KKCT Union Application

**กลุ่มก้อนชายไทย กลุ่มที่ 10**

**สมาชิก**

| | | | |
|---|---|---|---|
| นาย จักริน | จอนจำรัส | รหัสนักศึกษา | 63010126 |
| นาย จิรกานต์ | กุลสิงห์ | รหัสนักศึกษา | 63010136 |
| นาย จิรภัทร | แก้วส่งแสง | รหัสนักศึกษา | 63010139 |
| นาย ชญานิน | เลียงจินดาถาวร | รหัสนักศึกษา | 63010177 |
| นาย ชนสรณ์ | จึงมาริศกุล | รหัสนักศึกษา | 63010185 |
| นาย ชินพัฒน์ | ศิริยาใจ | รหัสนักศึกษา | 63010231 |
| นาย ฐานพัฒน์ | สิทธิพรชัยสกุล | รหัสนักศึกษา | 63010256 |
| นาย ณพงศ์ | เคหะ | รหัสนักศึกษา | 63010277 |
| นาย ณภัทร | จิรารัตนกุลชัย | รหัสนักศึกษา | 63010279 |
| นาย ภาสกร | คงบุญเกียรติ | รหัสนักศึกษา | 63010750 |

**เสนอ**

ดร.ปริญญา เอกปริญญา

รายงานฉบับนี้เป็นส่วนหนึ่งของรายวิชา 01076024 Software Architecture And Design

สาขาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

## Content

## Proposal

1. Problem Statement

- ช่องทางการตรวจสอบความน่าเชื่อถือเกี่ยวการทำธุรกรรมทางการเงินมีน้อยและไม่สะดวกต่อผู้ใช้งาน
- การรายงานผลเครดิตผิดพลาด
- การตรวจสอบเครดิตมีความโปร่งใสต่ำ
- ความล่าช้าในการจัดเก็บและแยกประเภทธุรกรรมรายบุคคล
- ความล่าช้าในการตรวจสอบเครดิต

2. Solution to Solve/Relieve

- ลดการส่งข้อมูล credit ให้ลูกค้าผ่านทางบริษัทที่ทำธุรกรรมทางด้านการเงิน เป็นแสดงผลผ่านทาง application โดยตัว user สามารถ check profile เองได้ เพิ่มความสะดวก และลดการเสียเวลาในการทำธุรกรรมต่างๆที่มีความสำคัญของ user ได้
- เปลี่ยนวิธีการเช็ค credit โดยจากปกติบริษัทที่ทำธุรกรรมทางด้านการเงินจะต้องส่งรายละเอียดการทำธุรกรรมให้กับ บริษัทข้อมูล credit bureaus แต่เราสามารถเชื่อมโยงการทำธุรกรรมของ user กับบริษัทที่ทำธุรกรรมทางด้านการเงิน ซึ่งเราสามารถที่จะ update credit ของ user ได้รวดเร็ว และสะดวกมากขึ้น

## Functionality

- ลูกค้าสามารถตรวจสอบ credit ของตนได้
- ลูกค้าสามารถตรวจสอบรายการที่ unpaid ได้
- ลูกค้าสามารถตรวจสอบ log ของตนได้
- ผู้ประกอบการสามารถตรวจสอบ credit ของลูกค้าที่ต้องการทำธุรกรรมด้วยได้
- ผู้ประกอบการสามารถบันทึกการทำธุรกรรมได้
- ผู้ประกอบการสามารถอัพเดตธุรกรรมได้

