ACTIVITE PRATIQUE N° 5 - EVENT DRIVEN ARCHITECTURE CQRS AND EVENT SOURCING)

Filière :« Ingénierie Informatique : Big Data et Cloud Computing » II-BDCC



Réalisé par :

Khadija BENJILALI

Encadré par :

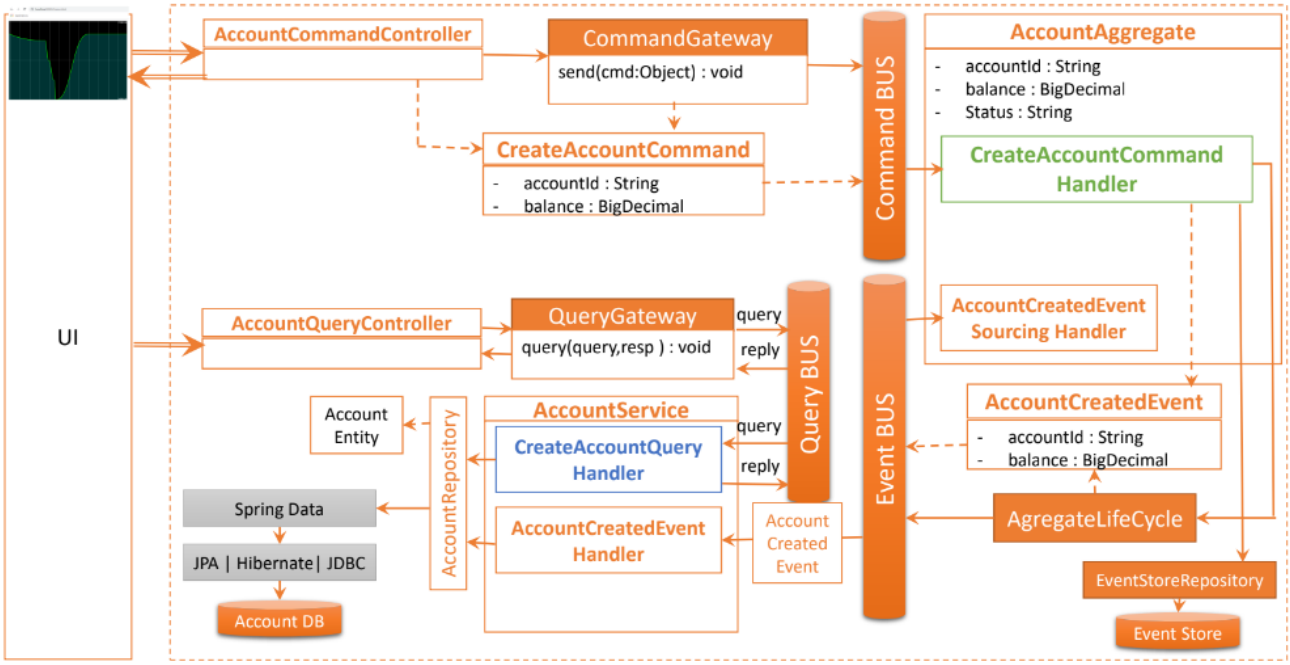
Pr. Mohamed YOUSSFI

Année Universitaire : 2022-2023

Sommaire

Travail à faire

Créer une application qui permet de gérer des comptes bancaires.

