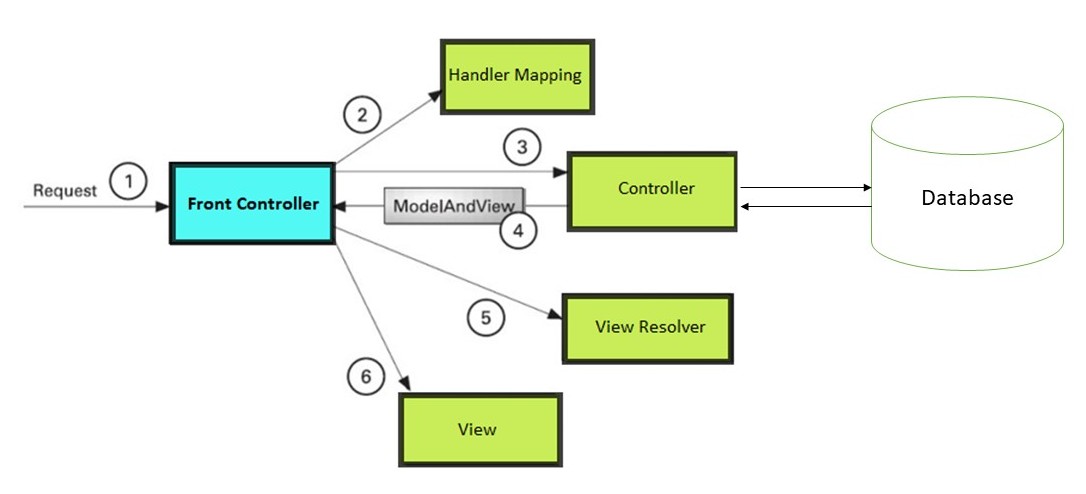
Bank Application

A: Project Scope

The project is an online banking application for customers of a bank to open an account, view the account information and execute basic banking operations.

|  |  |  |
| --- | --- | --- |
| Team Name: **Blitzkrieg** | | |
| **PSID** | **Name** | **Email ID** |
| 45066058 | Akash Nainani | akash.shailesh.nainani@hsbc.co.in |
| 45063942 | Haindavi Bondugula | bondugula.haindavi@hsbc.co.in |
| 45067129 | Musthyala Srikar | srikar.mustyala@hsbc.co.in |
| 45066078 | Nagendra Singh | nagendra.singh@hsbc.co.in |
| 45066116 | Sarah Naik | sarah.mansoor.naik@hsbc.co.in |
| 45066188 | Saurabh Kumar | kumar.saurabh@hsbc.co.in |
| 45064721 | Shwetha S | swetha3.s@hsbc.co.in |
| 45063641 | Sunanda Saha | sunanda.saha@hsbc.co.in |
| 45066145 | Swarajdeep Saha | swarajdeep.saha@hsbc.co.in |
| 45063211 | Syed Safdar Ali | safdar.safdar.ali@hsbc.co.in |

B: MVC Design Pattern - Public Interface



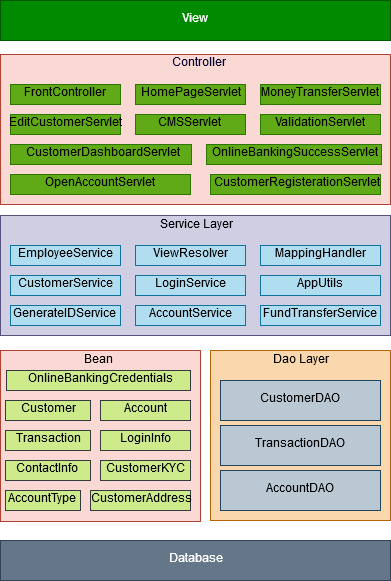
* As displayed in the figure, all the incoming request is intercepted by the the front controller.
* The front controller does handler mapping and forwards the request to the controller.
* The controller returns the object of ModelAndView.
* The DispatcherServlet checks the entry of view resolver in the XML file and invokes the specified view component.