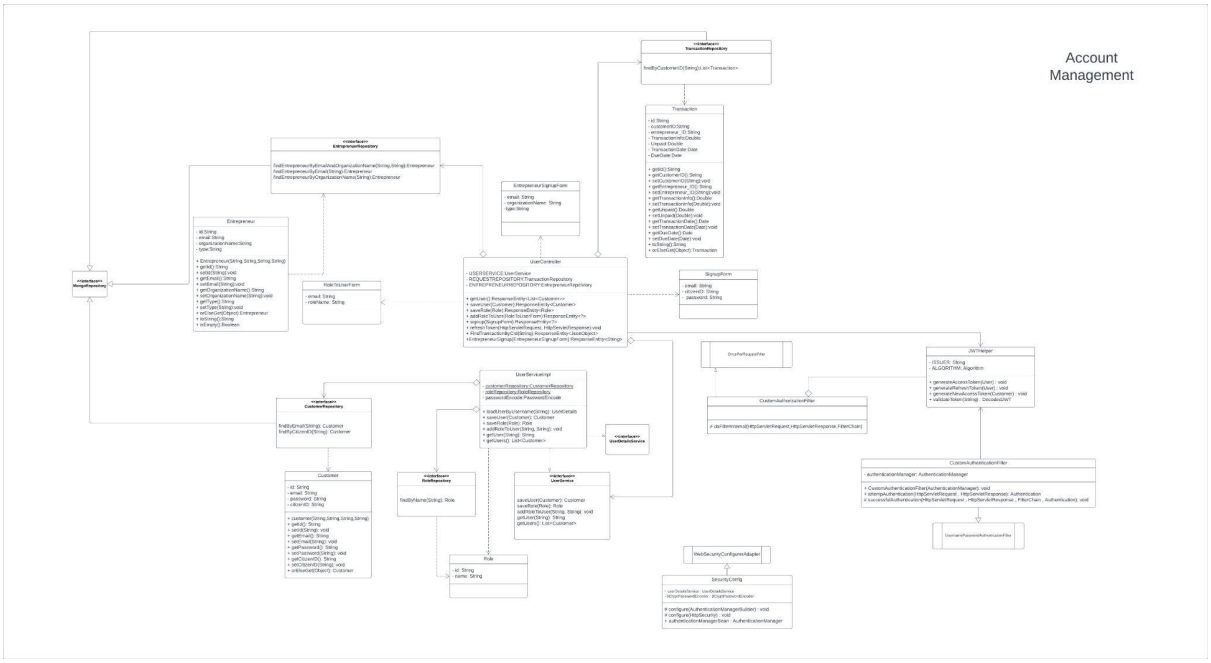
## Software Architecture

Client Server โดยติดต่อด้วย Rest API

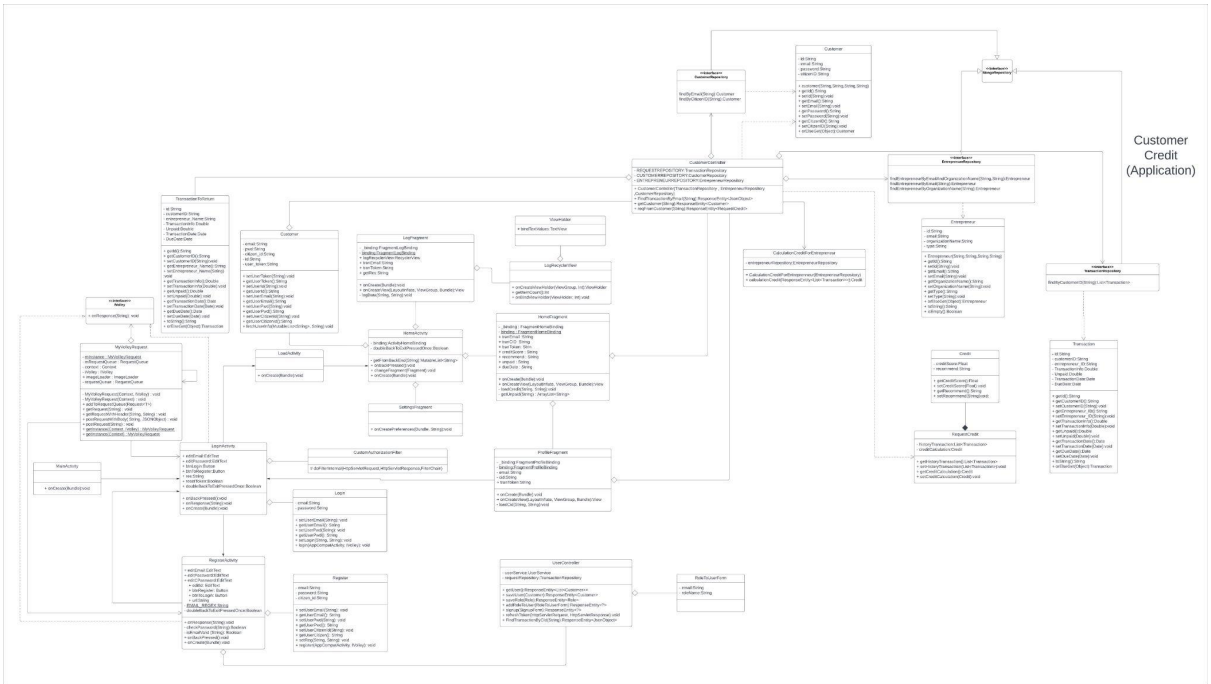
โครงสร้างหลักๆของเราจะประกอบไปด้วยฝั่งไคลเอนต์ซึ่งก็คือตัวแอพพลิเคชันและฝั่งหลังบ้าน (Database) โดยจะมีการจัดการงานสลับกันไปมา โดยฝั่งไคลเอนต์จะรับผิดชอบงานด้าน Presentation Logic เช่นการแสดงผลข้อมูลเครดิตลูกค้าและประวัติธุรกรรม และส่วนของฝั่งหลังบ้านจะมีหน้าที่ในการรับผิดชอบในเรื่องของ Data Access Logic และ Data Storage เช่นการ Update ข้อมูล