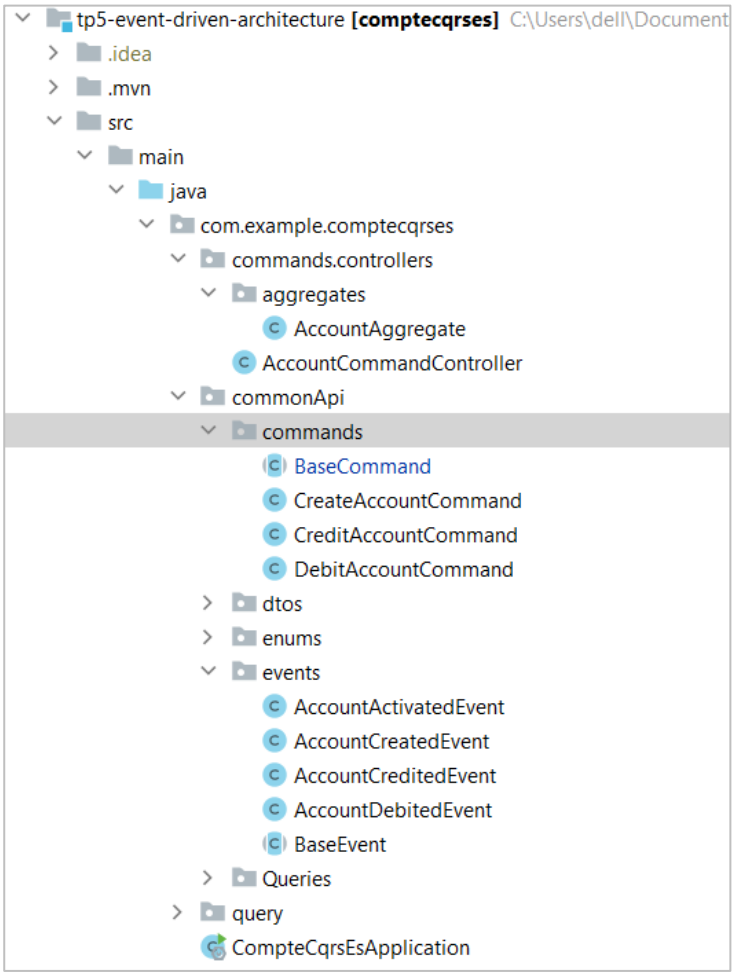


Permet de :

- **Ajouter un Compte**
- **Activer un compte après création**
- **Créditer un compte**
- **Débiter un compte**
- **Consulter un compte**
- **Consulter les comptes**
- **Consulter les opérations d'un compte**
- **Suivre en temps réel l'état d'un compte**

PARTIE 1 : Structure et dépendances du projet

1. Structure du projet :



2. Maven Dependencies :

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>org.axonframework</groupId>
    <artifactId>axon-spring-boot-starter</artifactId>
    <version>4.4.3</version>
    <exclusions>
        <exclusion>
            <groupId>org.axonframework</groupId>
            <artifactId>axon-server-connector</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

```xml
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <version>8.0.31</version>
        <scope>runtime</scope>
    </dependency>
```

```xml
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>
```

3. Configuration de compte-service :

```
spring.application.name=compte-service
spring.datasource.url=jdbc:mysql://${MYSQL_HOST:localhost}:${MYSQL_PORT:3306}/bank?createDatabaseIfNotExist=true
spring.datasource.username=${MYSQL_USER:root}
spring.datasource.password=${MYSQL_PASSWORD:}
spring.jpa.hibernate.ddl-auto=create
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MariaDBDialect
server.port=8082
```

PARTIE 2 : Common-api

1. BaseCommand :

```
public abstract class BaseCommand<T>
{
    //identifiant de l'aggregat où on va effectuer la commande
    @TargetAggregateIdentifier
    @Getter private T id;

    public BaseCommand(T id) { this.id = id; }
}
```

2. Les commandes de ce service :

- **CreateAccountCommand**

```
public class CreateAccountCommand extends BaseCommand<String>
{
    @Getter private double initialBalance;
    @Getter private String currency;
    public CreateAccountCommand(String id, double initialBalance, String currency) {
        super(id);
        this.initialBalance = initialBalance;
        this.currency = currency;
    }
}
```

- **CreditAccountCommand**

```
public class CreditAccountCommand extends BaseCommand<String>
{
    @Getter private double creditAmount;
    @Getter private String currency;
    public CreditAccountCommand(String id, double creditAmount, String currency) {
        super(id);
        this.creditAmount = creditAmount;
        this.currency = currency;
    }
}
```

➢ **DebitAccountCommand**

```
public class DebitAccountCommand extends BaseCommand<String>
{
    @Getter private double debitAmount;
    @Getter private String currency;
    public DebitAccountCommand(String id, double debitAmount, String currency) {
        super(id);
        this.debitAmount = debitAmount;
        this.currency = currency;
    }
}
```

3. BaseEvent :

```
public abstract class BaseEvent<T>
{
    @Getter private T id;

    public BaseEvent(T id) {
        this.id = id;
    }
}
```

4. Les évènements de ce service :

➢ **AccountCreatedEvent**

```
public class AccountActivatedEvent extends BaseEvent<String>
{
    @Getter private AccountStatus accountStatus;

    public AccountActivatedEvent(String id, AccountStatus accountStatus) {
        super(id);
        this.accountStatus = accountStatus;
    }
```

➢ **AccountActivatedEvent**

```
public class AccountCreatedEvent extends BaseEvent<String> {
    @Getter private double accountBalance;
    @Getter private String currency;

    public AccountCreatedEvent(String id, double accountBalance, String currency) {
        super(id);
        this.accountBalance = accountBalance;
        this.currency = currency;
    }
}
```

