Loan Service

By Crisyanto Parulian

Background

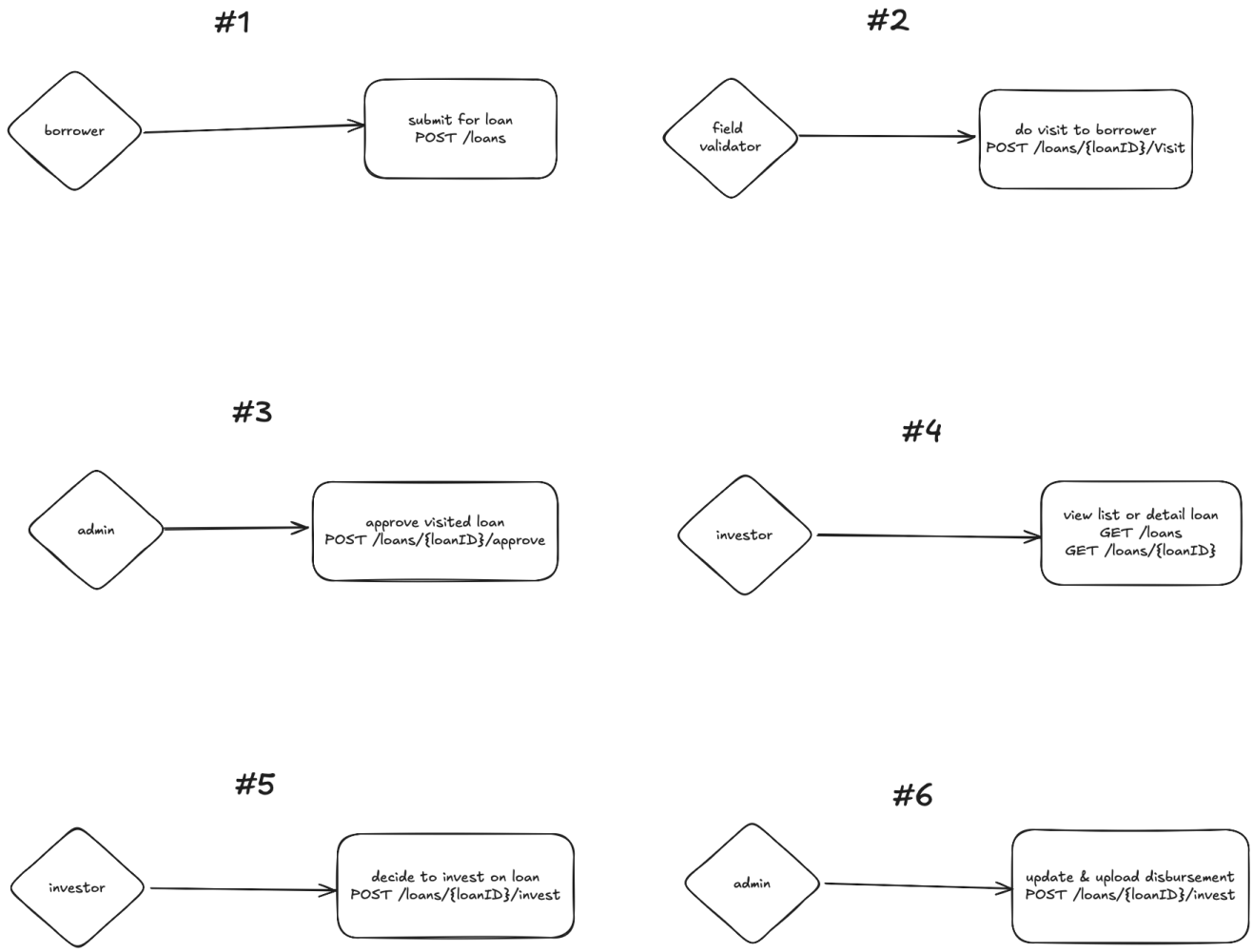
To design a RESTful API for the loan service, we need to create endpoints that allow users to interact with the loan system, manage loan states, and handle transitions between states.

Requirements

A. Rule of state (proposed, approved, invested , disbursed). Movement between state can only move forward.
    a. proposed is the initial state (when loan created it will has proposed state):
    b. approved is once it approved by our staff.
        i. approval must contains several information:
            1. the picture proof of the a field validator has visited the borrower
            2. the employee id of field validator
            3. date of approval
        ii. once approved it can not go back to proposed state
        iii. once approved loan is ready to be offered to investors/lender
    c. invested is once total amount of invested is equal the loan principal
        i. loan can have multiple investors, each with each their own amount
        ii. total of invested amount can not be bigger than the loan principal amount
        iii. once invested all investors will receive an email containing link to agreement letter (pdf)
    d. disbursed is when is loan is given to borrower.
        i. a disbursement must contains several information:
            1. the loan agreement letter signed by borrower (pdf/jpeg)
            2. the employee id of the field officer that hands the money and/or collect the agreement letter
            3. date of disbursement

B. Loan only need following information
    a. borrower id number
    b. principal amount
    c. rate, will define total interest that borrower will pay
    d. ROI return of investment, will define total profit received by investors
    e. Link to the generated agreement letter

