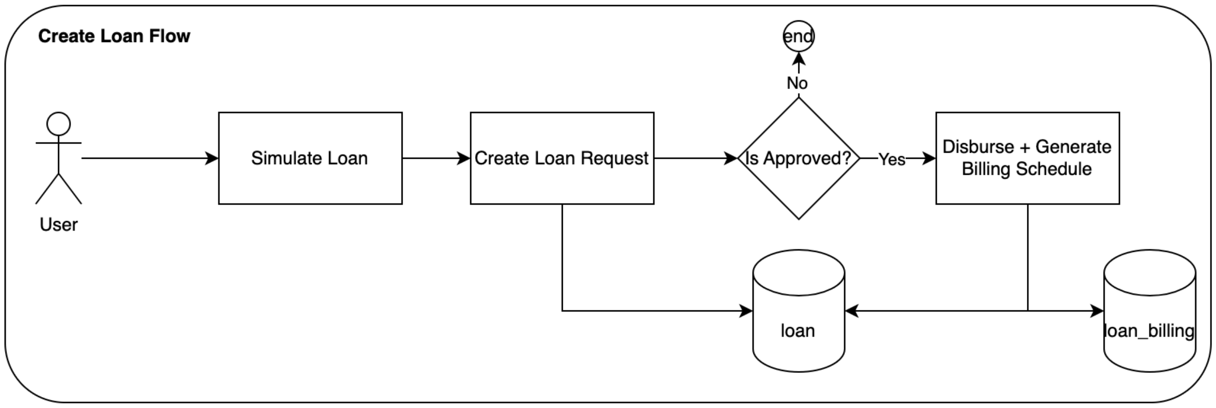
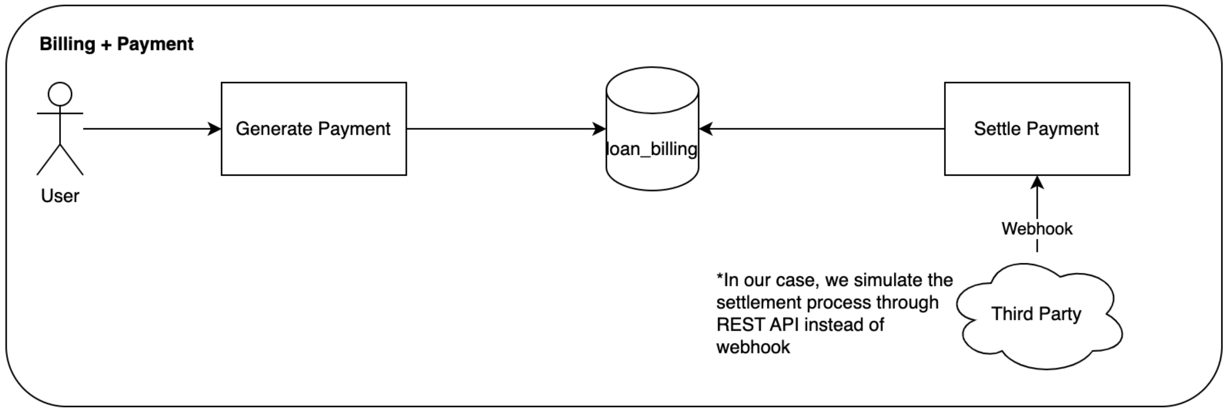
# High Level Design

## Create Loan Flow



- **Simulate Loan**: This function simulates a loan, including the billing schedule. It uses an annual flat rate with a weekly cycle (1 year = 50 weeks). The flat rate value can be configured in the backend server through an environment variable.
  **Formula**:
  - Total Interest = Principal * Interest(%) * (InstallmentLength/50)
  - Principal per Month = Principal / InstallmentLength
  - Interest per Month = Total Interest / InstallmentLength
- **Create Loan Request**: Inserts loan data into the loan database table. At this stage, the status is initially set to PENDING. However, since this case assumes auto-approval, the loan is directly inserted with the status APPROVED.
- **Disburse + Generate**: Waits for the loan request to be approved. Once approved, the service disburses the loan, generates the billing schedule (similar to the loan simulation but with an actual due date), and updates the loan status to ACTIVE.