➢ **AccountCreditedEvent**

```
public class AccountCreditedEvent extends BaseEvent<String>
{
    @Getter private double creditAmount;
    @Getter private String currency;

    public AccountCreditedEvent(String id, double creditAmount, String currency) {
        super(id);
        this.creditAmount = creditAmount;
        this.currency = currency;
    }
```

➢ **AccountDebitedEvent**

```
public class AccountDebitedEvent extends BaseEvent<String> {
    @Getter private double debitAmount;
    @Getter private String currency;

    public AccountDebitedEvent(String id, double debitAmount, String currency) {
        super(id);
        this.debitAmount = debitAmount;
        this.currency = currency;
    }
}
```

5. DTOS :

➢ **CreateAccountRequestDTO**

```
@Data @NoArgsConstructor @AllArgsConstructor
public class CreateAccountRequestDTO
{
    private double initialBalance;
    private String currency;
}
```

➢ **CreditAccountRequestDTO**

```
@Data @NoArgsConstructor @AllArgsConstructor
public class CreditAccountRequestDTO {
    private String accountId;
    private double creditAmount;
    private String currency;
}
```

➢ **DebitAccountRequestDTO**

```java
@Data @NoArgsConstructor @AllArgsConstructor
public class DebitAccountRequestDTO {
    private String accountId;
    private double debitAmount;
    private String currency;
}
```

6. Les requêtes :

```java
public class GetAllAccountsQuery {

}
```

```java
@Data @NoArgsConstructor @AllArgsConstructor
public class GetAccountQuery
{
    private String id;
}
```

PARTIE 3 : Partie Ecriture « Command »

1. Agrégat AccountAggregate :

```java
// la classe où on va executer la logique métier
@Aggregate
public class AccountAggregate
{
    @AggregateIdentifier
    private String accountId;
    private double balance;
    private String currency;
    private AccountStatus status;

    public AccountAggregate()
    {
        //required by AXON
    }
```

```java
// La fonction de décision
@CommandHandler
public AccountAggregate(CreateAccountCommand createAccountCommand)
{
    if(createAccountCommand.getInitialBalance()<0) throw new RuntimeException("Impossible ...");
    //OK
    // Créer un evenement et stocker dans lui createAccountCommand
    AggregateLifecycle.apply(new AccountCreatedEvent(
            createAccountCommand.getId(),
            createAccountCommand.getInitialBalance(),
            createAccountCommand.getCurrency()
    ));
}
// La fonction d'évolution
@EventSourcingHandler
public void on(AccountCreatedEvent event)
{
    this.accountId = event.getId();
    this.balance = event.getAccountBalance();
    this.currency = event.getCurrency();
    this.status = AccountStatus.CREATED;

    AggregateLifecycle.apply(new AccountActivatedEvent(
            event.getId(),
            AccountStatus.ACTIVATED
    ));
}

@EventSourcingHandler
public void on(AccountActivatedEvent event) { this.status = event.getAccountStatus(); }
```

```java
// La fonction de déscision de  la commande credit
@CommandHandler
public void handle(CreditAccountCommand command) {
    if (command.getCreditAmount() < 0) {
        throw new RuntimeException("Credit amount cannot be negative");
    }
    AggregateLifecycle.apply(new AccountCreditedEvent(
            command.getId(), command.getCreditAmount(), command.getCurrency()
    ));
}

@EventSourcingHandler
public void on(AccountCreditedEvent event) { this.balance += event.getCreditAmount(); }

@CommandHandler
public void handle(DebitAccountCommand command) {
    if (command.getDebitAmount() < 0) {
        throw new RuntimeException("Debit amount should not be negative");
    }
    if (this.balance < command.getDebitAmount()) {
        throw new RuntimeException("Balance not sufficient");
    }
    AggregateLifecycle.apply(new AccountDebitedEvent(
            command.getId(), command.getDebitAmount(), command.getCurrency()
    ));
}

@EventSourcingHandler
public void on(AccountDebitedEvent event) { this.balance -= event.getDebitAmount(); }
```

