Cash Manager

Technical Documentation

Back-End-API

This document serves as an explanation of the functioning and realization of the api for our "cash-manager" application. We will explain through this document :

- The technical stack used to build the api.
- The different specifications of the application as well as their implementation and the way of doing chosen.
- Examples to illustrate concretely the use of the api.

Technical Stack :

- Java 11 with Intellij
- The project is build with Maven

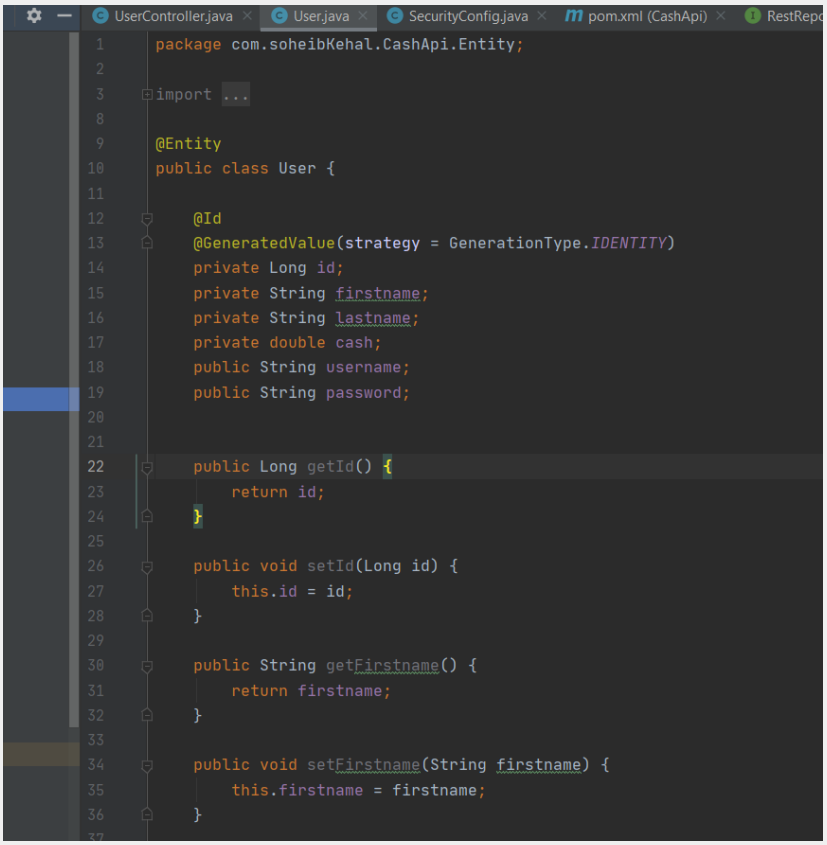
- Spring Boot 2.4.0 with some dependecies :
  - spring-boot-starter-web
  - spring-boot-starter-data-jpa
  - spring-boot-starter-security
  - Jjwt
  - spring-boot-starter-data-jdbc
  - H2 database
  - Junit

---

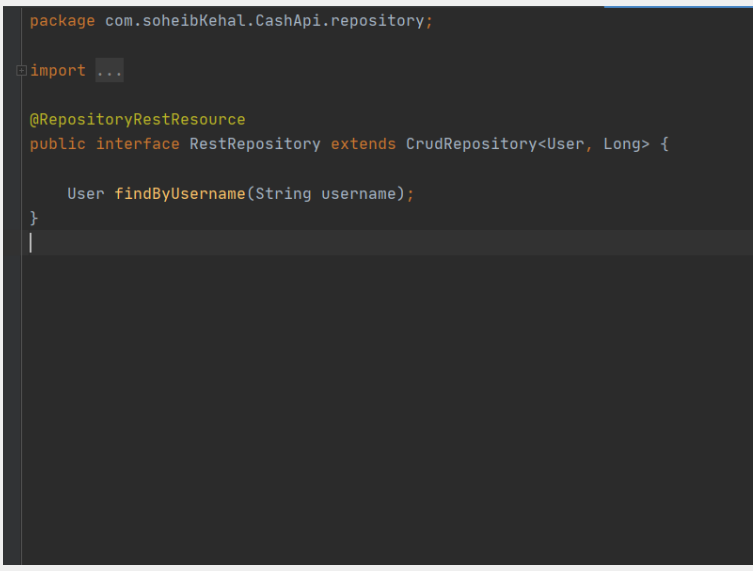
what does the application do ?

