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

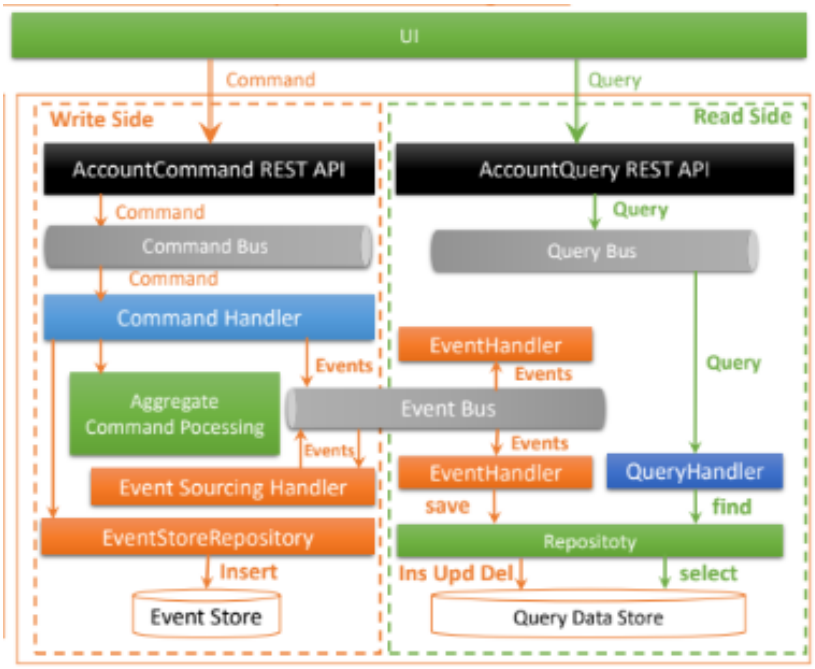
Département Mathématiques et Informatique

Cycle Ingénieur

« Ingénierie Informatique – Big Data et Cloud Computing »

COMPTE-RENDU:

Activité pratique N° 5 - Event Driven Architecture CQRS and Event Sourcing



Réalisé par: Asmaa ELASRI **Encadré par:** Pr. Mohamed YOUSSFI

Année Universitaire : 2022-2023

Travail à faire

Créer une application qui permet de gérer des comptes respectant les patterns CQRS et Event Sourcing avec les Framework AXON et Spring Boot.

PARTIE 1 : Ecriture

1. Creation des commandes (CommonApi)

- BaseCommand:

```java
public class BaseCommand<T> {

    1 usage
    @TargetAggregateIdentifier
    @Getter private T id;

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

- **CreateAccountCommand**

```java
public class CreateAccountCommand extends  BaseCommand<String>{
    1 usage
    @Getter
    private  String currency ;
    1 usage
    @Getter private  double initialSolde;
    no usages
    public CreateAccountCommand(String id, String currency, double initialSolde) {
        super(id);
        this.currency = currency;
        this.initialSolde = initialSolde;
    }
}
```

- **CreditAccountCommand**

```java
public class CreditAccountCommand extends  BaseCommand<String>{
    1 usage
    @Getter
    private  String currency ;
    1 usage
    @Getter private  double amount;
    no usages
    public CreditAccountCommand(String id, String currency, double amount) {
        super(id);
        this.currency = currency;
        this.amount = amount;
    }
}
```

- **DebitAccountCommand**

```java
public class DebitAccountCommand extends  BaseCommand<String>{
    1 usage
    @Getter
    private  String currency ;
    1 usage
    @Getter private  double amount;
    no usages
    public DebitAccountCommand(String id, String currency, double amount) {
        super(id);
        this.currency = currency;
        this.amount = amount;
    }
}
```

- **Controller CreateAccountCommand (Commands)**

```java
@AllArgsConstructor @NoArgsConstructor
@Data
public class CreateAccountRequestDTO {
    no usages
    private String currency ;
    no usages
    private double initialBalance;
}
```

```java
@RestController
@RequestMapping(path = "/commands/account")
@AllArgsConstructor
public class AccountCommandController {
    1 usage
    private CommandGateway commandGateway;
    no usages
    @PostMapping(path = "/create")
    public CompletableFuture<String> createAccount(CreateAccountRequestDTO request){
        CompletableFuture<String>commandResponse= commandGateway.send(new CreateAccountCommand(
                UUID.randomUUID().toString(),
                request.getCurrency(),
                request.getInitialBalance()
        ));
```

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

```
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=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MariaDBDialect
server.port=8082
```

- **Base de données (PhpMyAdmin)**

Table	Action	Lignes	Type	Interclassement	Taille	Perte
association_value_entry	Parcourir Structure Rechercher Insérer Vider Supprimer	0	InnoDB	utf8mb4_general_ci	48,0 kio	-
domain_event_entry	Parcourir Structure Rechercher Insérer Vider Supprimer	0	InnoDB	utf8mb4_general_ci	48,0 kio	-
hibernate_sequence	Parcourir Structure Rechercher Insérer Vider Supprimer	1	InnoDB	utf8mb4_general_ci	16,0 kio	-
saga_entry	Parcourir Structure Rechercher Insérer Vider Supprimer	0	InnoDB	utf8mb4_general_ci	16,0 kio	-
snapshot_event_entry	Parcourir Structure Rechercher Insérer Vider Supprimer	0	InnoDB	utf8mb4_general_ci	32,0 kio	-
token_entry	Parcourir Structure Rechercher Insérer Vider Supprimer	0	InnoDB	utf8mb4_general_ci	16,0 kio	-
6 tables	Somme	1	InnoDB	utf8mb4_general_ci	176,0 kio	0 o

Tout cocher Avec la sélection :

2. Creation des événements (CommonApi)

- **BaseEvent**

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

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

- **AccountCreatedEvent**