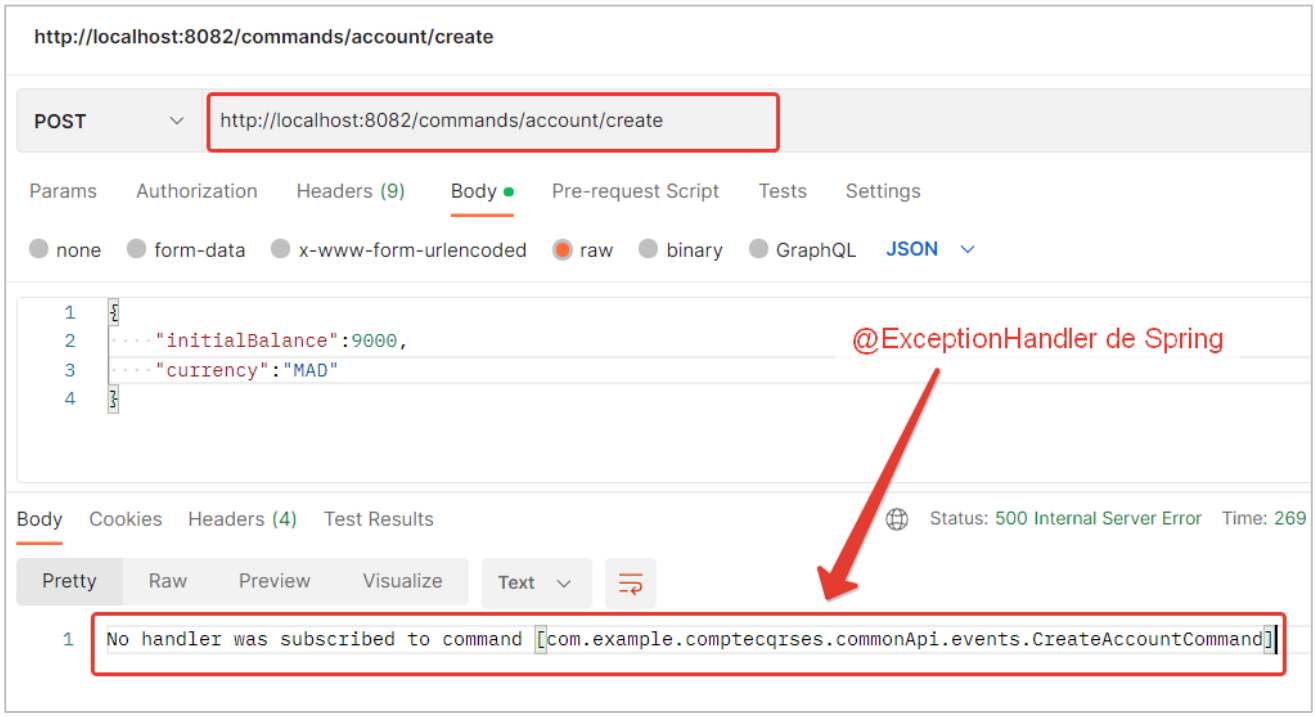
2. AccountCommandController :

```java
@RestController
@RequestMapping(path = "/commands/account")
@AllArgsConstructor
public class AccountCommandController
{
    private CommandGateway commandGateway;
    private EventStore eventStore;
    @PostMapping("/create")
    public CompletableFuture<String> createAccount(@RequestBody CreateAccountRequestDTO request)
    {
        // chaque fois il y a une commande créer on va la mettre
        // va nous retouner l'id de commande
        CompletableFuture<String> commandResponse = commandGateway.send(new CreateAccountCommand(
                UUID.randomUUID().toString(),
                request.getInitialBalance(),
                request.getCurrency()
        ));
        return commandResponse;
    }
```