System Design



API interactions

Available role in system

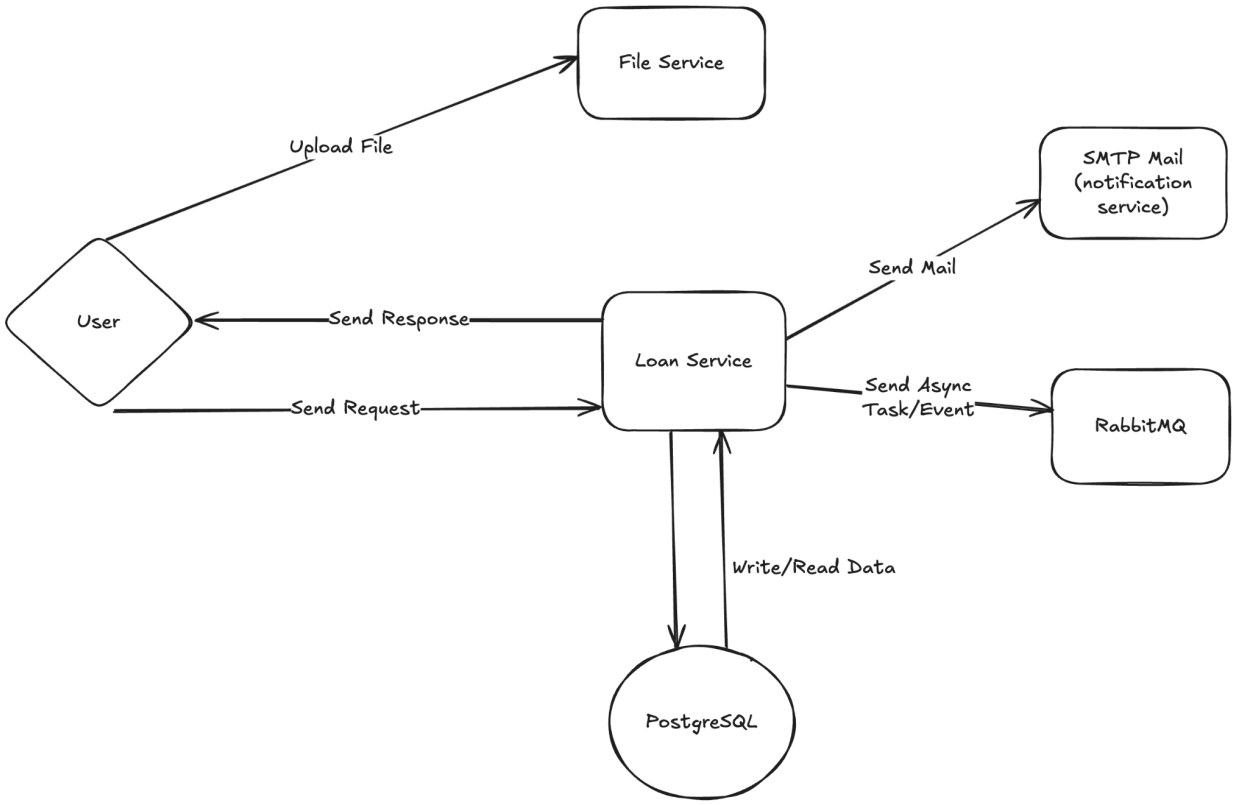
External User

- borrower
- investor

Internal User

- field_validator =>
    - visit the borrower & take photo proof
- field_officer =>
    - doing disbursement in field (*but not access the system*)
- admin =>
    - approve the proposed loan
    - do disbursement in system

A. System Architecture



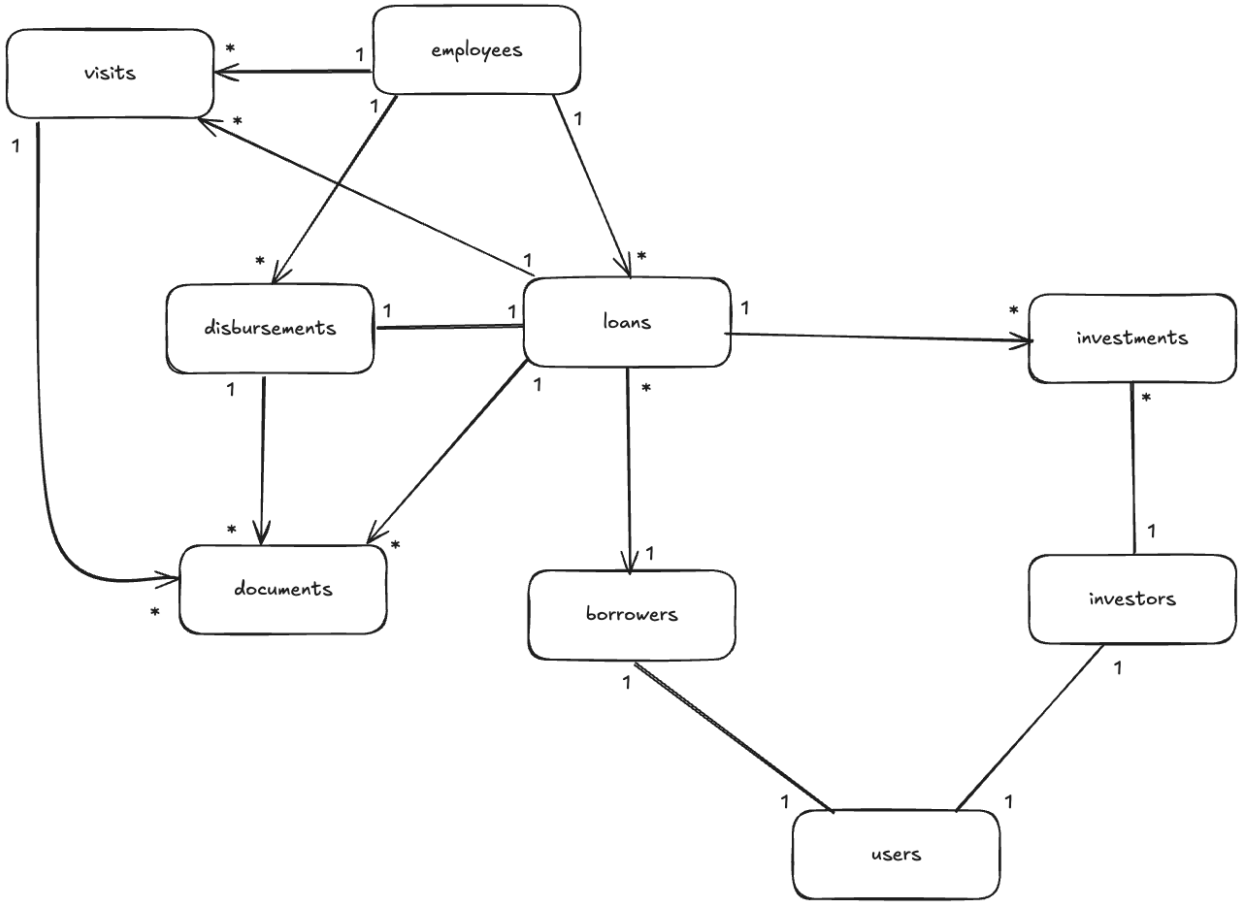
architecture visualization

Note:

- For this task I do not cover part of authentication, so for the information logged in user, I use request header to assume user is already logged in & has valid token jwt that is already decoded. The information about user is followed by this key :
    - login-user-id: {uuid} contains user uuid for login user
    - login-user-role: {enum role} contains role for user

  this request header will required since every resource API on loan service required to authenticated
- User (mobile/webapps) directly upload to file service, so the loan service just get the full url path of file

B. Table Design



relational picture

Table Details:

1. loans

Description : store all incoming proposed loan

Name	Type	Description
id	uuid(pk)	
loan_code	varchar()	generated loan code by system to easier understand/copy by user format: {code}YYMMDD{5 digit random} code by borrower type=> - RGL for regular - PRM for premium - ULT for ultimate

borrower_id	uuid (fk->borrowers->id)	NOT NULL, relation to table borrowers
field_validator_id	uuid(fk->employees->id)	nullable, relation to table employees
principal_amount	decimal(12.2)	NOT NULL | principal_amount > 0
total_investment_amount	decimal(12,2)	NOT NULL | DEFAULT 0 | total_investement_amount > 0
total_investor	int	NOT NULL | DEFAULT 0 | total_investor > 0
rate	decimal(5,2)	rate > >= 0 | interest that borrower will pay
roi	decimal(5,2)	NOT NULL | roi >= 0 , return of investment
status	varchar(255)	ENUM ('proposed', 'approved', 'invested', 'disbursed')
approval_date	timestamp	nullable, once approve it will be filled
created_at	timestamp	default NOW()
updated_at	timestamp	nullable

2. users

Name	Type	Description
id	uuid (pk)	
full_name	text	NOT NULL
email	varchar(255)	NOT NULL | UNIQUE
phone	varchar(20)	NOT NULL | UNIQUE
nik	char(16)	nullable | UNIQUE
user_role	array(string)	NOT NULL | ENUM ('borrower', 'investor', 'admin', ‘field_validator’, ‘field_officer’)

		for current condition we use array, next we can enhance using other tables
created_at	timestamp	default NOW()
updated_at	timestamp	nullable

3. borrowers

Description: table for storing borrowers user

Name	Type	Description
id	uuid (pk)	
user_id	uuid (fk -> users->id)	NOT NULL, relation to table users
type	varchar(255)	NOT NULl | ENUM(‘regular’, ‘premium’, ‘ultimate’)
created_at	timestamp	default NOW
updated_at	timestamp	nullable

4. investors

Description: table for storing data investors/lenders

Name	Type	Description
id	uuid (pk)	
user_id	uuid (fk -> users->id)	NOT NULL, relation to table users
company	varchar(255)	nullable, if investor is company
investment_limit	decimal(12,2)	NOT NULL | investment_limit >= 0
created_at	timestamp	default NOW
updated_at	timestamp	nullable

5. investments

Description: for storing where the user investors invest their money

Name	Type	Description
id	uuid (pk)	

loan_id	uuid (fk -> loan->id)	NOT NULL, relation to table loan
investor_id	uuid (fk ->investors -> id)	NOT NULL, relation to table investors
amount	decimal(12,2)	NOT NULL | amount >= 0
created_at	timestamp	default NOW
updated_at	timestamp	nullable

6. employees

Description: for storing internal user, e.g: admin, field_validator

Name	Type	Description
id	uuid (pk)	
user_id	uuid (fk ->users->id)	nullable, because not all employee can access the system
nik	varchar(255)	NOT NULL, unique number for employee
role	varchar(50)	NOT NULL | ENUM('admin', 'field_officer')
created_at	timestamp	default NOW
updated_at	timestamp	nullable

7. visits

Description: for storing data visit by field validator

Name	Type	Description
id	uuid (pk)	
loan_id	uuid (fk ->loans->id)	NOT NULL, relation to loans
employee_id	uuid (fk ->employees->id)	NOT NULL, relation to employees
notes	text	NOT NULL
created_at	timestamp	default NOW()

updated_at	timestamp	nullable

8. disbursements

Description: store disbursement by admin

Name	Type	Description
id	uuid (pk)	
loan_id	uuid (fk ->loans->id)	NOT NULL , references to table loans
field_id_officer	uuid (fk ->employees->id)	NOT NULL, references to employees
date_of_disbursement	timestamp	NOT NULL
created_at	timestamp	default NOW()

9. documents

Description: store all documents (loan agreements, approval proofs, disbursement receipt)

Name	Type	Description
id	uuid (pk)	
reference_id	uuid	NOT NULL , can references to table users, loans or other entity
reference_type	varchar(255)	NOT NULL, dynamic type base on reference | ENUM: visits
document_type	varchar(255)	NOT NULL, we can explain what the document type is | ENUM: proof_visit
file_url	TEXT	path to stored document
created_at	timestamp	default NOW()

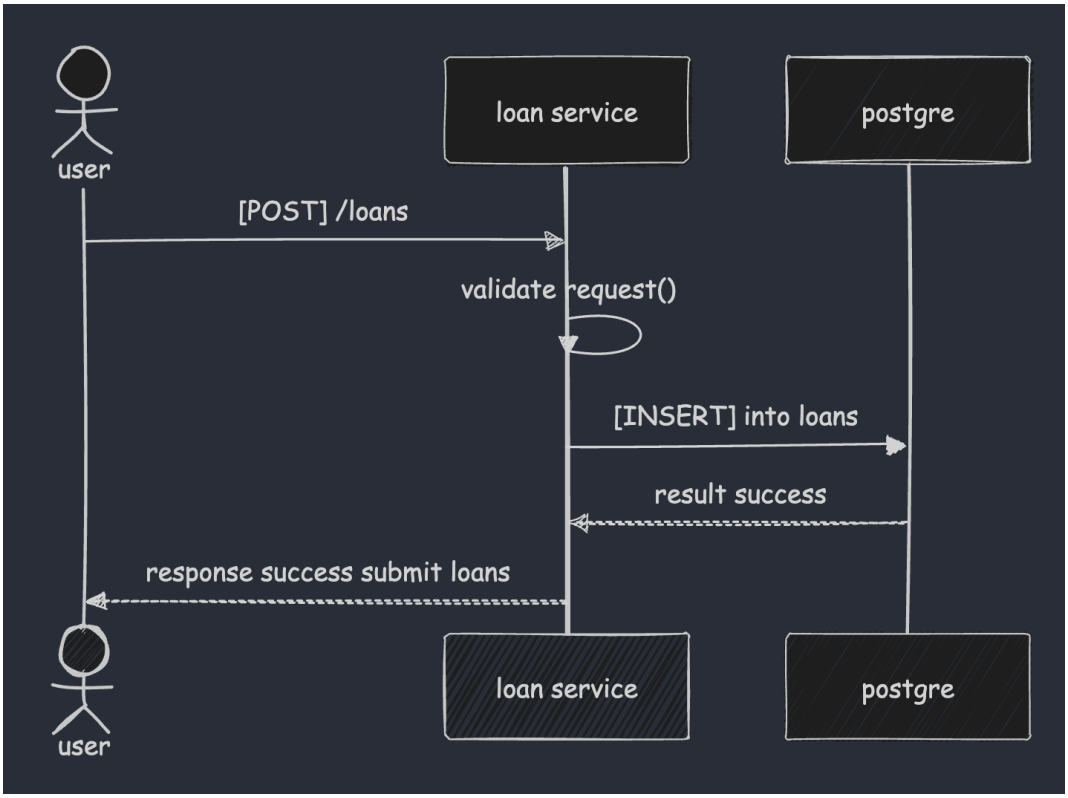
C. List Endpoint

a. POST /loans

Description: API for submit new loan proposal, need to
Headers:

- login-user-id: {uuid}
- login-user-role: {enum role}

Flow:



sequence diagram create loan

```
sequenceDiagram
    actor u as user
    participant ls as loan service
    participant pq as postgre

    u->>ls: [POST] /loans
    ls->>ls: validate request()
    ls->>pq: [INSERT] into loans
    pq-->>ls: result success
    ls-->>u: response success submit loans
```

```
Example Request body:
{
“principal_amount”: 1000000.0,
 “rate”: 10.00,
“roi”: 8.00
}

Example Response body:
{
“success”: true,
“message”: “success submit loan”,
“data”: {
      “loan_id”: "550e8400-e29b-41d4-a716-446655440000"
 }
}
```

b. GET / loans

Description: API for get list loans

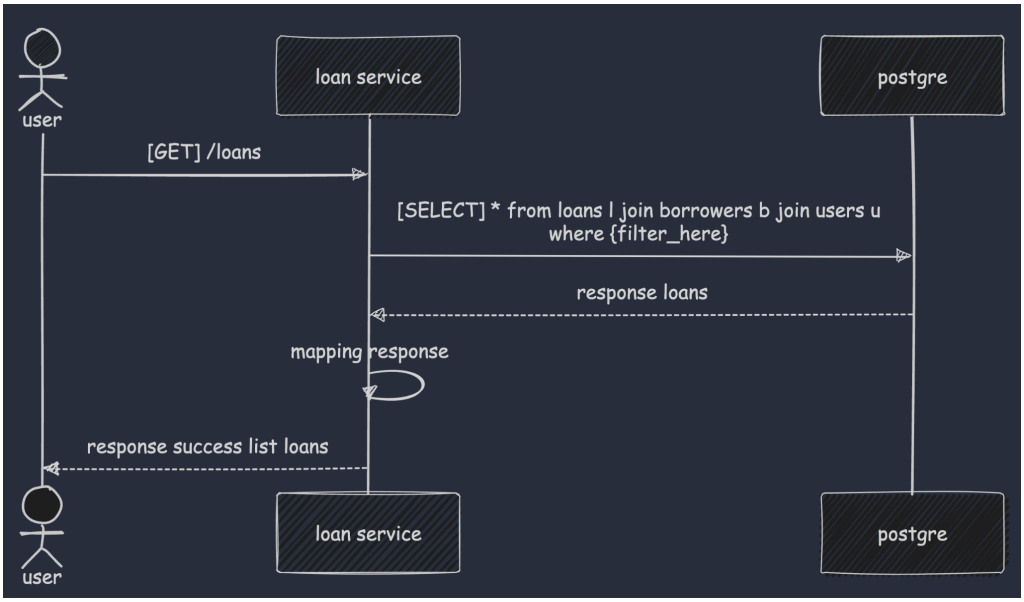
Headers:

- login-user-id: {uuid}
- login-user-role: {enum role}

QueryParams:

- _s: enum: asc | desc (sort)
- _s_by: {column name}, enum: principal_amount | total_investment_amount | rate | roi (sort by)
- _status: enum status

Flow:



sequence get loans

```
sequenceDiagram
    actor u as user
    participant ls as loan service
    participant pq as postgre

    u->>ls: [GET] /loans
    ls->>pq: [SELECT] * from loans l join borrowers b join users u <br>
where {filter_here}
    pq-->>ls: response loans
    ls->>ls: mapping response
    ls-->>u: response success list loans
```

Example Response Body:


```
{
    "success": true,
    "message": "success",
    "data": {
        "total": 1,
        "offset": 0,
        "limit": 10,
        "items": [
            {
                "id": "8703c0a6-ff6f-4af2-af51-148d7bb317a7",
                "borrower_id": "7893c0a6-ff6f-4af2-af51-148d7bb317a9",
                "field_validator_id":
"7893c0a6-ff6f-4af2-af51-148d7bb317a9",
                "principal_amount": 10000000.00,
                "total_investment_amount": 3000000.00,
                "total_investors": 2,
                "rate": 10.00,
                "roi": 10.00,
                "status": "proposed",
                "approval_date": null,
                "created_at": "2019-02-14T17:49:20.987"
            }
        ]
    }
}
```

c. GET /loans/{loanID}

Description: API for get detail loans

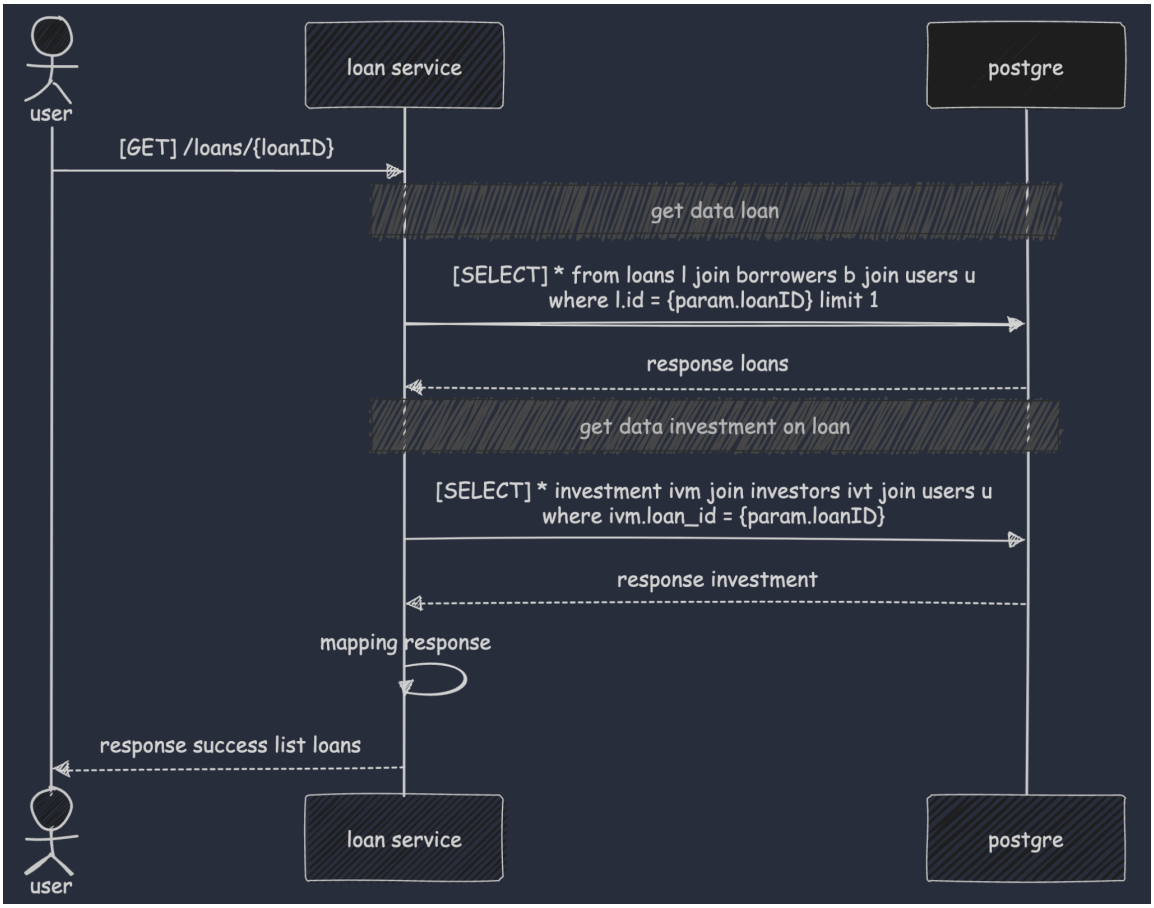
Headers:

- login-user-id: {uuid}
- login-user-role: {enum role}

Path Params:

- loanID string

Flow:



sequence get loan by id

```
sequenceDiagram
    actor u as user
    participant ls as loan service
    participant pq as postgre

    u->>ls: [GET] /loans/{loanID}

    note over ls,pq: get data loan
    ls->>pq: [SELECT] * from loans l join borrowers b join users u <br>
where l.id = {param.loanID} limit 1
    pq-->>ls: response loans
    note over ls,pq: get data investment on loan
    ls->>pq: [SELECT] * investment ivm join investors ivt join users u
<br> where ivm.loan_id = {param.loanID}
    pq-->>ls: response investment
    ls->>ls: mapping response
    ls-->>u: response success list loans
```

Example Response Body:

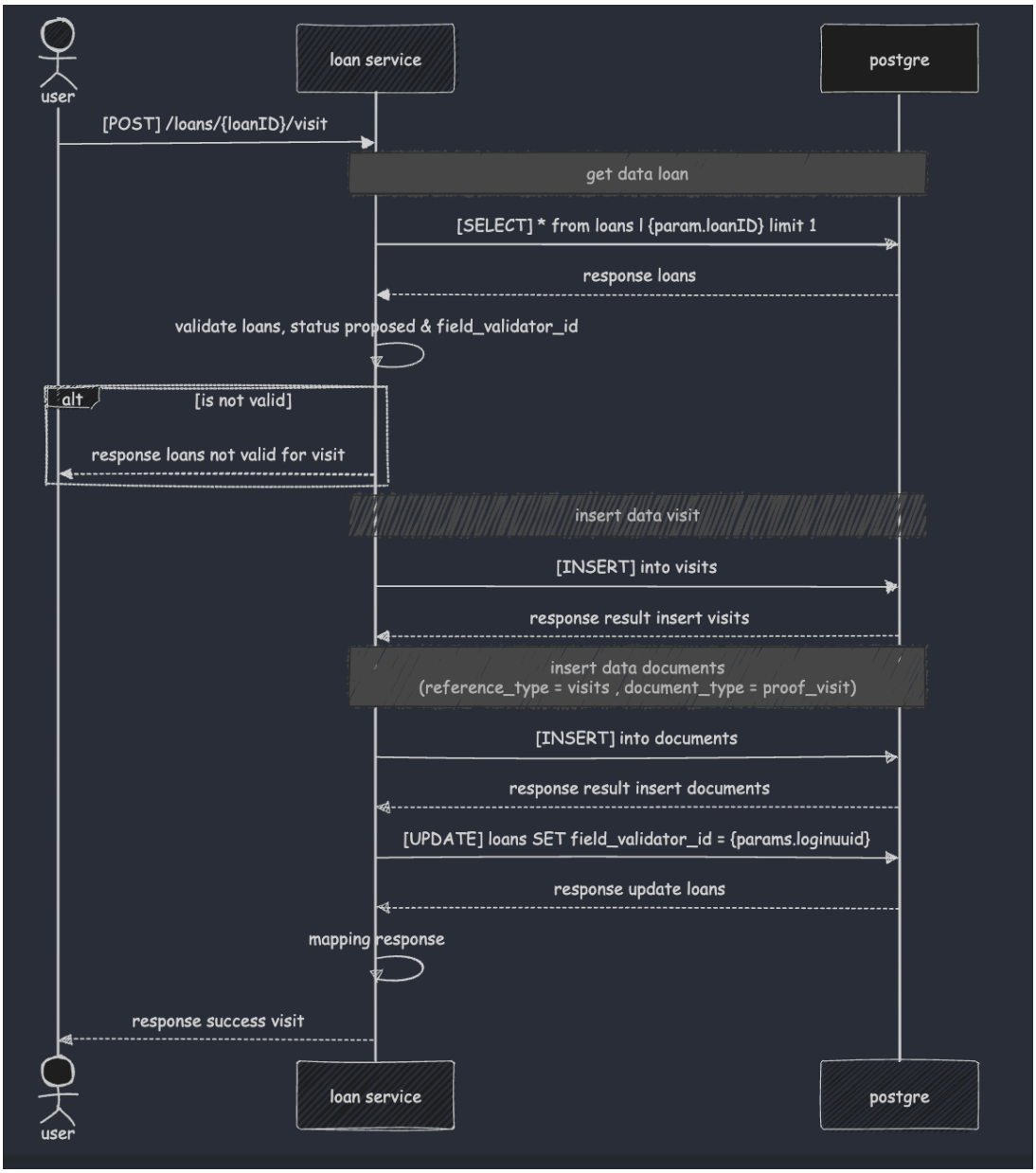

```
{
    "success": true,
    "message": "success",
    "data": {
        "id": "8703c0a6-ff6f-4af2-af51-148d7bb317a7",
        "field_validator_id": "7893c0a6-ff6f-4af2-af51-148d7bb317a9",
        "principal_amount": 10000000.00,
        "total_investment_amount": 3000000.00,
        "total_investors": 1,
        "rate": 10.00,
        "roi": 10.00,
        "status": "proposed",
        "approval_date": null,
         "borrower": {
            "id": "7893c0a6-ff6f-4af2-af51-148d7bb317a9",
            "name": "jhon doe",
            "email": "jhon@gmail.com",
            "phone": "628524288849",
            "nik": "3277289999238844",
            "joined_at": "2019-02-14T17:49:20.987",
         },
        "investments": [
            {
                "investment_id": "999c0a6-ff6f-4af2-af51-148d7bb3178k",
                "investor_id": "8993c0a6-ff6f-4af2-af51-148d7bb31l0l",
                "investor_name": "conello",
                "investor_email": "conello@gmail.com",
                "amount": 3000000.00,
```


d. POST /loans/loanID/visit

Description: create visit to check the validity of borrower (field validator)
Headers:

- login-user-id: {uuid}
- login-user-role: {enum role}

Flow:



sequence visit borrower

```
sequenceDiagram
    actor u as user
    participant ls as loan service
    participant pq as postgre

    u->>ls: [POST] /loans/{loanID}/visit

    note over ls,pq: get data loan
    ls->>pq: [SELECT] * from loans l {param.loanID} limit 1;
    pq-->>ls: response loans
    ls->>ls: validate loans, status proposed & field_validator_id

    alt is not valid
    ls-->>u: response loans not valid for visit
    end

    note over ls,pq: insert data visit
    ls->>pq: [INSERT] into visits
    pq-->>ls: response result insert visits
    note over ls,pq: insert data documents <br>(reference_type = visits
, document_type = proof_visit)
    ls->>pq: [INSERT] into documents
    pq-->>ls: response result insert documents
    ls->>pq: [UPDATE] loans SET field_validator_id = {params.loginuuid}
    pq-->>ls: response update loans

    ls->>ls: mapping response
    ls-->>u: response success visit
```

```
Example Request Body:
Body :
{
“proof”: [“https://example-image.com/proof1.jpg”],
“notes”: “customer is human”
}

Example Response Body:
{
“success”: true,
“message”: “success visit”
}
```

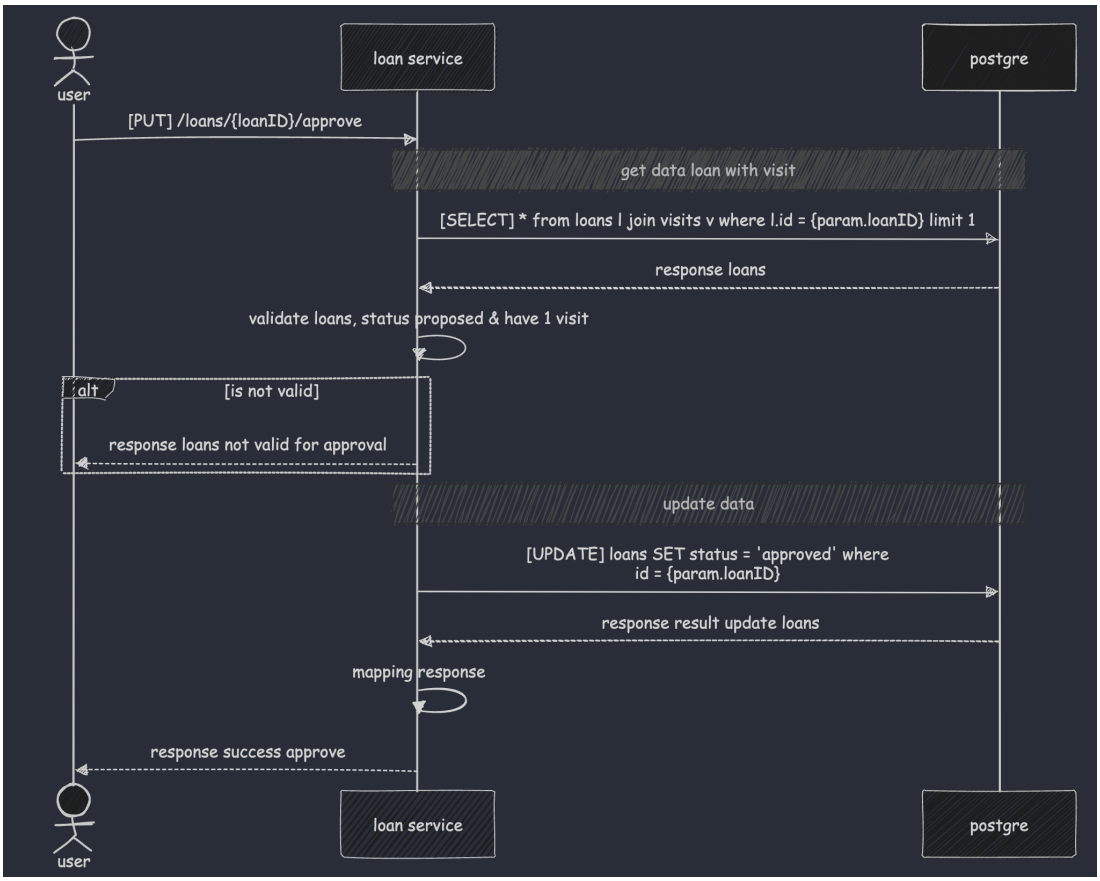
e. PUT /loans/{loanID}/approve

Description: approve loan proposal by admin

Headers:

- login-user-id: {uuid}
- login-user-role: {enum role}

Flow:



sequence approve loan

```
sequenceDiagram
    actor u as user
    participant ls as loan service
    participant pq as postgre

    u->>ls: [PUT] /loans/{loanID}/approve

    note over ls,pq: get data loan with visit
    ls->>pq: [SELECT] * from loans l join visits v where l.id =
{param.loanID} limit 1;
    pq-->>ls: response loans
    ls->>ls: validate loans, status proposed & have 1 visit
    alt is not valid
    ls-->>u: response loans not valid for approval
    end
    note over ls,pq: update data
    ls->>pq: [UPDATE] loans SET status = 'approved', approved_date =
NOW() where <br> id = {param.loanID}
    pq-->>ls: response result update loans
    ls->>ls: mapping response
    ls-->>u: response success approve
```

```
Example Response Body:
{
“success”: true,
“message”: “success approve loan”
}
```