Basic CRUD functionalities :

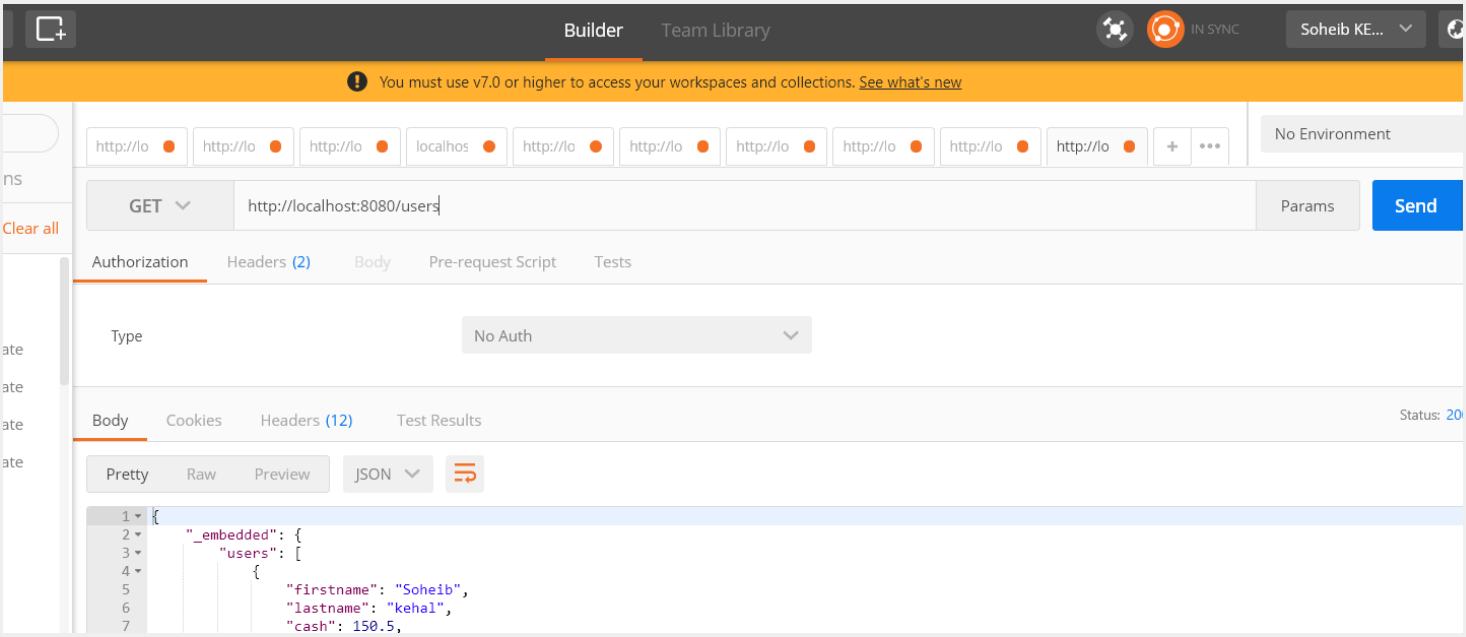
For this part we use the interfance Repository as a @RepositoryRestResource which inherits from CrudRepository we add User and the type of the id as type paramaters and thats it. We just have to add a model, for example user, and initialize a database and we can Get , Post , Delete and Put request.



Basic user model Class



User Repository Class



Just a basic example with a GETrequest that shows us all users stored on the database.

Authentification with JWT method :