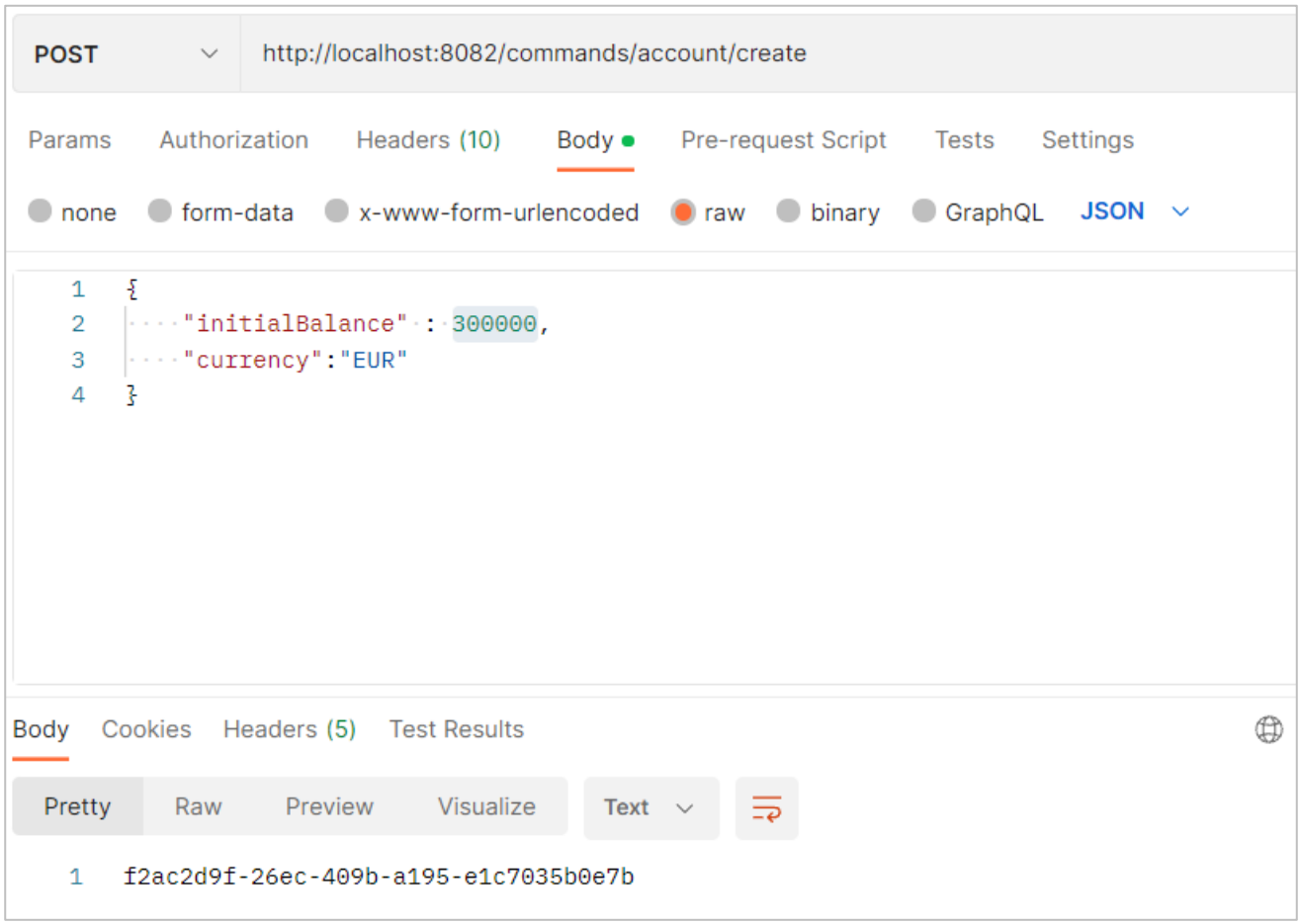
```java
@PutMapping("/credit")
public CompletableFuture<String> creditAccount(@RequestBody CreditAccountRequestDTO request)
{
    CompletableFuture<String> commandResponse = commandGateway.send(new CreditAccountCommand(
            request.getAccountId(),
            request.getCreditAmount(),
            request.getCurrency()
    ));
    return commandResponse;
}

@PutMapping(path = "/debit")
public CompletableFuture<String> debitAccount(@RequestBody DebitAccountRequestDTO request) {
    CompletableFuture<String> commandResponse = commandGateway.send(new DebitAccountCommand(request.getAccountId(),
            request.getDebitAmount(), request.getCurrency()));
    return commandResponse;
}
```

3. Gérer une exception :

```java
@ExceptionHandler(Exception.class)
public ResponseEntity<String> exceptionHandler(Exception exception)
{
    ResponseEntity<String> entity = new ResponseEntity<>(
            exception.getMessage(),
            HttpStatus.INTERNAL_SERVER_ERROR);
    return entity;
}
```