```java
public class AccountCreatedEvent extends BaseEvent<String>{

    1 usage
    @Getter
    private String currency ;
    1 usage
    @Getter
    private double initialBalance ;

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

- **Creation de l'agrégat (AccountAggregate)**

```java
public class AccountAggregate {
    1 usage
    @AggregateIdentifier
    private String accoutId;
    1 usage
    private String currency;
    1 usage
    private double balance ;
    1 usage
    private AccountStatus status;
    no usages
    public AccountAggregate(){
        //Required by AXON
    }
```

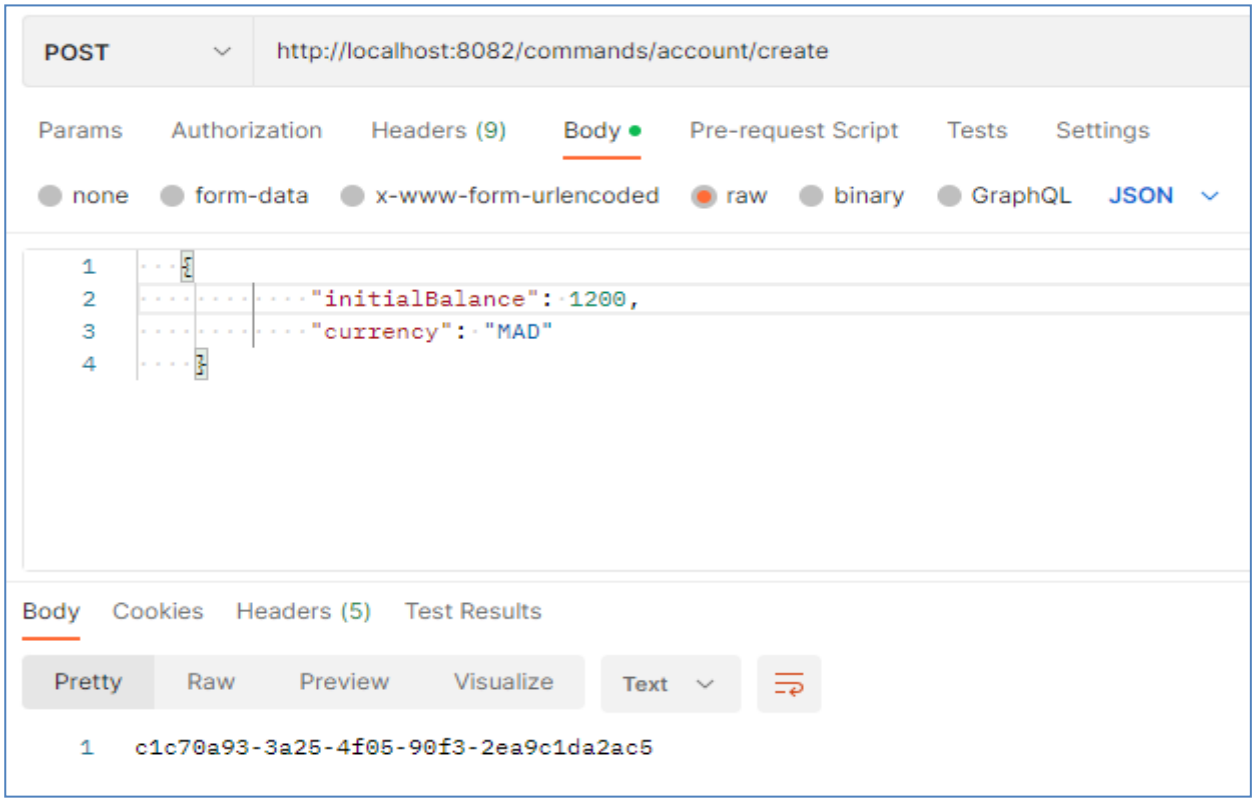
- **CommandHandler**

```java
    @CommandHandler
    public AccountAggregate(CreateAccountCommand createAccountCommand) {
        if(createAccountCommand.getInitialBalance()<0) throw new RuntimeException("Impossible...");
        AggregateLifecycle.apply(new AccountCreatedEvent(
                createAccountCommand.getId(),
                createAccountCommand.getCurrency(),
                createAccountCommand.getInitialBalance()
        ));
    }
```

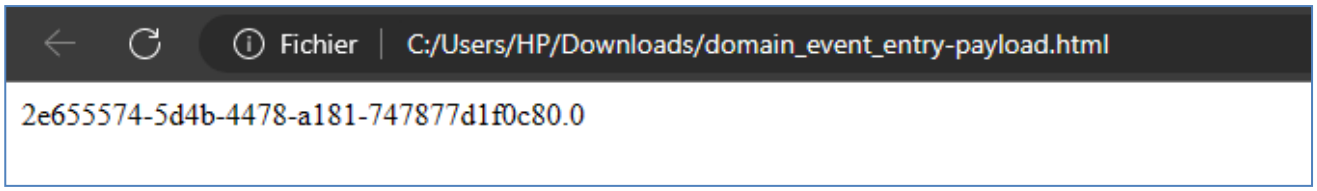
- **EventSourcingHandler**

```java
@EventSourcingHandler
public void on(AccountCreatedEvent event){
    this.accoutId = event.getId();
    this.balance =event.getInitialBalance() ;
    this.status = AccountStatus.CREATED;
    this.currency = event.getCurrency() ;
}
}
```

