Dear Maybank,

Sorry that I were late. But anyhow, I still would like to share out the things that i

prepared from last weekend (Saturday and Sunday) and (Monday and Tuesday for documentation

while busy on working as usual). Due to lack of times, and also having some skillset

issue on Spring Boot. Therefore, for me to implement it will become one of the greater

challenge in this trial and I would take this opportunity to boost my knowledge! Thanks again,

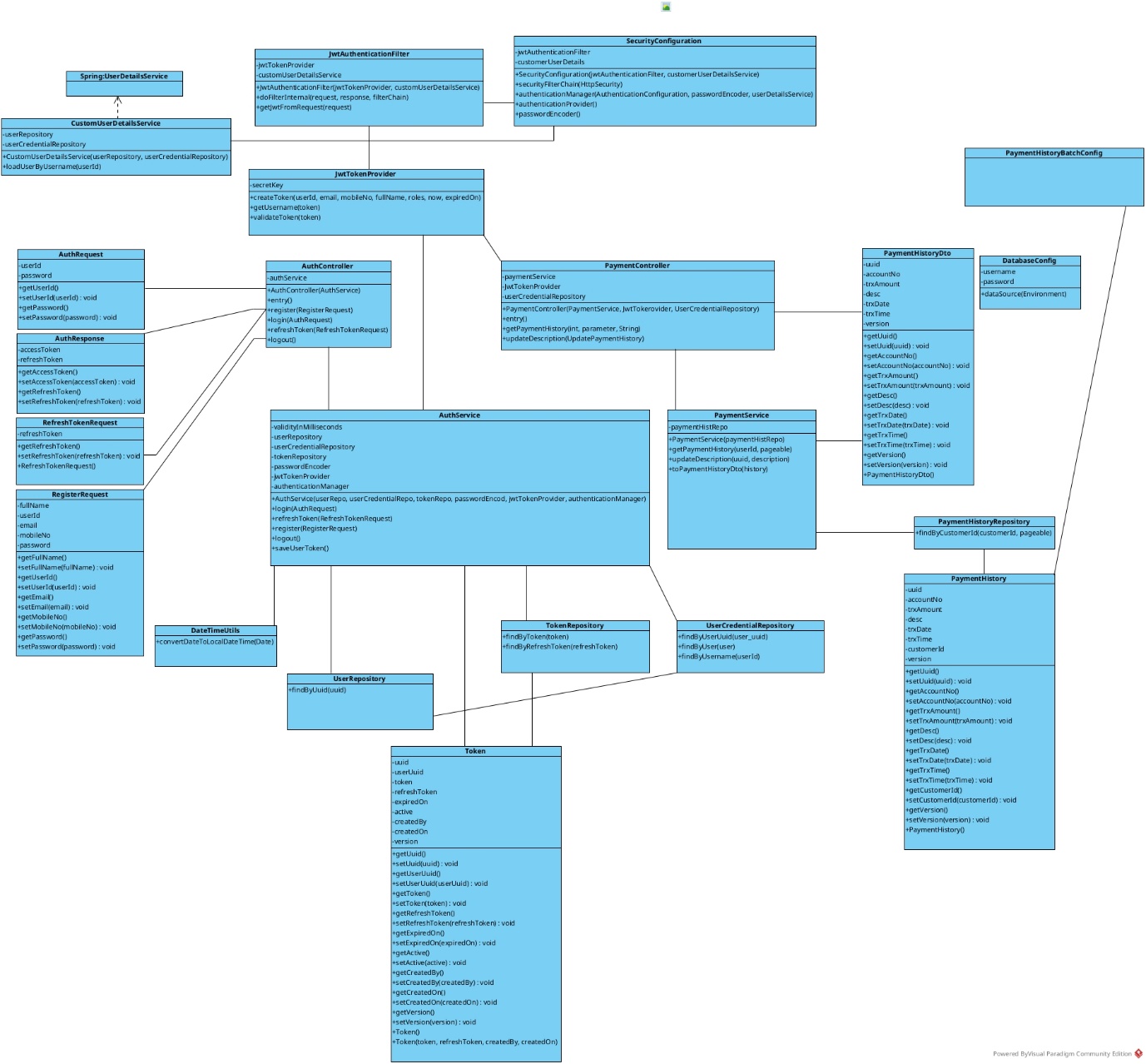
Maybank!