## Software Designs

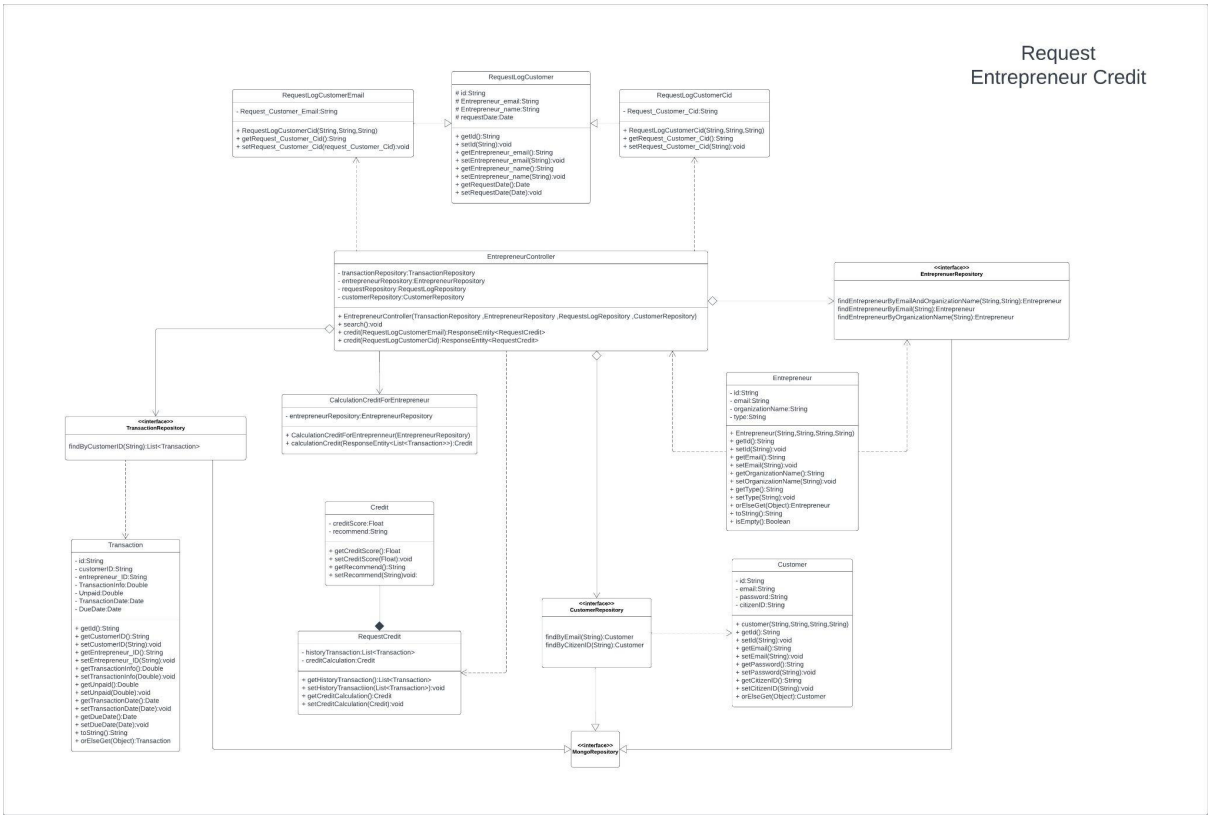
1. UML ของ Domain Account Management



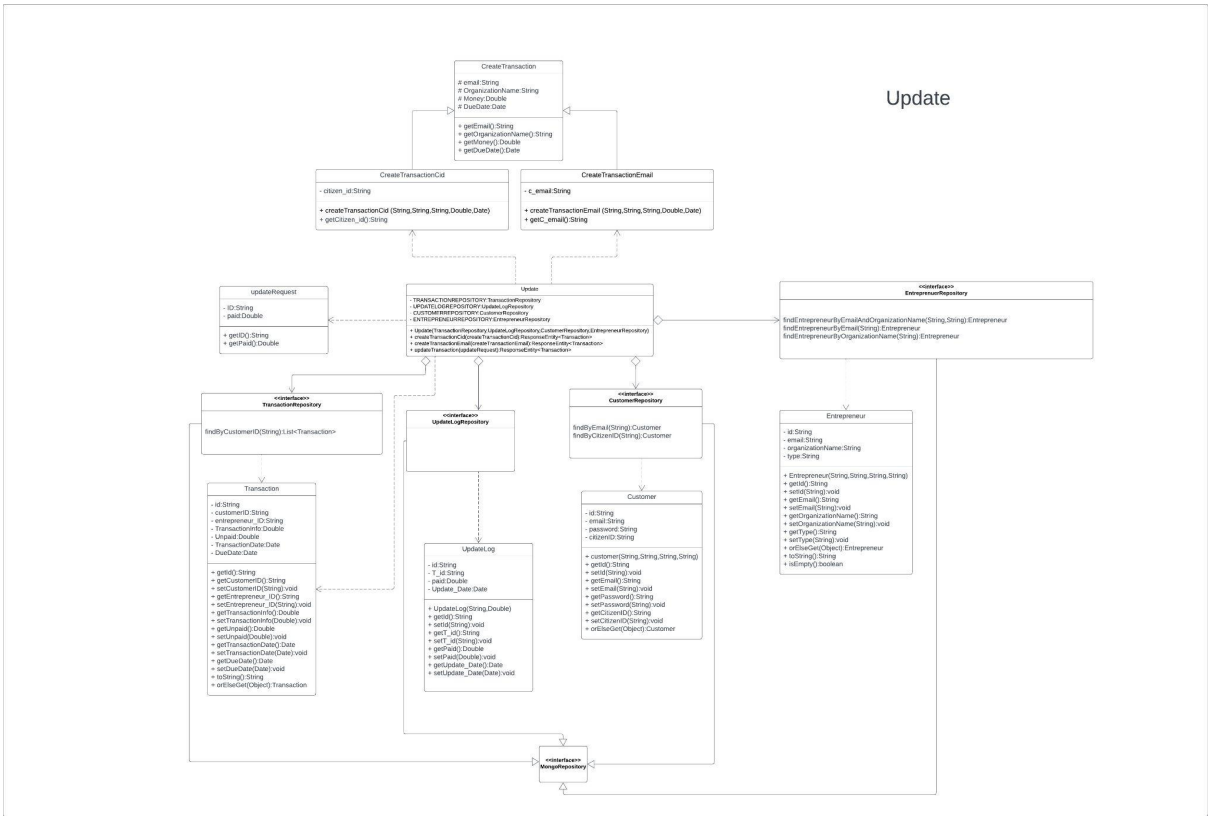