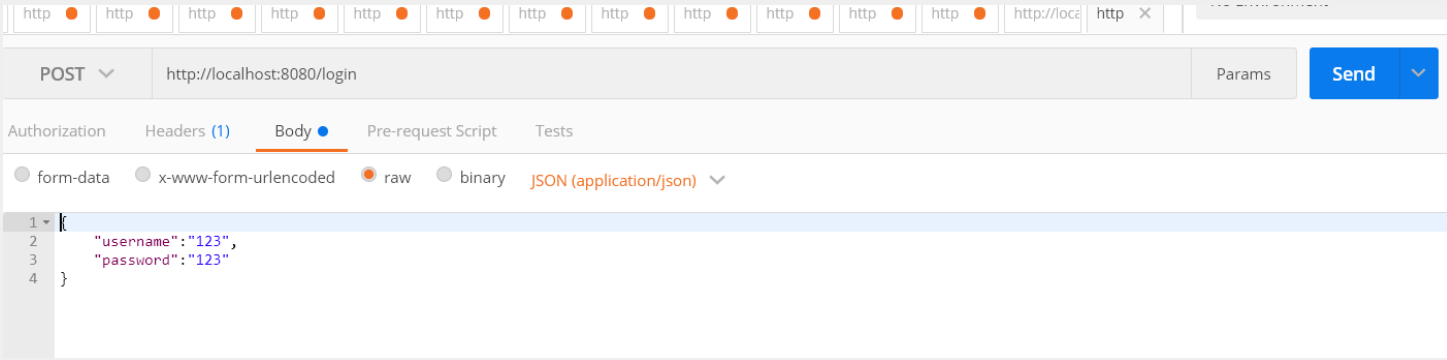
- accept or refuse authentication based on stored data (we first added the basic auth and after the JWT authentification)

```
private String secret = "soheib";
// class generique de manipulation de token avec plusieurs method differentes

public String extractUsername(String token) { return extractClaim(token, Claims::getSubject); }

public Date extractExpiration(String token) { return extractClaim(token, Claims::getExpiration)

public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {
    final Claims claims = extractAllClaims(token);
    return claimsResolver.apply(claims);
}
private Claims extractAllClaims(String token) {
    return Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody();
}

private Boolean isTokenExpired(String token) { return extractExpiration(token).before(new Date(

public String generateToken(String username) {
    Map<String, Object> claims = new HashMap<>();
    return createToken(claims, username);
}

private String createToken(Map<String, Object> claims, String subject) {

    return Jwts.builder().setClaims(claims).setSubject(subject).setIssuedAt(new Date(System.cur
            .setExpiration(new Date(System.currentTimeMillis() + 1000 * 60 * 60 * 10))
            .signWith(SignatureAlgorithm.HS256, secret).compact();
}

public Boolean validateToken(String token, UserDetails userDetails) {
    final String username = extractUsername(token);
    return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));
```

JWT token generation and handling class

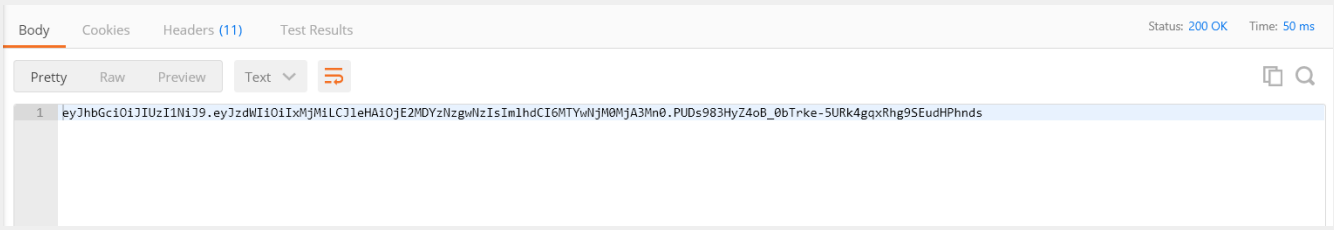
```java
@PostMapping("/login")
public String generateToken(@RequestBody AuthRequest authRequest) throws Exception {
    try {
        authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(authRequest.getUsername(), authRequest.getPa
        );
    } catch (Exception ex) {
        throw new Exception("inavalid username/password");
    }
    return jwtUtil.generateToken(authRequest.getUsername());
}
```

class controller to log in, after you authenticate yourself here you can hit any private endpoint.



To authenticate, hit the endpoint "/login" with a POST method with the username and password of an existing user

---



It will generate you a valid token for 10 hours

Now you can hit any endpoint with adding the headers Authorization with on value : Bearer and the token.