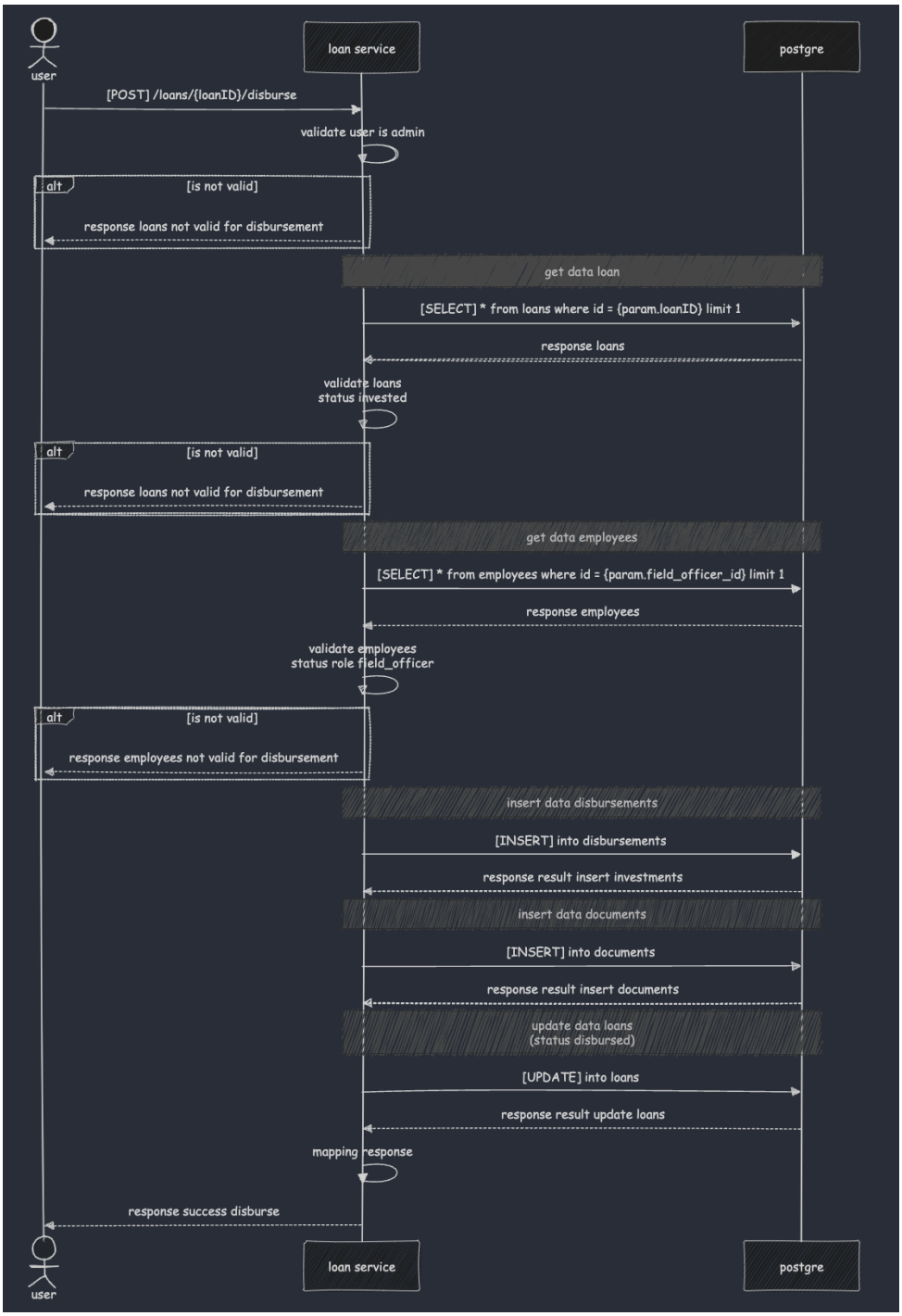
f. POST /loans/{loanID}/disburse

Description: create disbursement on loan

Headers:

- login-user-id: {uuid}
- login-user-role: {enum role}

Flow:



sequence disbursement

```
sequenceDiagram
    actor u as user
    participant ls as loan service
    participant pq as postgre

    u->>ls: [POST] /loans/{loanID}/disburse

    ls->>ls: validate user is admin
    alt is not valid
    ls-->>u: response loans not valid for disbursement
    end
    note over ls,pq: get data loan
    ls->>pq: [SELECT] * from loans where id = {paran.loanID} limit 1;
    pq-->>ls: response loans;
    ls->>ls: validate loans<br> status invested
    alt is not valid
    ls-->>u: response loans not valid for disbursement
    end
    note over ls,pq: get data employees
    ls->>pq: [SELECT] * from employees where id =
{paran.field_officer_id} limit 1;
    pq-->>ls: response employees;
    ls->>ls: validate employees<br> status role field_officer
    alt is not valid
    ls-->>u: response employees not valid for disbursement
    end

    note over ls,pq: insert data disbursements
    ls->>pq: [INSERT] into disbursements
    pq-->>ls: response result insert investments
    note over ls,pq: insert data documents
    ls->>pq: [INSERT] into documents
    pq-->>ls: response result insert documents
    note over ls,pq: update data loans <br>(status disbursed)
    ls->>pq: [UPDATE] into loans
    pq-->>ls: response result update loans
    ls->>ls: mapping response
```

```
Example Request Body:
{
"loan_aggrement_url": "https://test.com/loan-aggrement.pdf",
"field_officer_id": "37a8b511-3561-4be7-9049-aa0cbcd837c9",
"date_of_disbursement": "2025-02-27T08:38:19.8484332"
}
```

```
Example Response Body:
{
"success": true,
"message": "success disburse loan"
}
```

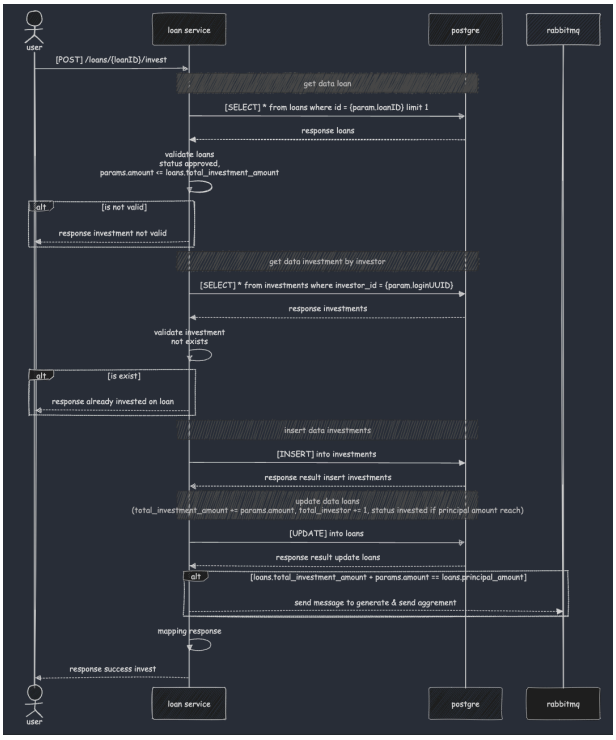
g. POST /loans/{loanID}/invest

Description: Create investment on loan

- 1 investor can only invest once on a loan
- currently we skip the payment validation