2. UML ของ Domain Request Customer Credit



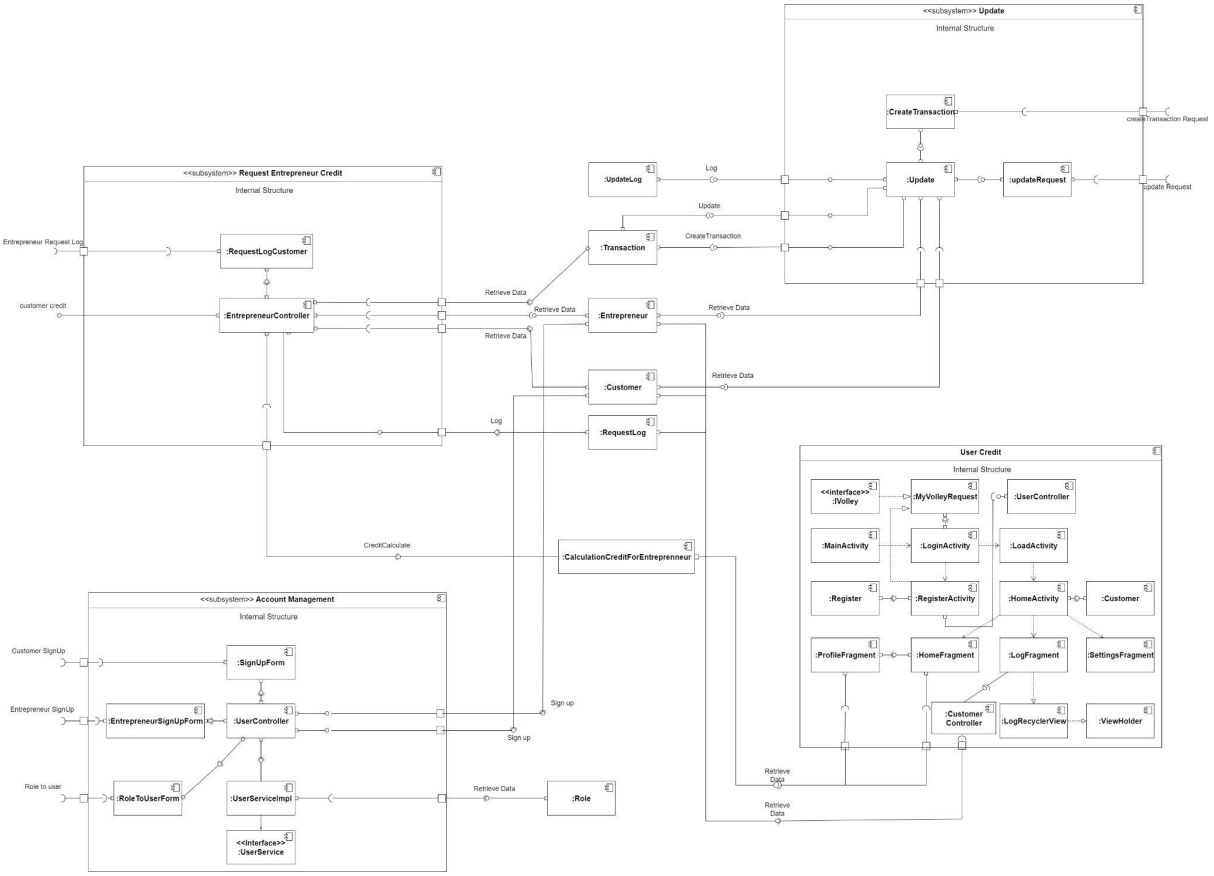
3. UML ของ Domain Request Entrepreneur Credit