- **Test (Postman)**

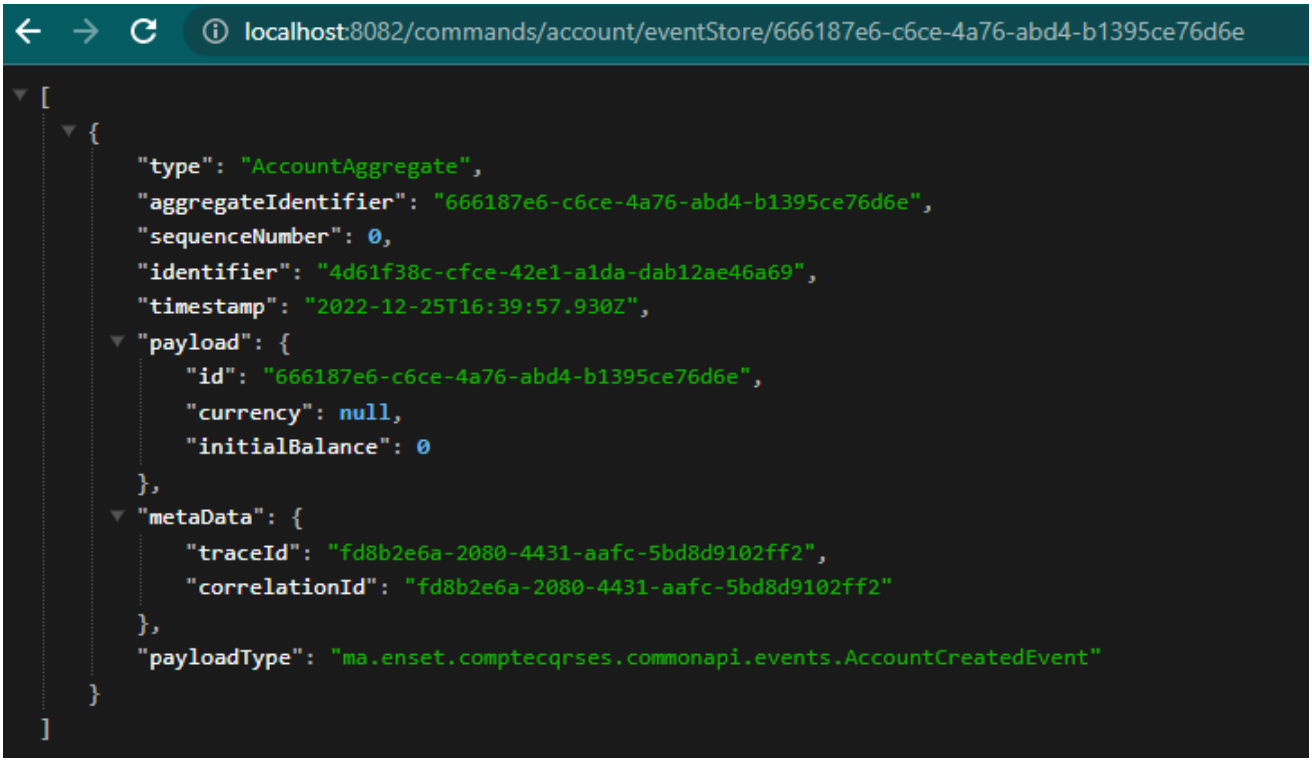


- **EventStore**

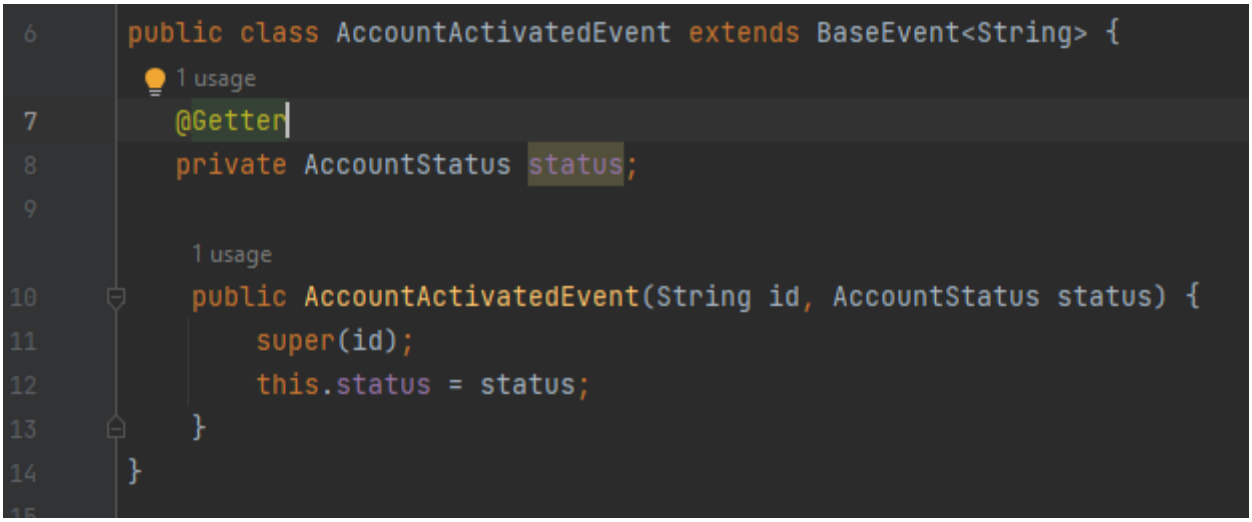


- **EventStore (Controller)**

```java
@GetMapping("/eventStore/{accountId}")
public Stream eventStore(@PathVariable String accountId){
    return eventStore.readEvents(accountId).asStream() ;
}
```