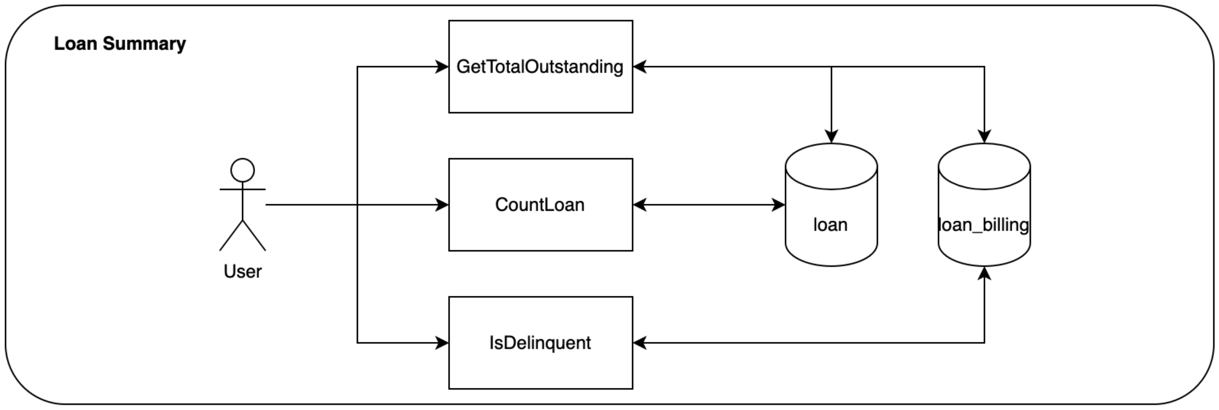
## Billing & Payment Flow


Billing + Payment
User
Generate Payment
loan_billing
Settle Payment
Webhook
Third Party
*In our case, we simulate the settlement process through REST API instead of webhook

There are three billing states: **UNPAID, WAIT_FOR_PAYMENT, PAID**

- **Generate Payment**: This function generates a payment link, including the virtual account (VA) and expiry date. If the expiry date hasn't passed, it retrieves the latest generated payment link. If expired, it regenerates a new one. After this function is triggered, the status changes to WAIT_FOR_PAYMENT. If the payment link expires, the status reverts to UNPAID. In a real scenario, it would be beneficial to use a dedicated table to maintain the history, but here, it's stored in the `loan_billing` table for simplicity.
- **Settle Payment**: In a real-world scenario, a webhook function is typically created and triggered by a third party (e.g., payment processor/gateway service). In this case, we use a REST API to simulate that process. Once this function is triggered, the status changes from WAIT_FOR_PAYMENT to PAID.