4. UML ของ Domain Update



5. Component Diagram ของ Project



## Design patterns

1. Singleton Pattern (MyVolleyRequest.kt)

**ปัญหาที่พบ**

- ต้องการดึงค่าจากฐานข้อมูลในหลายๆหน้า

**วิธีการแก้ไขปัญหาด้วย Design Pattern**

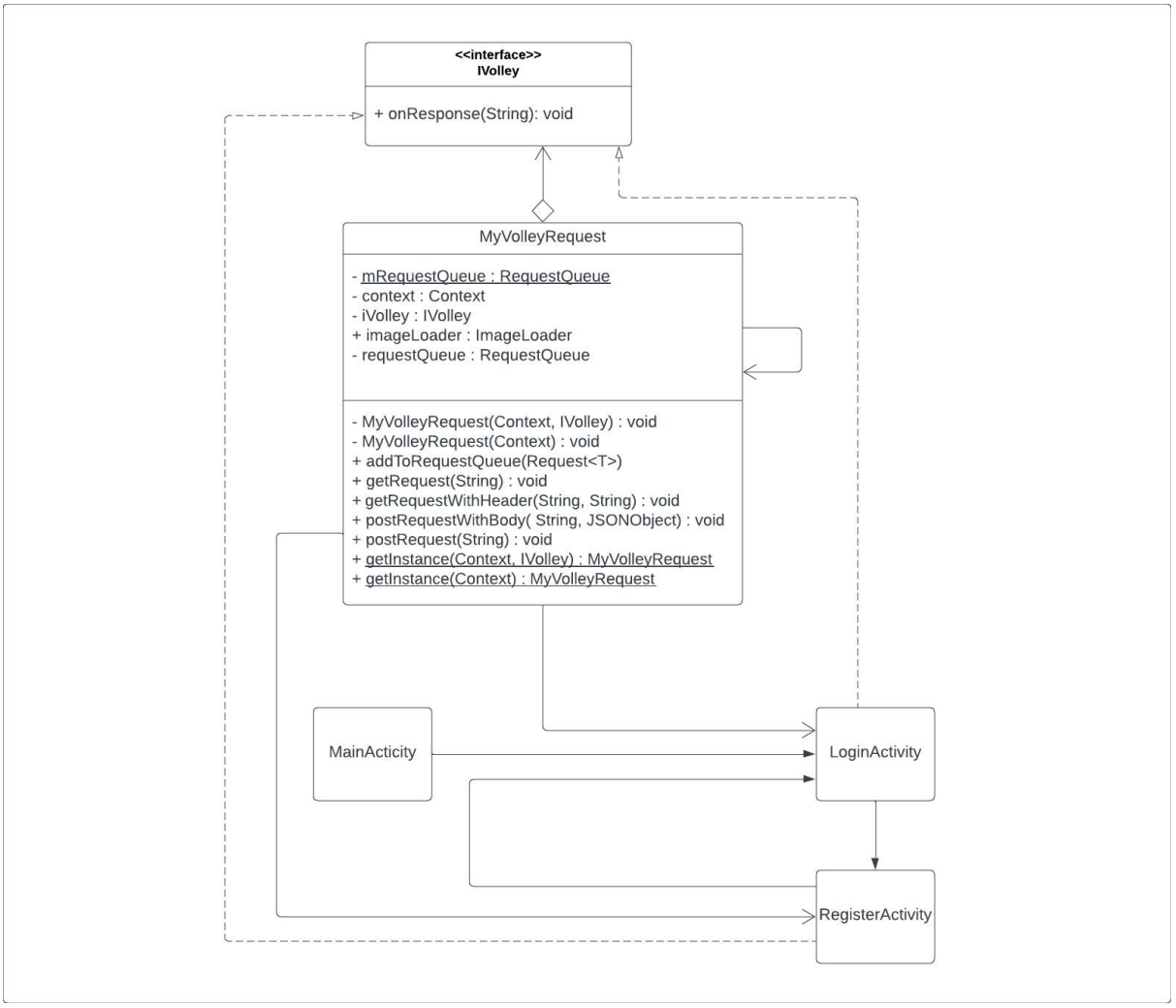
- สร้าง Class ที่มี Object สำหรับดึงข้อมูลจากฐานข้อมูลซึ่งสามารถเรียกใช้ได้ และ มีเพียงตัวเดียว

**Code ส่วนที่ใช้ Design Pattern**

```
private constructor(context: Context,iVolley: IVolley){
    this.context = context
    this.iVolley = iVolley
    mRequestQueue = requestQueue
    this.imageLoader = ImageLoader(mRequestQueue, object :
ImageLoader.ImageCache {
        private  val mCache = LruCache<String,Bitmap>(10)
        override fun getBitmap(url: String?): Bitmap? {
            return mCache.get(url)
        }

        override fun putBitmap(url: String?, bitmap: Bitmap?) {
            mCache.put(url,bitmap)
        }

    })
}
private constructor(context: Context){
    this.context = context
    mRequestQueue = requestQueue
    this.imageLoader = ImageLoader(mRequestQueue, object :
ImageLoader.ImageCache {
        private  val mCache = LruCache<String,Bitmap>(10)
        override fun getBitmap(url: String?): Bitmap? {
            return mCache.get(url)
        }

        override fun putBitmap(url: String?, bitmap: Bitmap?) {
            mCache.put(url,bitmap)
        }

    })
}
```

```
companion object{
    private  var mInstance : MyVolleyRequest? = null
    @Synchronized
    fun getInstance(context: Context) : MyVolleyRequest{
        if(mInstance == null){
            mInstance = MyVolleyRequest(context)
        }
        return  mInstance!!
    }
    @Synchronized
    fun getInstance(context: Context,iVolley: IVolley) :
MyVolleyRequest{
        if(mInstance == null){
            mInstance = MyVolleyRequest(context,iVolley)
        }
        return  mInstance!!
    }
}
```