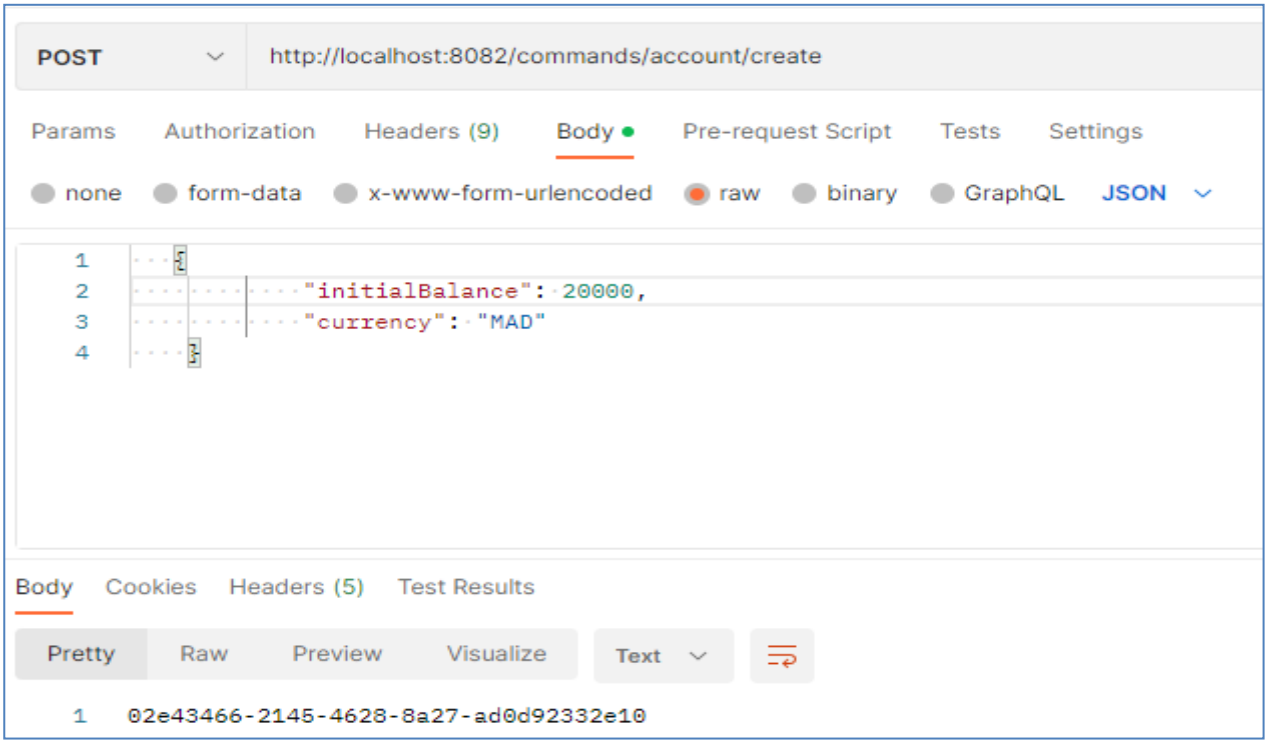


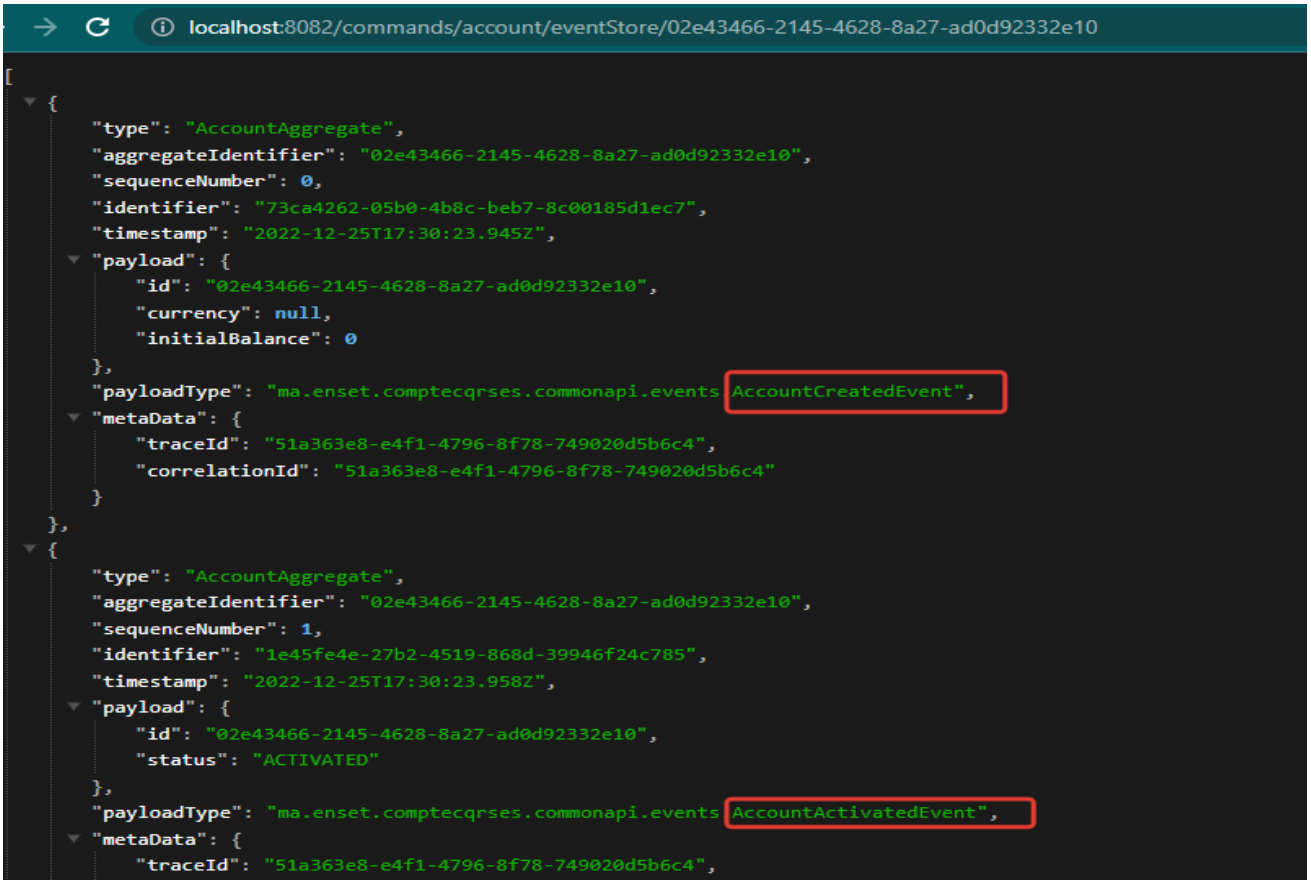
- **AccountActivatedEvent & EventSourcingHandler**



```java
    @EventSourcingHandler
    public void on(AccountCreatedEvent event){
        this.accoutId = event.getId();
        this.balance =event.getInitialBalance() ;
        this.status = AccountStatus.CREATED;
        this.currency = event.getCurrency() ;
        AggregateLifecycle.apply(new AccountActivatedEvent(
           event.getId(),
           AccountStatus.ACTIVATED