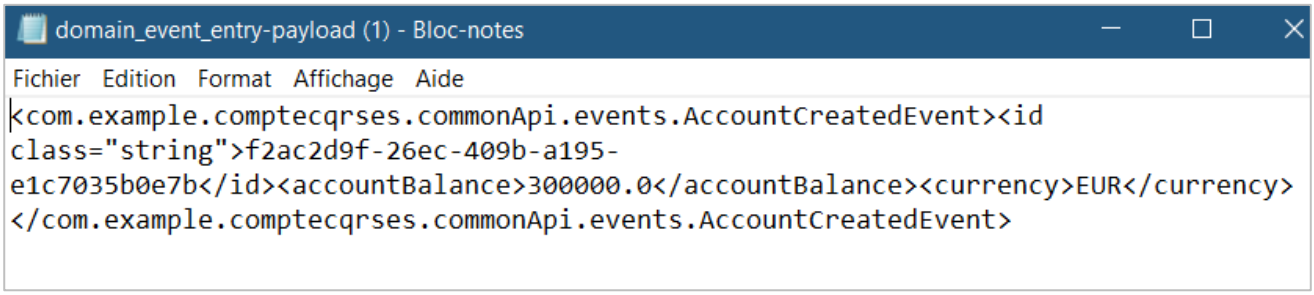


4. Ajouter un compte :

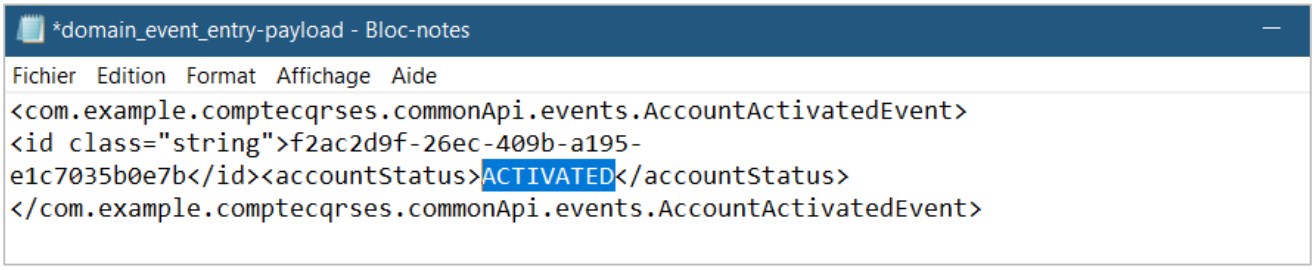


global_index	event_identifier	meta_data	payload	payload_revision	payload_type	time_stamp	aggregate_identifier
1	06aa6951-cf08-4112-bc5f-246150afd3a2	[BLOB - 213 o]	[BLOB - 252 o]	*NULL*	com.example.comptecqrses.commonApi.events.AccountC...	2022-12-26T23:19:04.312Z	f2ac2d9f-26ec-409b-a195-e1c7035b0e7b
2	1118101f-f8f5-44d6-9f89-539c9a297207	[BLOB - 213 o]	[BLOB - 231 o]	*NULL*	com.example.comptecqrses.commonApi.events.AccountA...	2022-12-26T23:19:04.320Z	f2ac2d9f-26ec-409b-a195-e1c7035b0e7b

Le compte a été créé:

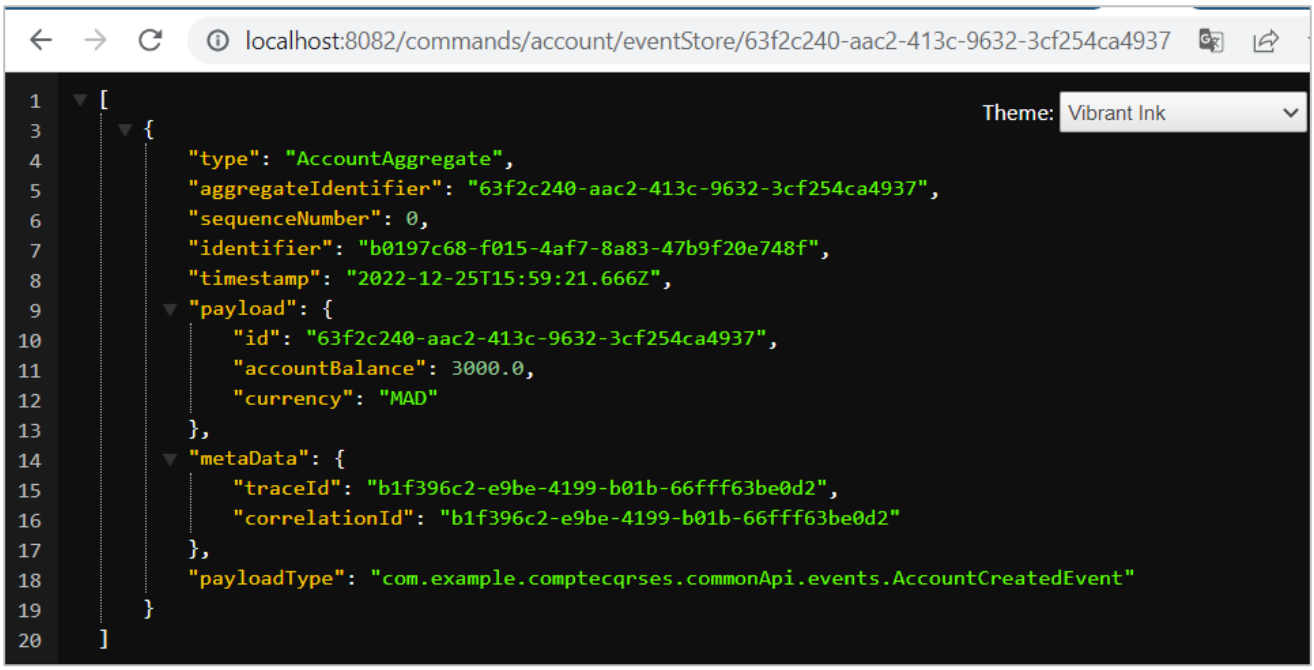


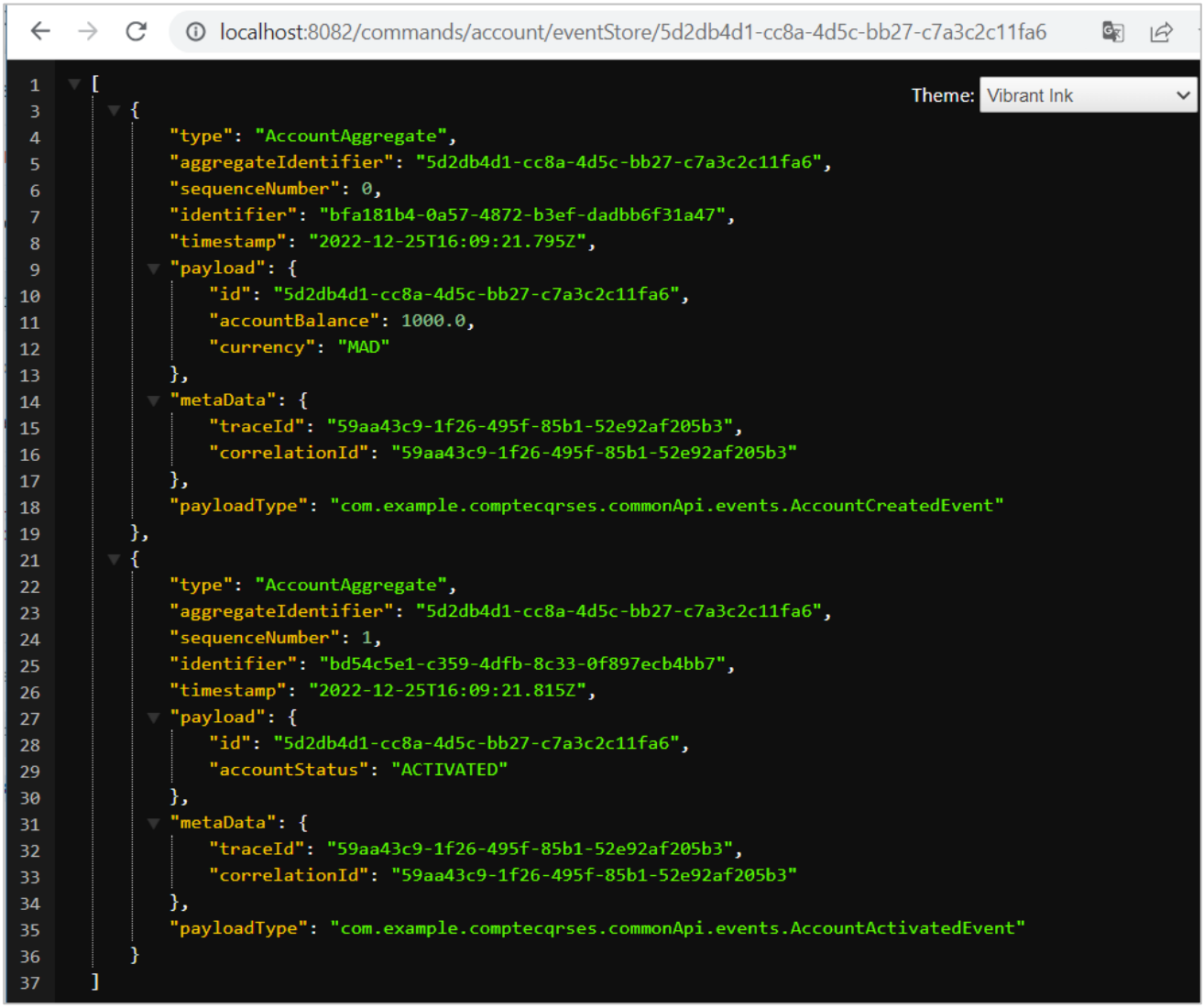
Le compte a été activé :



5. EventStore :

```
// afficher Event Store où on trouve les info des events
@GetMapping("/eventStore/{accountId}")
public Stream eventStore(@PathVariable String accountId)
{
    return  eventStore.readEvents(accountId).asStream();
}
```





PARTIE 4 : Partie Lecture « Query »