Headers:

- login-user-id: {uuid}
- login-user-role: {enum role}



sequence invest

```
sequenceDiagram
    actor u as user
    participant ls as loan service
    participant pq as postgre
    participant rmq as rabbitmq

    u->>ls: [POST] /loans/{loanID}/invest

    note over ls,pq: get data loan
    ls->>pq: [SELECT] * from loans where id = {param.loanID} limit 1;
    pq-->>ls: response loans;
    ls->>ls: validate loans<br> status approved, <br>params.amount <=
loans.total_investment_amount

    alt is not valid
    ls-->>u: response investment not valid
    end
    note over ls,pq: get data investment by investor
    ls->>pq: [SELECT] * from investments where investor_id =
{param.loginUUID};
    pq-->>ls: response investments;
    ls->>ls: validate investment<br> not exists

    alt is exist
    ls-->>u: response already invested on loan
    end
    note over ls,pq: insert data investments
    ls->>pq: [INSERT] into investments
    pq-->>ls: response result insert investments

    note over ls,pq: update data loans <br>(total_investment_amount +=
```

Example Request Body:

```
{
"amount": 1000000.00
}
```

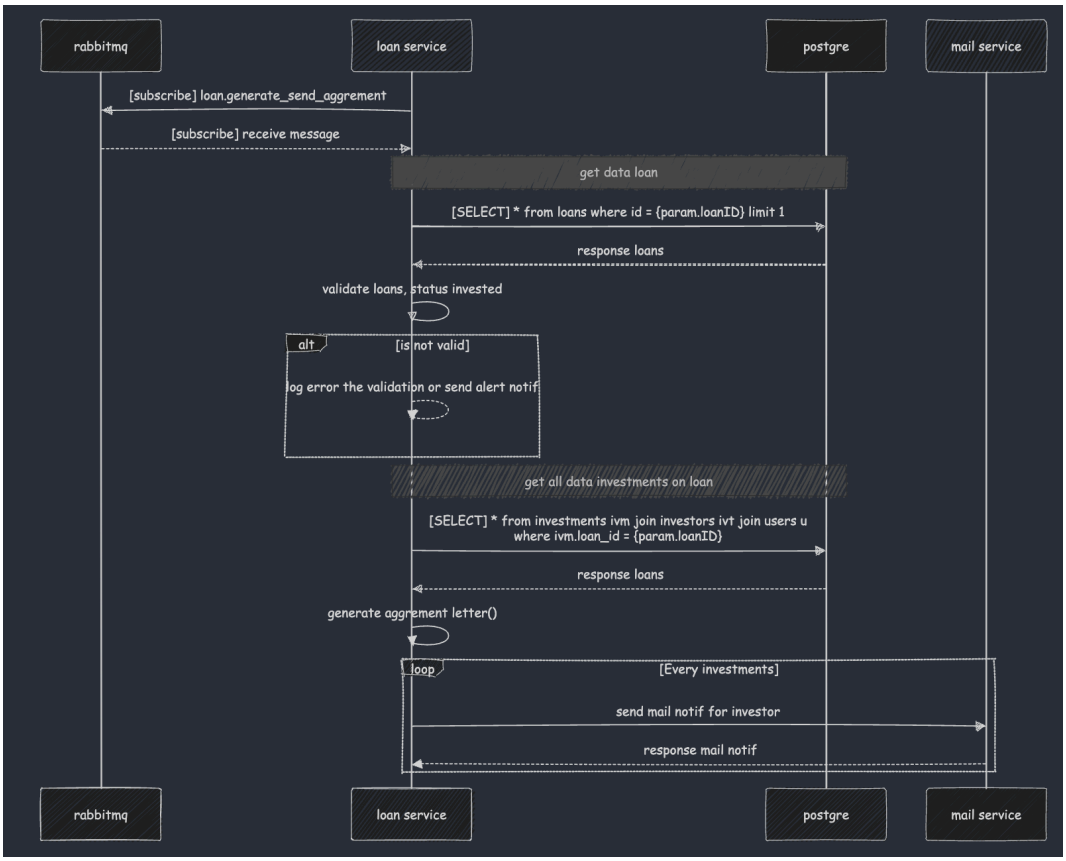
Example Response Body:

```
{
"success": true,
"message": "success invest on loan"
}
```

h. [Subscribe] Event/Queue loan.generate_send_aggrement

Description: subscribe event & proceed to generating pdf agreement then send the agreement to each investor

Flow:



sequence subscribe loan.generate_send_aggrement

```
sequenceDiagram
    participant rmq as rabbitmq
    participant ls as loan service
    participant pg as postgre
    participant mail as mail service

    ls->>rmq: [subscribe] loan.generate_send_aggrement
    rmq-->>ls: [subscribe] receive message

    note over ls,pg: get data loan
    ls->>pg: [SELECT] * from loans where id = {param.loanID} limit 1;
    pg-->>ls: response loans

    ls->>ls: validate loans, status invested

    alt is not valid
    ls-->>ls: log error the validation or send alert notif
    end
    note over ls,pg: get all data investments on loan
    ls->>pg: [SELECT] * from investments ivm join investors ivt join
users u <br> where ivm.loan_id = {param.loanID};
    pg-->>ls: response loans
    ls->>ls: generate aggrement letter()
    loop Every investments
        ls->>mail: send mail notif for investor
        mail-->>ls: response mail notif
    end
```

Example data json:

```
{
  "loan_id":"153ff8b5-3b7c-4fe2-a4c7-eca858d66730"
}
```

i.

j. t

D. Test

Pencapaian

1. ## Masukkan teks Anda di sini

   Masukkan teks Anda di sini Masukkan teks Anda di sini Masukkan teks Anda di sini Masukkan teks Anda di sini Masukkan teks Anda di sini.

2. ## Masukkan teks Anda di sini

   Masukkan teks Anda di sini Masukkan teks Anda di sini Masukkan teks Anda di sini Masukkan teks Anda di sini.

3. Masukkan teks Anda di sini

Masukkan teks Anda di sini Masukkan teks Anda di sini Masukkan teks Anda di sini Masukkan teks Anda di sini Masukkan teks Anda di sini Masukkan teks Anda di sini.