        ));
    }

    no usages
    @EventSourcingHandler
    public void on(AccountActivatedEvent event) {
        this.status = event.getStatus();
    }
```

- **Test (Postman)**





- **AccountCrediteddEvent**

```java
public class AccountCreditedEvent extends BaseEvent<String> {
    1 usage
    @Getter private double amount;
    1 usage
    @Getter private String currency;

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

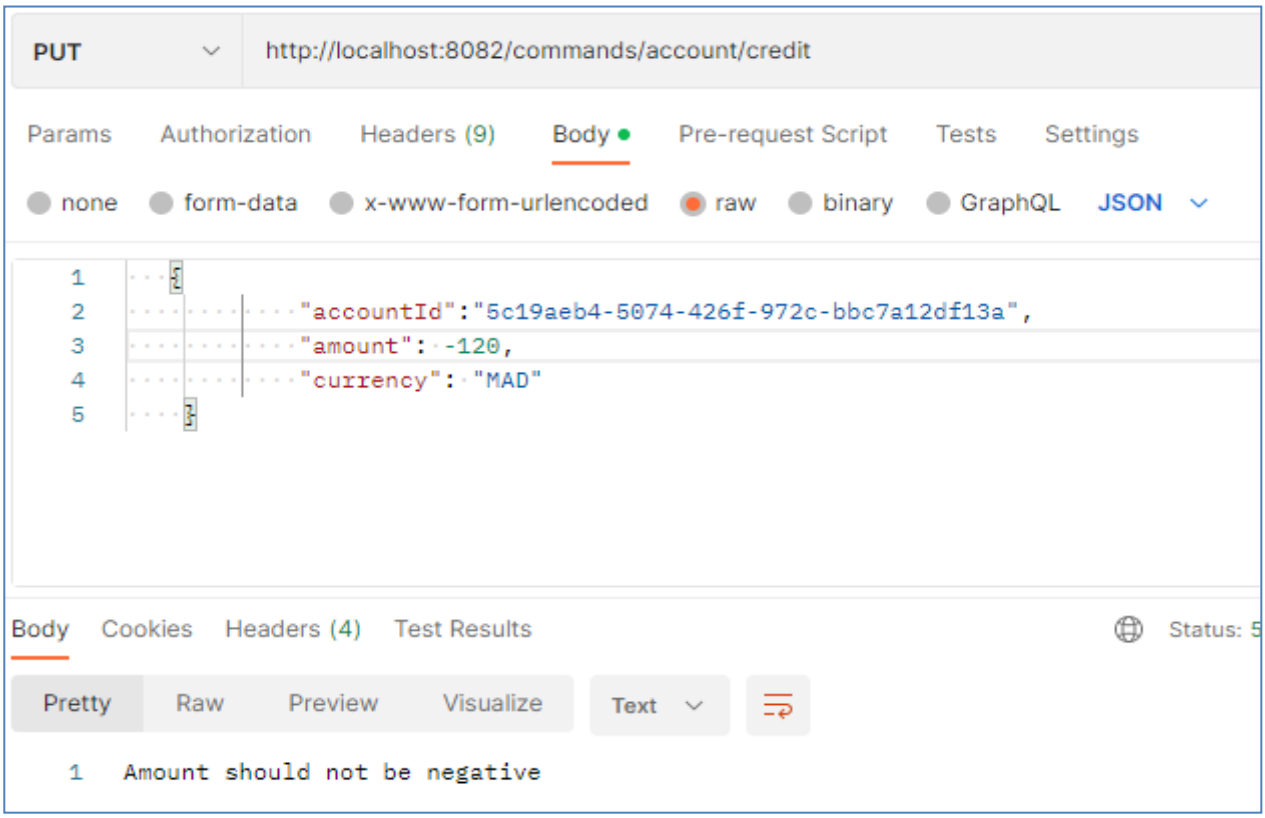
- **EventSourcingHandler**