The Reason why I would like to choose Spring Boot to applied in this Demo Project?

|  |
| --- |
| 1. It is one of the famous Java framework that have been using through the IT Market.  Unfortunately, from my current project, they are not using it. So i can't maximize my  knowledge in Spring Boot but just applied those knowledge that i used to learn in the  past. So i want to brought this opportunity to growth me up by challenge by building  up a entire program. Besides, there are a lot of spring features can build up  ease of use and easy adaption due to community.  2. Spring have their own great standard quality. So it also help to handling very well  for Singleton Design Pattern. So we don't have to waste memory by keep defining an object.  example,  Case (a)  PaymentService paymentService = new PaymentService();  List<PaymentHistory> historys = paymentService.getHistory();  @Autowired  PaymentService paymentService;  Case (b)  // We don't have to define it again, we already autowired to get the Singleton.  Lets get back to what I have planned and have been doing. |

**Initial Plan**

|  |
| --- |
| **1. I have think of using Factory Design Pattern to make it into Multiple Payment Type. (Failed to delivery, due to Time factor since it will result in creating more API)**  Parent -> PaymentInfo  Child -> Fund Transfer  Child -> Bill Payment  Child -> 3rd Party Fund Transfer  Child -> ATM Withdrawal  example.  private Class FundTransferPayment extends PaymentInfo {  @Id  @Column("")  private String uuid;  private BigDecimal amount;  private String accountNo;  private String desc;  private String customerId;  private LocalDate trxDate;  private LocalTime trxTime;  }  and So on for the rest of the payment.  **2. Handling for Authentication/Authorization for actual BAU features. (delivered)**  - Prepare Basic Register  -> Encode(Password)  - Prepare Basic Login  -> Do UserId and Password Matching for build JWT Token  - Refresh Token  -> Regenerate another AccessToken  - Make use of this JWT as our Authentication and Authorization  - Handling for validity of the login-user  - Utilize Spring Security for disabled csrf (due to using Stateless) and cors for manipulate  the permission of the API URL. However, also using Filter built in from Spring to handling  for tokenizer validation.  - Reason why are we using JWT, first of it, it is one of the famous token being use in  out there.  **3. Spring Batch (delivered)**  - Prepare a reader, processor and writer to perform the Job.  Reader -> Pick up the List of data from the datasource.txt, and first line are not  required to insert, so we are going to avoid reading the file line of the text file,  and pre-define the column from the datasource and pair with the database. We did utilize  the FieldMapper to perform dataType Compatibility  Processor -> Did Logger with some validation  Writer -> As per main purpose of Spring Batch, insert processed data and insert into Database.  Scheduling -> I set up every 5 minutes will trigger it via Fixed Rate instead of Cron Job for easy trigger purpose.  **4. Selection for Database to be use (delivered)**  - Since we are performing a demo project, so to avoid complicate database setup. I aim to  use some local and simple database. H2 Database, and why i am using the old version of it,  based on the dbeaver i used, it only able to read that for my testing.  **5. Utilize Database versioning (delivered)**  - We have been using liquibase for the auto-update DML script. So it can be more useful for  employee who step in and work with catch-up time quickly and without worry about the sql causing  unable to start server and slow down the entire process. I'd understand there are many things  to learn when we just switch into a whole new environment, it is crucial to everyone.  **6. Haven't provide an encrypted password and backend decryption (failed to delivery)**  - When come from Front End, they would need to prepare JSEncrypt (RSA or others) to encrypted with Public Key during login and register and  provide the using Private Key to decrypted it back to plain password during validation.  **7. Partially cover with some error handling (Partially Delivered)**  - Whenever we hit error, we might need return a meaningful exception error.  **8. Application.properties Password Suppose to be Encrypted (failed to delivery)**  - Password displayed in anywhere are not suppose to be plain. |

Class Diagram

Activity Diagram