## UML Diagram ส่วนที่ใช้ Design Pattern



2. Facade (Update.java)

**ปัญหาที่พบ**

- การเปลี่ยนหน้าด้วย Intent ในขณะที่ส่งข้อมูลเดิมบ่อยๆ ทำให้ performance เครื่องลดลงการ Update และสร้าง Transaction มีตรรกะที่ซับซ้อน

**วิธีการแก้ไขปัญหาด้วย Design Pattern**

- รวมการเรียกใช้งานอยู่ใน Controller เดียว ที่มีชื่อว่า Update

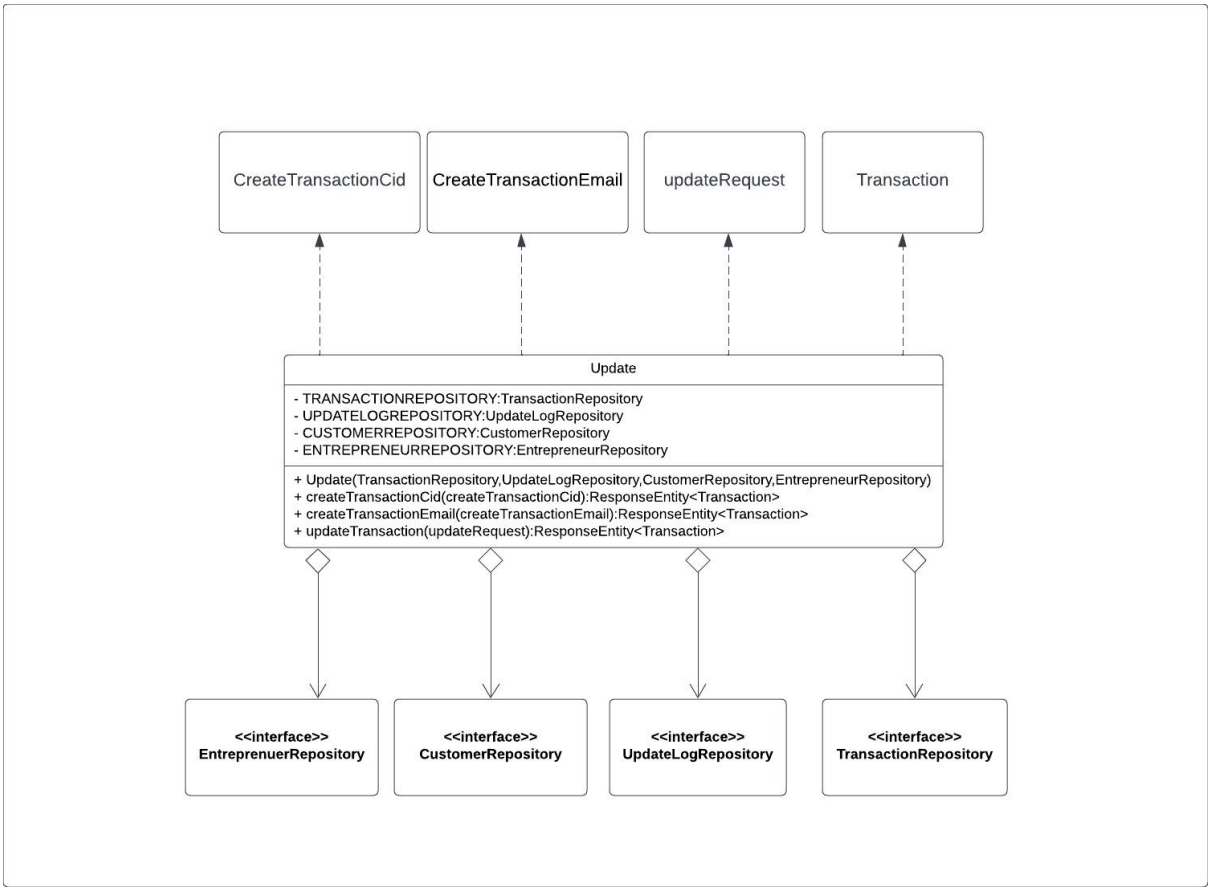
**Code ส่วนที่ใช้ Design Pattern**

```
public class Update {
    private final TransactionRepository transactionRepository;
    private final UpdateLogRepository updateLogRepository;
    private final CustomerRepository customerRepository;
    private final EntrepreneurRepository entrepreneurRepository;

    @Autowired
    public Update(TransactionRepository transactionRepository,
UpdateLogRepository updateLogRepository,CustomerRepository
customerRepository,EntrepreneurRepository entrepreneurRepository){
        this.transactionRepository = transactionRepository;
        this.updateLogRepository = updateLogRepository;
        this.customerRepository = customerRepository;
        this.entrepreneurRepository = entrepreneurRepository;
    }
```

```
@PostMapping("/createTransaction_CustomerCid")
public ResponseEntity<Transaction> createTransactionCid(@
RequestBody createTransactionCid ct){
    Customer customer = customerRepository.findByCitizenID(ct.
getCitizen_id());
    String customerID = customer.getId();
    System.out.println(ct.getEmail());
    System.out.println(ct.getOrganizationName());
    Entrepreneur entrepreneur = entrepreneurRepository.
findEntrepreneurByEmailAndOrganizationName(ct.getEmail(),ct.
getOrganizationName());
    String entrepreneurID = entrepreneur.getId();
    if (customerID == null ||entrepreneurID == null) {return new
ResponseEntity<Transaction>(HttpStatus.NOT_FOUND);}
    Transaction t = new Transaction(customerID, entrepreneurID, ct.
getMoney(), ct.getDueDate());
    return new ResponseEntity<Transaction>(transactionRepository.
save(t),HttpStatus.OK);
}

@PostMapping("/createTransaction_CustomerEmail")
public ResponseEntity<Transaction> createTransactionEmail(@
RequestBody createTransactionEmail ct){
    Customer customer = customerRepository.findByEmail(ct.
getC_email());
    String customerID = customer.getId();
    System.out.println(ct.getEmail());
    System.out.println(ct.getOrganizationName());
    Entrepreneur entrepreneur = entrepreneurRepository.
findEntrepreneurByEmailAndOrganizationName(ct.getEmail(),ct.
getOrganizationName());
    String entrepreneurID = entrepreneur.getId();
    if (customerID == null ||entrepreneurID == null) {return new
ResponseEntity<Transaction>(HttpStatus.NOT_FOUND);}
    Transaction t = new Transaction(customerID, entrepreneurID, ct.
getMoney(), ct.getDueDate());
    return new ResponseEntity<Transaction>(transactionRepository.
save(t),HttpStatus.OK);
}
```

```
@PostMapping("/update")
public ResponseEntity<Transaction> updateTransaction(@RequestBody
updateRequest req){
    Transaction transaction = transactionRepository.findById(req.
getID()).get();
    if (transaction == null){return new ResponseEntity<Transaction
>(HttpStatus.NOT_FOUND);}
    if (transaction.getUnpaid() - req.getPaid() < 0){return new
ResponseEntity<Transaction>(HttpStatus.CONFLICT);}
    else{
        transaction.setUnpaid(transaction.getUnpaid() - req.getPaid
());
        UpdateLog updateLog = new UpdateLog(transaction.getId(),
req.getPaid());
        transactionRepository.save(transaction);
        updateLogRepository.save(updateLog);
    }
    return new ResponseEntity<Transaction>(transaction,HttpStatus.
OK);
}
```