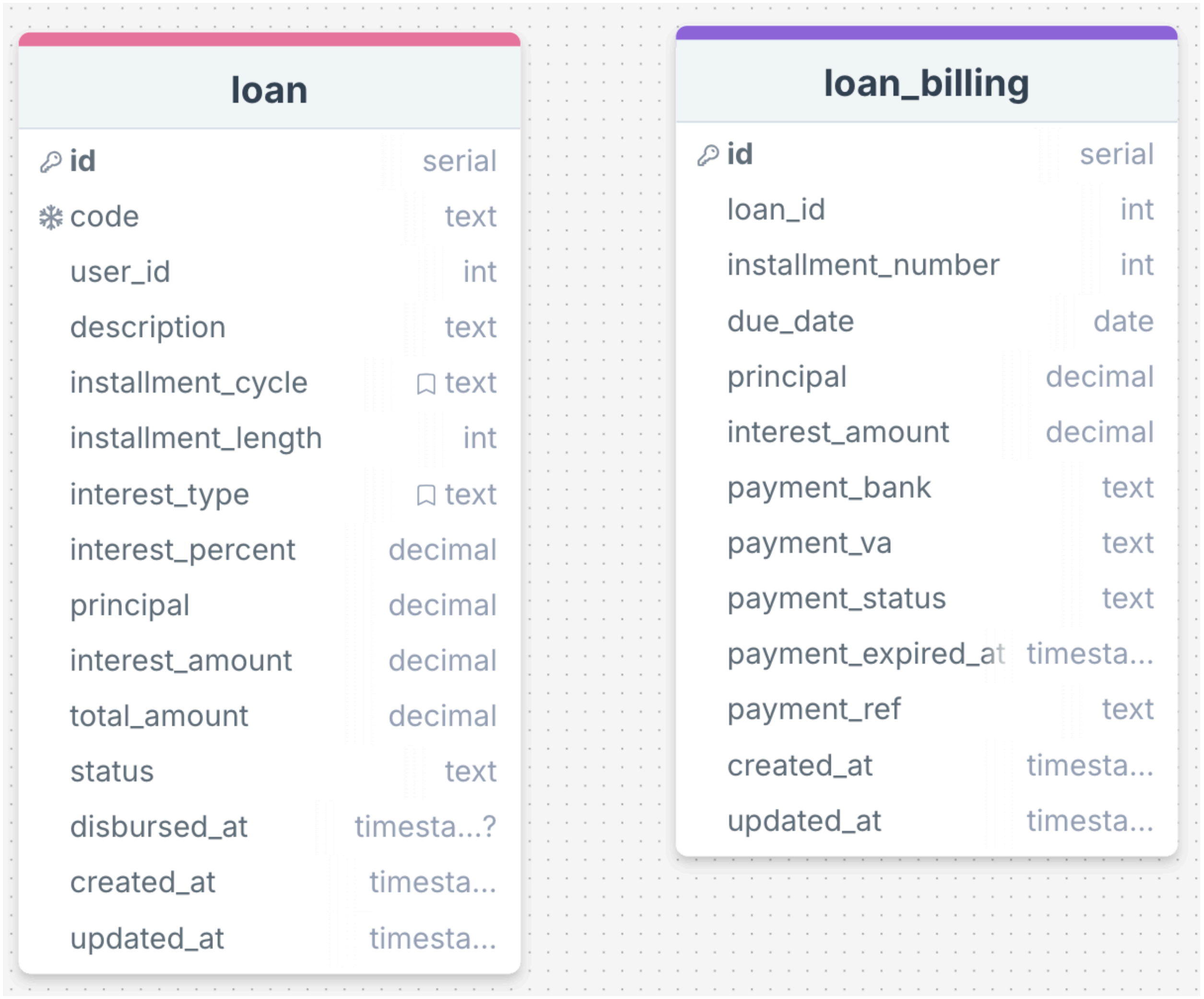
## Get Loan Summary



This flow is typically used on the main dashboard (summary page).

- **GetTotalOutstanding**: Queries data from the `loan` and `loan_billing` tables, filtering for billing statuses that are not yet PAID (statuses UNPAID and WAIT_FOR_PAYMENT).
- **CountLoan**: Queries data from the `loan` table, filtering for loans with an ACTIVE status.
- **IsDelinquent**: Queries the `loan_billing` table, filtering for data past the due date and with a status that is not yet PAID.

## Database Schema



| loan | |
|---|---|
| **id** | serial |
| code | text |
| user_id | int |
| description | text |
| installment_cycle | text |
| installment_length | int |
| interest_type | text |
| interest_percent | decimal |
| principal | decimal |
| interest_amount | decimal |
| total_amount | decimal |
| status | text |
| disbursed_at | timesta...? |
| created_at | timesta... |
| updated_at | timesta... |

| loan_billing | |
|---|---|
| **id** | serial |
| loan_id | int |
| installment_number | int |
| due_date | date |
| principal | decimal |
| interest_amount | decimal |
| payment_bank | text |
| payment_va | text |
| payment_status | text |
| payment_expired_at | timesta... |
| payment_ref | text |
| created_at | timesta... |
| updated_at | timesta... |

For more detail, follow this: https://github.com/herryg91/billing/tree/main/rest-api/migrations