```java
    @CommandHandler
    public void handle(CreditAccountCommand command) {
        if (command.getAmount() < 0) {
            throw new AmountNegativeException("Amount should not be negative");
        }

        AggregateLifecycle.apply(new AccountCreditedEvent(
                command.getId(),
                command.getAmount(),
                command.getCurrency()
        ));
    }
    no usages
    @EventSourcingHandler
    public void on(AccountCreditedEvent event) {
        this.balance += event.getAmount();
```

- **CreditAccount (Controller)**

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

- **Test (Postman)**




```
{
    "type": "AccountAggregate",
    "aggregateIdentifier": "5c19aeb4-5074-426f-972c-bbc7a12df13a",
    "sequenceNumber": 3,
    "identifier": "07c8f5df-500c-471c-bc76-b240d25dd18a",
    "timestamp": "2022-12-25T18:21:00.587Z",
    "payloadType": "ma.enset.comptecqrses.commonapi.events.AccountCreditedEvent",
    "metaData": {
        "traceId": "4a8e5190-2b18-4c43-b676-fcdac401d4c7",
        "correlationId": "4a8e5190-2b18-4c43-b676-fcdac401d4c7"
    },
    "payload": {
        "id": "5c19aeb4-5074-426f-972c-bbc7a12df13a",
        "amount": 122000,
        "currency": "MAD"
    }
}
```


- **AccountDebitedEvent**

```java
public class AccountDebitedEvent extends BaseEvent<String> {
    1 usage
    @Getter private double amount;
    1 usage
    @Getter private String currency;

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

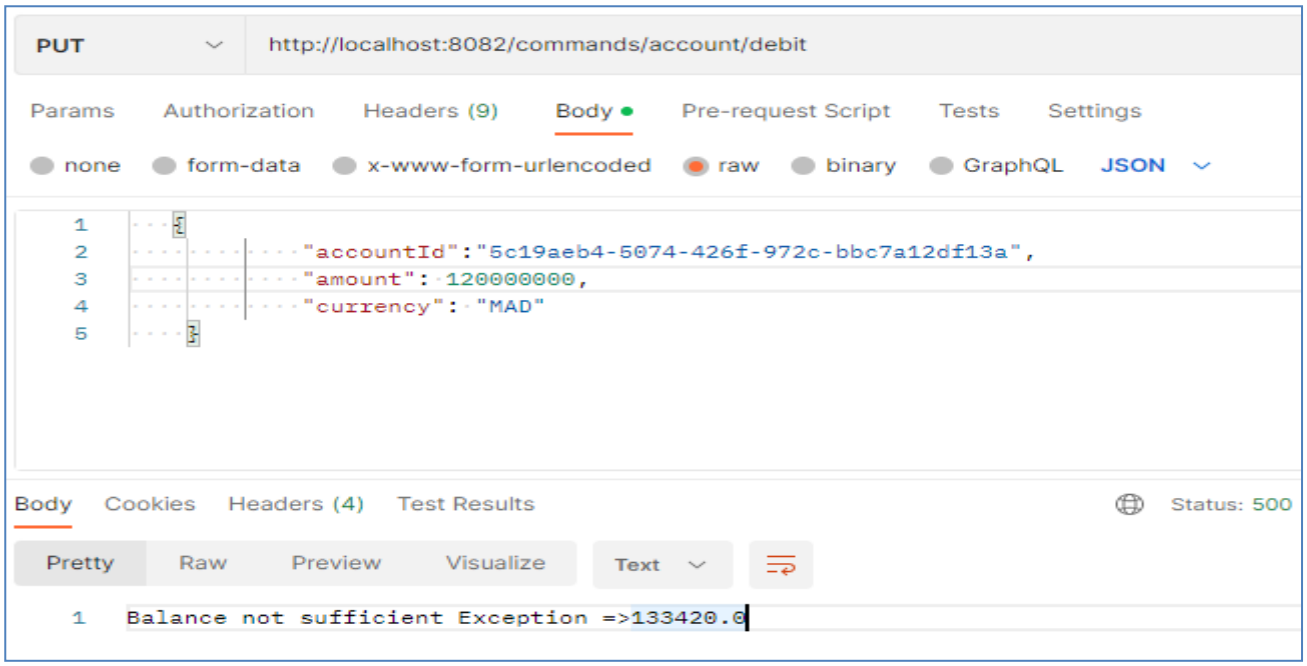
- **CommandHandler & EventSourcingHandler**