Stripe Payment System

we used stripe as a payment service as it offers a simple api, with a template already ready to use. A few adjustments are to be made but nothing very complicated.

```java
@Data
public class ChargeRequest {

    public enum Currency {
        EUR, USD;
    }
    private String description;
    private int amount; // cents
    private Currency currency;
    private String stripeEmail;
    private String stripeToken;
    public String getDescription() { return description; }
    public int getAmount() { return amount; }
    public Currency getCurrency() { return currency; }
    public String getStripeEmail() { return stripeEmail; }
    public String getStripeToken() { return stripeToken; }
    public void setDescription(String description) { this.description = description; }
    public void setCurrency(Currency currency) { this.currency = currency; }
```

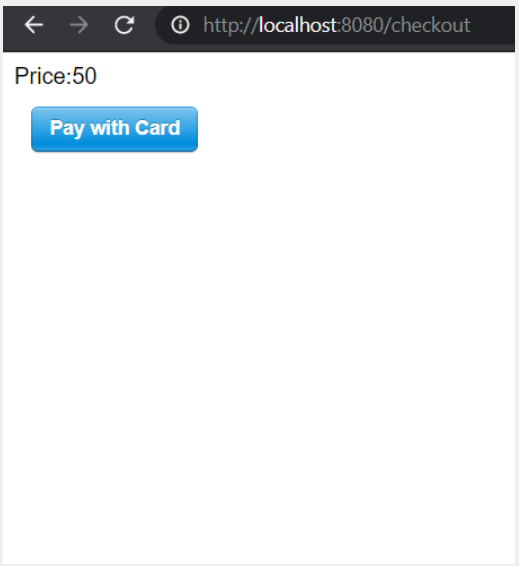
Charge Request class

```java
@Controller
public class CheckoutController {

    @Value("${STRIPE_PUBLIC_KEY}")
    private String stripePublicKey;
    public int myAmount = 50;

    @RequestMapping("/checkout")
    public String checkout(Model model) {
        model.addAttribute( s: "amount", o: myAmount * 100);
        model.addAttribute( s: "stripePublicKey", stripePublic
        model.addAttribute( s: "currency", ChargeRequest.Curre
        return "checkout";
    }
}
```

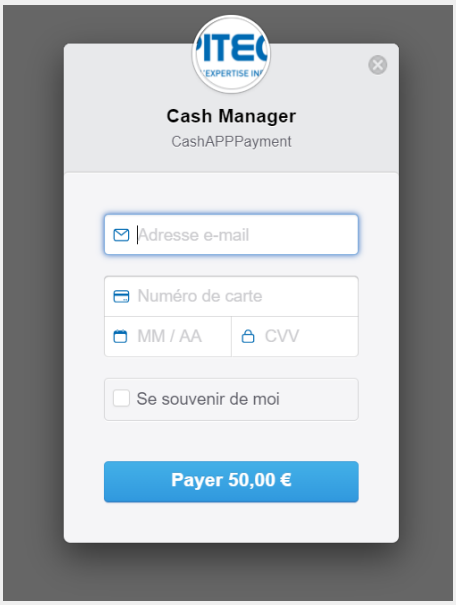
Checkout Controller class

```java
@Autowired
StripeService paymentsService;

@PostMapping("/charge")
public String charge(ChargeRequest chargeRequest, Model model) throws StripeEx
    chargeRequest.setDescription("Example charge");
    chargeRequest.setCurrency(Currency.EUR);
    Charge charge = paymentsService.charge(chargeRequest);
    model.addAttribute( s: "id", charge.getId());
    model.addAttribute( s: "status", charge.getStatus());
    model.addAttribute( s: "chargeId", charge.getId());
    model.addAttribute( s: "balance_transaction", charge.getBalanceTransaction()
    return "result";
}

@ExceptionHandler(StripeException.class)
public String handleError(Model model, StripeException ex) {
    model.addAttribute( s: "error", ex.getMessage());
    return "result";
}
```