UML Diagram ส่วนที่ใช้ Design Pattern



3. Adapter (LogRecycleView.kt)

**ปัญหาที่พบ**

- เรามีค่าลิสต์ที่ต้องการแสดงผลมากเกินกว่าจะรันลูปเพื่อสร้าง textView ทีละอัน(กิน memory เยอะเกินไป)

**วิธีการแก้ไขปัญหาด้วย Design Pattern**

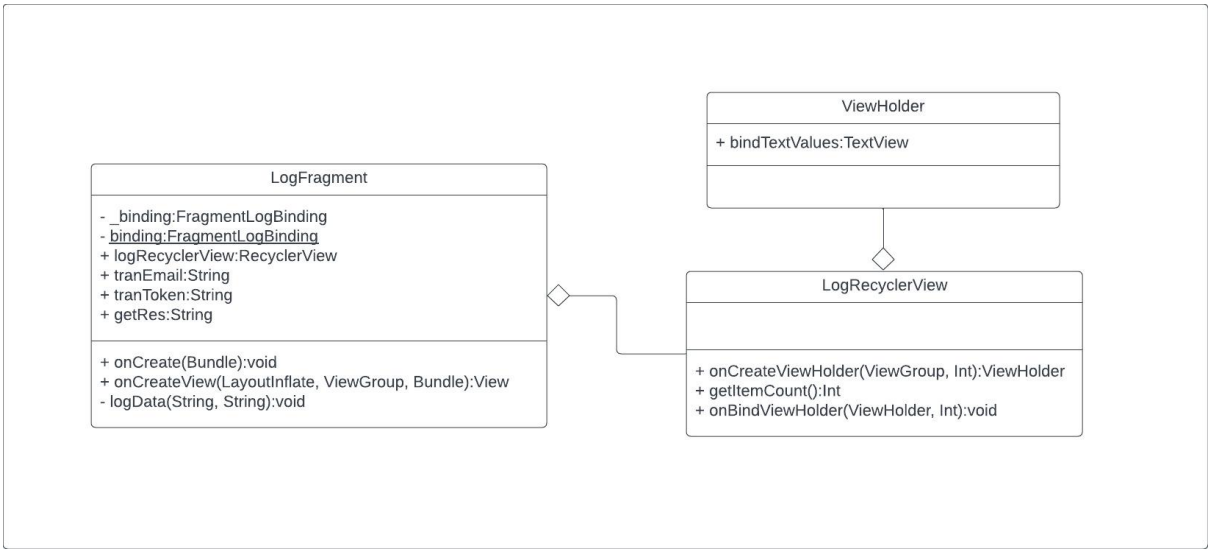
- เขียน Class LogRecycleView เพื่อแปลงค่า arrayList ที่เราต้องการแสดงผลให้กลายเป็น ViewHolder เพื่อให้แสดงผลได้