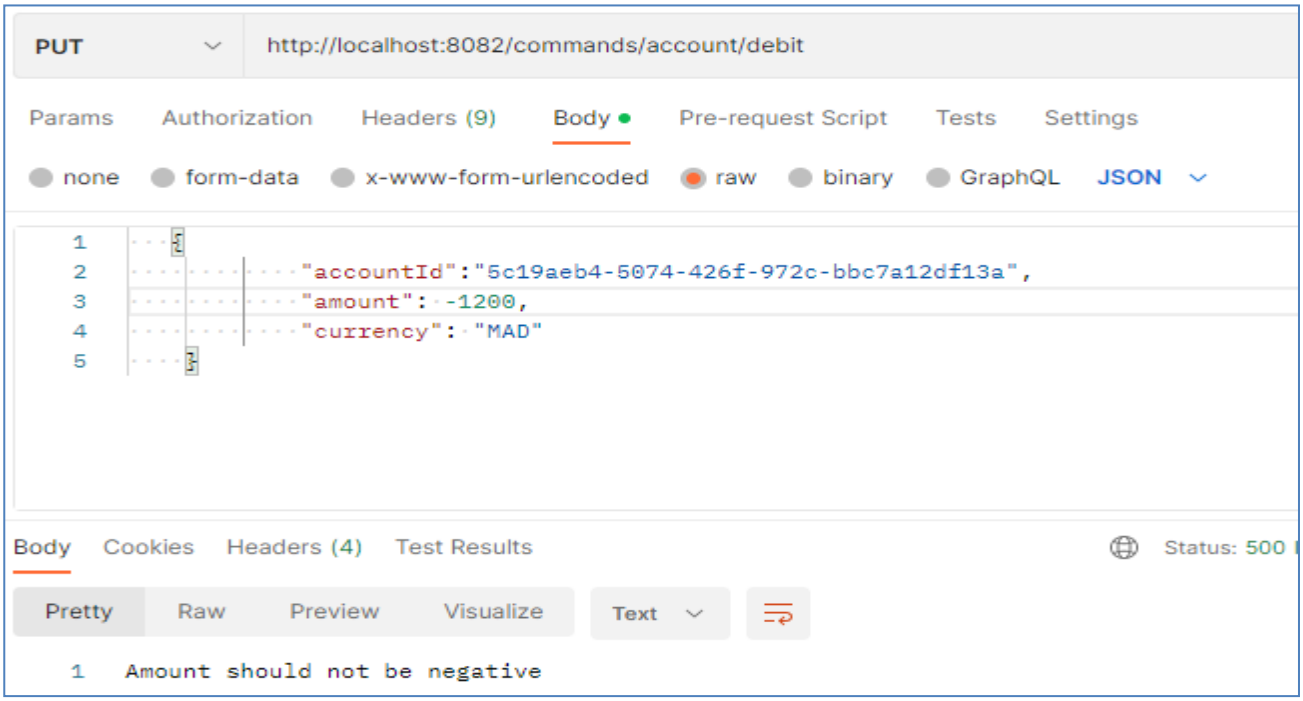
```java
    @CommandHandler
    public void handle(DebitAccountCommand command) {
        if (command.getAmount() < 0) {
            throw new AmountNegativeException("Amount should not be negative");
        }
        if(this.balance<command.getAmount()) throw new BalanceNotSufficientException("Balance not sufficient Exception =>"+balance);

        AggregateLifecycle.apply(new AccountDebitedEvent(
                command.getId(),
                command.getAmount(),
                command.getCurrency()
        ));
    }
    no usages
    @EventSourcingHandler
    public void on(AccountDebitedEvent event) {
        this.balance -= event.getAmount();
    }
```

