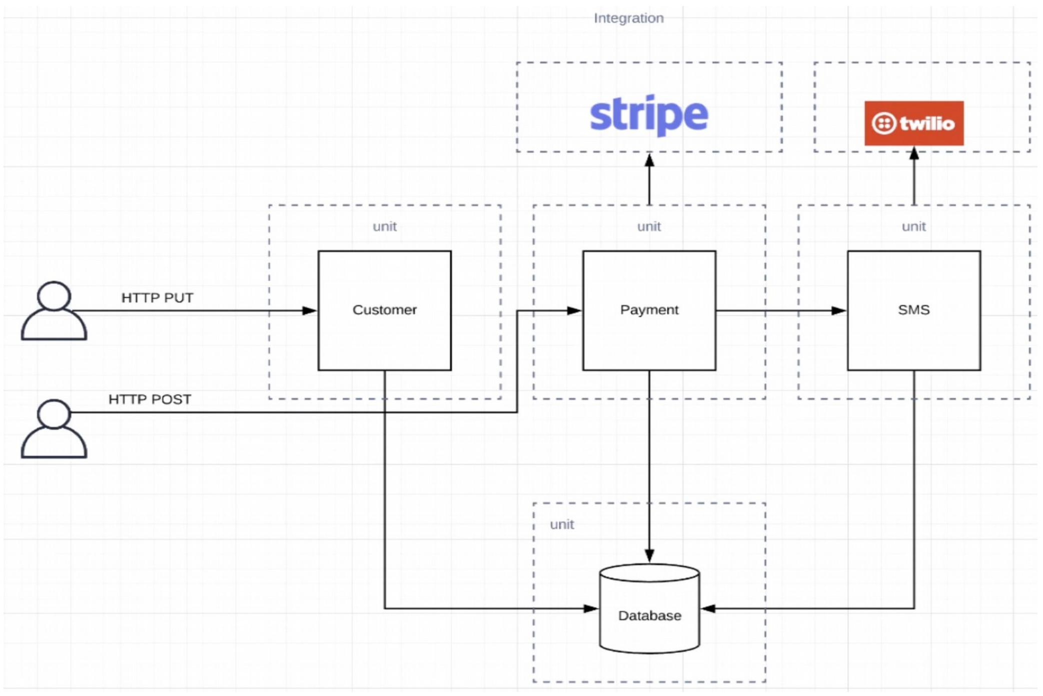
SECTION 06 - INTEGRATION TESTING

Index

- Intro

- In integration testing we test that all the pieces work together whereas the unit test focuses on the unit itself.

- Integration test means that you have the application up and running.



- Mock Stripe service

```
@ConditionalOnProperty(
        value = "stripe.enabled",
        havingValue = "false"
        )
```

- Makes that with integration testing stripe is disabled so that every time we run our application Spring will initialize the mock service being injected instead of StripeService class.

- We can read it as: This bean will be initialized whenever the stripe.enabled property has the value of *false*.

Procedure:

→ From Stripe service:

```
@Service
@ConditionalOnProperty(
        value = "stripe.enabled",
        havingValue = "true"
        )
public class StripeService implements CardPaymentCharger {
```

→ From Stripe mock service:
```
@Service
@ConditionalOnProperty(
        value = "stripe.enabled",
        havingValue = "false"
        )
public class MockStripeService implements CardPaymentCharger {
```

→ From application.properties
```
stripe.enabled=false
```

According to the environment I set stripe.enabled property as I need it.

- PaymentIntegrationTest class

- Add @SpringBootTest annotation on top of the class so whenever I run a test inside an integration test class, Spring will start the entire application.

- If I do:
  → Given:

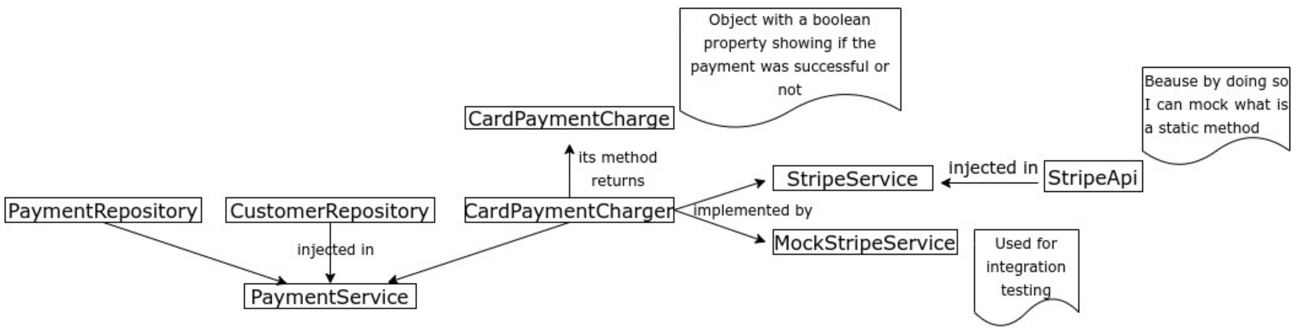
  → CustomerRegistrationController:
  ```
  @RestController
  @RequestMapping("api/v1/customer-registration")
  public class CustomerRegistrationController {

      @PutMapping
      public void registerNewCustomer(
              @Valid @RequestBody CustomerRegistrationRequest request) {
          System.out.println(request);
      }
  }
  ```

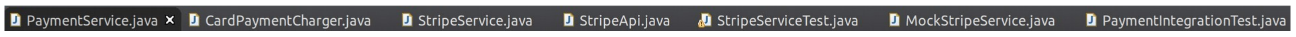
  → PaymentIntegrationTest:
  ```
  @Autowired
  private CustomerRegistrationController customerRegistrationController;

  @Test
  void itShouldCreatePaymentSuccessfully() {
      // given
      UUID customerId = UUID.randomUUID();
      Customer customer = new Customer(customerId, "james", "123");

      customerRegistrationController.registerNewCustomer(...);
  ```

- By injecting CustomerRegistrationController in the test class I'm not testing the API, instead I'm simply invoking the method directly. What I want to test is when I perform a PUT request to api/v1/customer-registration.

- In itShouldCreatePaymentSuccessfully() test case I "break a rule" by using:
  ```
  @Autowired
  private PaymentRepository paymentRepository;
  ```

  in PaymentIntegrationTest because I don't have any endpoint to get a customer given its id.


Object with a boolean property showing if the payment was successful or not
Beause by doing so I can mock what is a static method
CardPaymentCharge
its method returns
StripeService
injected in
StripeApi
PaymentRepository
CustomerRepository
CardPaymentCharger
implemented by
injected in
MockStripeService
Used for integration testing
PaymentService


PaymentService.java
CardPaymentCharger.java
StripeService.java
StripeApi.java
StripeServiceTest.java
MockStripeService.java
PaymentIntegrationTest.java