**Code ส่วนที่ใช้ Design Pattern**

```
class LogRecycleView(private val items: Array<String>, val context:
Context) :
    RecyclerView.Adapter<ViewHolder>() {
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int
): ViewHolder {
        val v = LayoutInflater.from(parent.context).inflate(R
.layout.row,parent,false)
        return ViewHolder(v)
    }

    override fun getItemCount(): Int {
        return items.size
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int
) {
        holder.bindTextValues!!.text = items[position]
    }
}
```

```
val logAdapter = LogRecycleView(newlog.toTypedArray
(),requireContext())
logRecyclerView!!.adapter = logAdapter
```

## UML Diagram ส่วนที่ใช้ Design Pattern



**Quality attribute scenarios**

1. Availability [Exception Handling]

| | |
|---|---|
| Source of stimulus | ลูกค้า Register ด้วยข้อมูลที่มีอยู่แล้ว |
| Stimulus | Fault: Exception |
| Artifacts | กำลังดำเนินการ |
| Environment | ดำเนินการตามปกติ |
| Response | ส่ง Already Register กลับไปหาลูกค้า |
| Response measure | อัตราการทำงานโดยไม่เกิด Fault |

2. Integrability [Adhere to Standards]

| | |
|---|---|
| Source of stimulus | ผู้ประกอบการ |
| Stimulus | ต้องการ Integrate กับระบบ |
| Artifacts | ส่วนการชำระเงินของระบบ |
| Environment | Integration |
| Response | ระบบการชำระเงินมีการทำงานได้สมบูรณ์ |
| Response measure | การเปลี่ยนแปลงโค้ด |

3. Modifiability [Coupling]

| | |
|---|---|
| Source of stimulus | ผู้พัฒนา |
| Stimulus | ต้องการเพิ่มฟังก์ชั่น |
| Artifacts | โค้ด |
| Environment | เวลาในการออกแบบ |
| Response | เกิดการเปลี่ยนแปลงโค้ด |
| Response measure | เวลา ความพยายามในการเพิ่มฟังก์ชั่น |

4. Performance [Increase Resource]

| | |
|---|---|
| Source of stimulus | ลูกค้าต้องการดูเครดิต |
| Stimulus | ช่วงที่แอพพลิเคชั่นรับค่ามาแสดงผล |
| Artifacts | ระบบ |
| Environment | โหมดปกติ |
| Response | แอพพลิเคชั่นแสดงผลเครดิตของลูกค้า |
| Response measure | เวลาที่ใช้ตอบสนอง |

5. Usability [Using System Efficiency]

| | |
|---|---|
| Source of stimulus | ผู้ประกอบการ |
| Stimulus | ต้องการเรียกดูข้อมูลของลูกค้าที่จะทำธุรกรรมด้วย |
| Artifacts | Command-line Interface |
| Environment | เวลาในการทำงาน |
| Response | ให้ข้อมูลของลูกค้าที่ทำธุรกรรมด้วยในลักษณะ json format |
| Response measure | เวลาในการทำงาน |

6. Usability [Using System Efficiency]

| | |
|---|---|
| Source of stimulus | ลูกค้า |
| Stimulus | ต้องการดูคะแนน Credit ของตนเอง |
| Artifacts | ระบบ |
| Environment | เวลาในการทำงาน |
| Response | แสดงผลคะแนน Credit |
| Response measure | ระยะเวลาประมวลผลก่อนลูกค้าได้เห็น Credit |

7. Security [Identity Actors] JWT token Username Password

| | |
|---|---|
| Source of stimulus | ผู้ที่ไม่ใช่ลูกค้าต้องการดูข้อมูลของลูกค้า |
| Stimulus | การล็อคอินเข้าสู่ระบบ |
| Artifacts | ข้อมูลของลูกค้า |
| Environment | ระบบขณะทำงานอยู่ |
| Response | ข้อมูลไม่ถูกดึงไปเพราะล็อคอินไม่สำเร็จ |
| Response measure | จำนวนการโจมตีที่ต่อต้านสำเร็จ |

8. Security [inform actors]

| | |
|---|---|
| Source of stimulus | ผู้ประกอบการที่ต้องการเข้าดูข้อมูล |
| Stimulus | การขอดูเครดิตลูกค้า |
| Artifacts | เครดิตของลูกค้า |
| Environment | ระบบขณะทำงานอยู่ |
| Response | เก็บประวัติข้อมูลของผู้ประกอบการที่ขอข้อมูล และช่วงระยะเวลาการขอดูเครดิต |
| Response measure | ระยะเวลาเวลาที่ลูกค้าสามารถพบประวัติการขอดูเครดิตได้ |

# Reference

1. Source code

   SoftArch by กลุ่มก้อนชายไทย (github.com)

2. UML Diagram (แยกแต่ละโดเมน)

   http://bit.ly/3EUllW8

3. Component Diagram

   http://bit.ly/3EomqnQ