1. Les entités de compte-service :

   ➢ **Account**

```java
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Account {
    @Id
    private String id;
    private double balance;
    private String currency;
    @Enumerated(EnumType.STRING)
    private AccountStatus status;
    @OneToMany(mappedBy = "account")
    private Collection<Operation> operations;

}
```

➢ **Operation**

```java
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Operation {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private Date date;
    private double montant;
    @Enumerated(EnumType.STRING)
    private OperationType type;

    @ManyToOne
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private Account account;
}
```

2. QueryAccountController :

```java
@RestController
@RequestMapping(path = "/query/accounts")
@AllArgsConstructor
public class QueryAccountController
{
    private QueryGateway queryGateway;


    @GetMapping(path = "/allAccounts")
    public List<Account> getAccounts() {
        return queryGateway.query(
                new GetAllAccountsQuery(),
                ResponseTypes.multipleInstancesOf(Account.class)).join();
    }

    @GetMapping(path="/getAccount/{id}")
    public Account getAccount(@PathVariable String id) {
        return queryGateway.query(new GetAccountQuery(id),
                ResponseTypes.instanceOf(Account.class)).join();
    }
}
```

3. Les EventHandler :

```java
@Service @AllArgsConstructor
@Slf4j
public class AccountServiceHandler {
    private AccountRepository accountRepository;
    private OperationAccountRepository operationAccountRepository;

    @EventHandler
    public void on(AccountCreatedEvent event) {
        log.info("*****************************");
        log.info("AccountCreatedEvent received");

        Account account = new Account();
        account.setId(event.getId());
        account.setBalance(event.getAccountBalance());
        account.setCurrency(event.getCurrency());
        account.setStatus(AccountStatus.CREATED);
        accountRepository.save(account);
    }
```

```java
    @EventHandler
    public void on(AccountActivatedEvent event) {
        log.info("*****************************");
        log.info("AccountActivatedEvent received");
        Account account = accountRepository.findById(event.getId()).get();
        account.setStatus(event.getAccountStatus());
        accountRepository.save(account);
    }

    @EventHandler
    public void on(AccountCreditedEvent event) {
        log.info("*****************************");
        log.info("AccountCreditedEvent received");
        Account account = accountRepository.findById(event.getId()).get();
        Operation operation = new Operation();
        operation.setMontant(event.getCreditAmount());
        operation.setDate(new Date());
        operation.setType(OperationType.CREDIT);
        operation.setAccount(account);
        operationAccountRepository.save(operation);
        account.setBalance(account.getBalance() + event.getCreditAmount());
        accountRepository.save(account);
    }
```

```java
    @EventHandler
    public void on(AccountDebitedEvent event) {
        log.info("****************************");
        log.info("AccountDebitedEvent received");

        Account account = accountRepository.findById(event.getId()).get();

        Operation operation = new Operation();
        operation.setMontant(event.getDebitAmount());
        operation.setDate(new Date());
        operation.setType(OperationType.DEBIT);
        operation.setAccount(account);
        operationAccountRepository.save(operation);
        account.setBalance(account.getBalance() - event.getDebitAmount());
        accountRepository.save(account);
    }

    @QueryHandler
    public List<Account> on(GetAllAccountsQuery query) { return accountRepository.findAll(); }

    @QueryHandler
    public Account on(GetAccountQuery query) { return accountRepository.findById(query.getId()).get(); }
}
```

4. Récupérer les comptes :

+ Options

	id	balance	currency	status
Éditer Copier Supprimer	04f59323-9a85-4746-9019-d0770ef8bdc6	2134	EUR	ACTIVATED
Éditer Copier Supprimer	de95a174-0aa3-4036-b228-dc01a7ee87f3	10000	MAD	ACTIVATED
Éditer Copier Supprimer	f2ac2d9f-26ec-409b-a195-e1c7035b0e7b	300000	EUR	ACTIVATED

localhost:8082/query/accounts/allAccounts


```json
[
    {
        "id": "04f59323-9a85-4746-9019-d0770ef8bdc6",
        "balance": 2134.0,
        "currency": "EUR",
        "status": "ACTIVATED",
        "operations": []
    },
    {
        "id": "de95a174-0aa3-4036-b228-dc01a7ee87f3",
        "balance": 10000.0,
        "currency": "MAD",
        "status": "ACTIVATED",
        "operations": []
    },
    {
        "id": "f2ac2d9f-26ec-409b-a195-e1c7035b0e7b",
        "balance": 300000.0,
        "currency": "EUR",
        "status": "ACTIVATED",
        "operations": []
    }
]
```