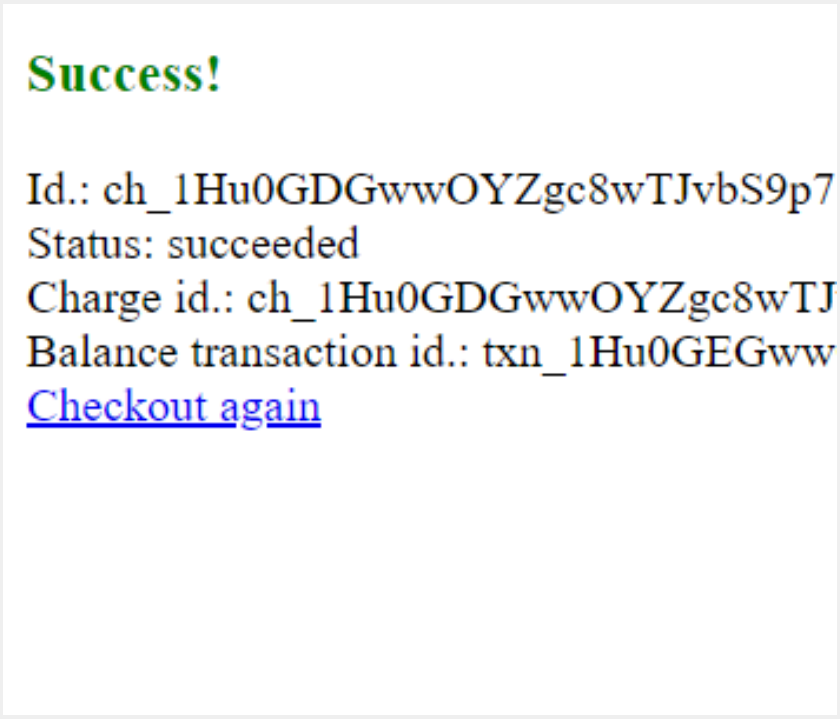
Payment Result class



Checkouthtml endpoint



Payment template



Success page

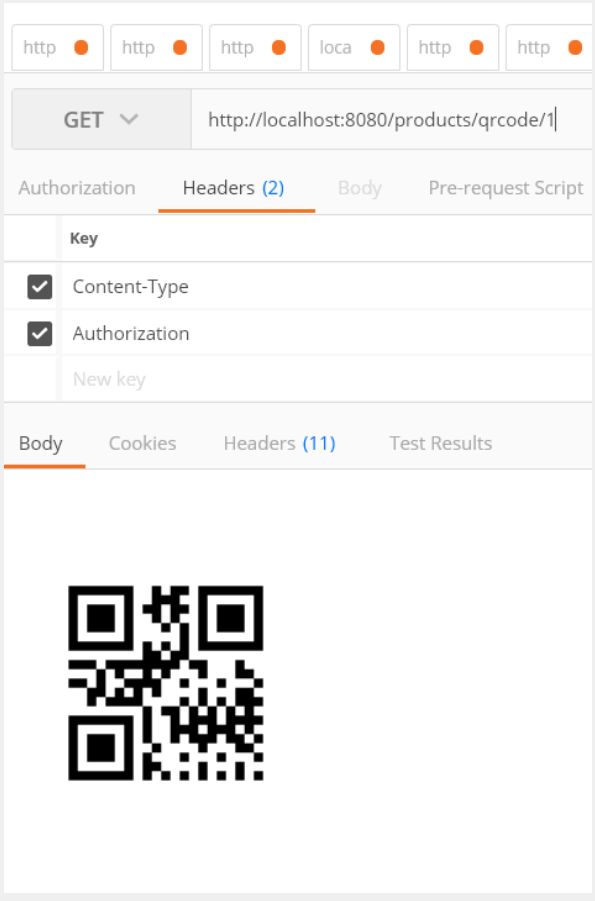
And more...

Qr code Generator :

we wanted to add a qr generator for each product in our database

```
public static byte[] getQRCodeImage(String text, int width, int height) {
    try {
        QRCodeWriter qrCodeWriter = new QRCodeWriter();
        BitMatrix bitMatrix = qrCodeWriter.encode(text, BarcodeFormat.QR_CODE, width, height);
        ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
        MatrixToImageWriter.writeToStream(bitMatrix, format: "png", byteArrayOutputStream);
        return byteArrayOutputStream.toByteArray();
    } catch (Exception e) {
        return null;
    }
}

}
```

Qr code Generator class



Qr code generated and endpoint

Some endpoints :

- localhost:8080/users/{id}
- localhost:8080/products/{id}
- localhost:8080/products/qrcode/{id} (get qr code of id object)
- localhost:8080/checkout (endpoint for payment)

Cash Manager

Technical Documentation

Project done by :

Back-end : Kehal Soheib |Carthic Sekar
Front-end : Valentin Noel | Logan Lagar-Jumeau