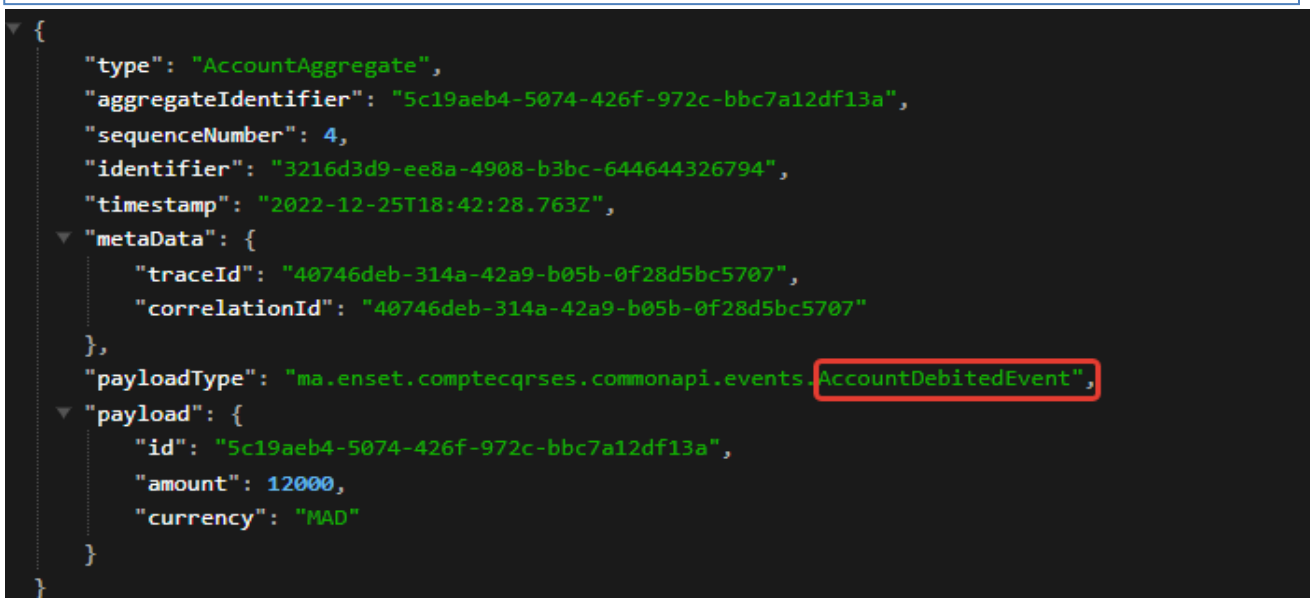
- **DebitAccount (Controller)**

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

- **Test (Postman)**







PARTIE 2 : Lecture

- **Entity Account**

```java
@AllArgsConstructor @NoArgsConstructor
public class Account {
    no usages
    @Id
    private String id;
    no usages
    private  double balance ;
    no usages
    @Enumerated(EnumType.STRING)
    private AccountStatus status;
    no usages
    private String currency ;
    no usages
    @OneToMany(mappedBy = "account")
    private Collection<Operation> operations;
```

- **Entity Operation**

```java
public class Operation {

    no usages
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    no usages
    private Date date;
    no usages
    private double amount;
    no usages
    @Enumerated(EnumType.STRING)
    private OperationType type ;
    no usages
    @ManyToOne
    private Account account;
}
```

- **Repository AccountRepository**

```java
public interface AccountRepository extends JpaRepository<Account, String> {

}
```

- **Repository OperationRepository**

```java
public interface OperationRepository extends JpaRepository<Operation, String> {

}
```

- **AccountServiceHandler (AccountCreatedEvent)**

```java
@EventHandler
public void on(AccountCreatedEvent event) {
    log.info("********************");
    log.info("AccountCreatedEvent received");
    Account account = new Account();
    account.setId(event.getId());
    account.setBalance(event.getInitialBalance());
    account.setCurrency(event.getCurrency());
    account.setStatus(event.getStatus());
    accountRepository.save(account);
}
```

```
2022-12-25 20:30:51.368  INFO 11516 --- [uery.service]-0] m.e.c.c.q.service.AccountServiceHandler  : AccountCreatedEvent received
2022-12-25 20:30:51.395  INFO 11516 --- [uery.service]-0] m.e.c.c.q.service.AccountServiceHandler  : ********************
2022-12-25 20:30:51.396  INFO 11516 --- [uery.service]-0] m.e.c.c.q.service.AccountServiceHandler  : AccountCreatedEvent received
2022-12-25 20:30:51.411  INFO 11516 --- [uery.service]-0] m.e.c.c.q.service.AccountServiceHandler  : ********************
2022-12-25 20:30:51.411  INFO 11516 --- [uery.service]-0] m.e.c.c.q.service.AccountServiceHandler  : AccountCreatedEvent received
```

- **AccountServiceHandler (AccountActivatedEvent, AccountDebitedEvent, AccountCreditedEvent)**

```java
@EventHandler
public void on(AccountActivatedEvent event) {
    log.info("****************************");
    log.info("AccountActivatedEvent received");
    Account account = accountRepository.findById(event.getId()).get();
    account.setStatus(event.getStatus());
    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.setAmount(event.getAmount());
    operation.setDate(new Date());
    operation.setType(OperationType.DEBIT);
    operation.setAccount(account);
    operationRepository.save(operation);
    account.setBalance(account.getBalance() - event.getAmount());
    accountRepository.save(account);
}
```

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

- **Controlleur pour la lecture(AllAccounts) et Controlleur pour la lecture (getAccountById)**

```java
@RequestMapping(path = "/query/accounts")
public class AccountQueryController {
    2 usages
    private QueryGateway queryGateway;

    no usages
    @GetMapping(path = "/allAccounts")
    public List<Account> accountList() {
        return queryGateway.query(new GetAllAccountsQuery(), ResponseTypes.multipleInstancesOf(Account.class)).join();
    }

    no usages
    @GetMapping(path="/getAccount/{id}")
    public Account getAccount(@PathVariable String id) {
        return queryGateway.query(new GetAccountQuery(id), ResponseTypes.instanceOf(Account.class)).join();
    }
}
```

```java
    @QueryHandler
    public List<Account> on(GetAllAccountsQuery query) {
        return accountRepository.findAll();
    }
    no usages
    @QueryHandler
    public Account on(GetAccountQuery query) {
        return accountRepository.findById(query.getId()).get();
    }
```