Benefits of the MVC Design Pattern

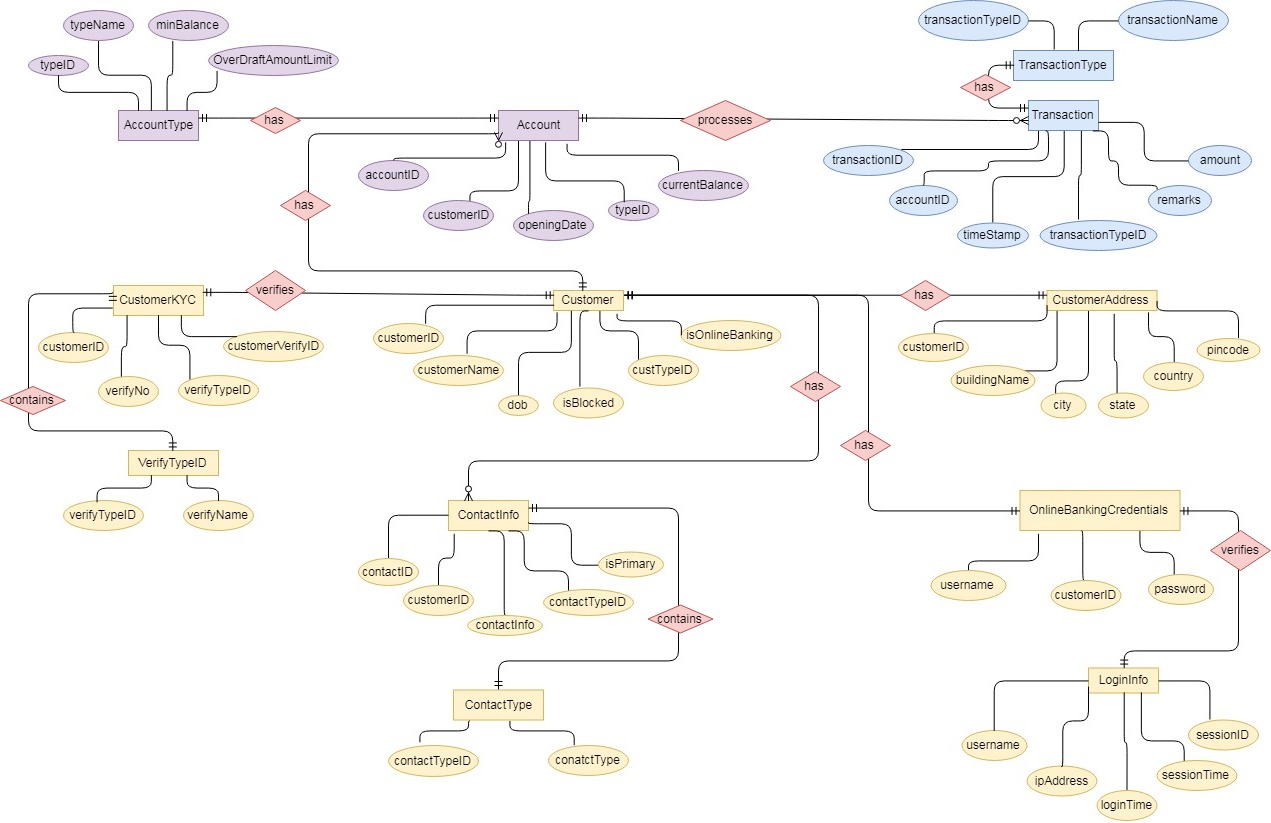
The application of the model-view-controller division has several benefits:

* We can distribute development efforts, so that implementation changes in one part of the web application do not require changes to another. The developers responsible for writing the business logic can work independently of the developers responsible for the flow of control, and web-page designers can work independently of the developers.
* We can more easily migrate legacy programs, because the view is separated from the model and the control and can be tailored to platform and user category.
* The MVC design has an organizational structure that better supports scalability (building bigger applications) and ease of modification and maintenance (due to the cleaner separation of tasks).

C: Architecture Diagram

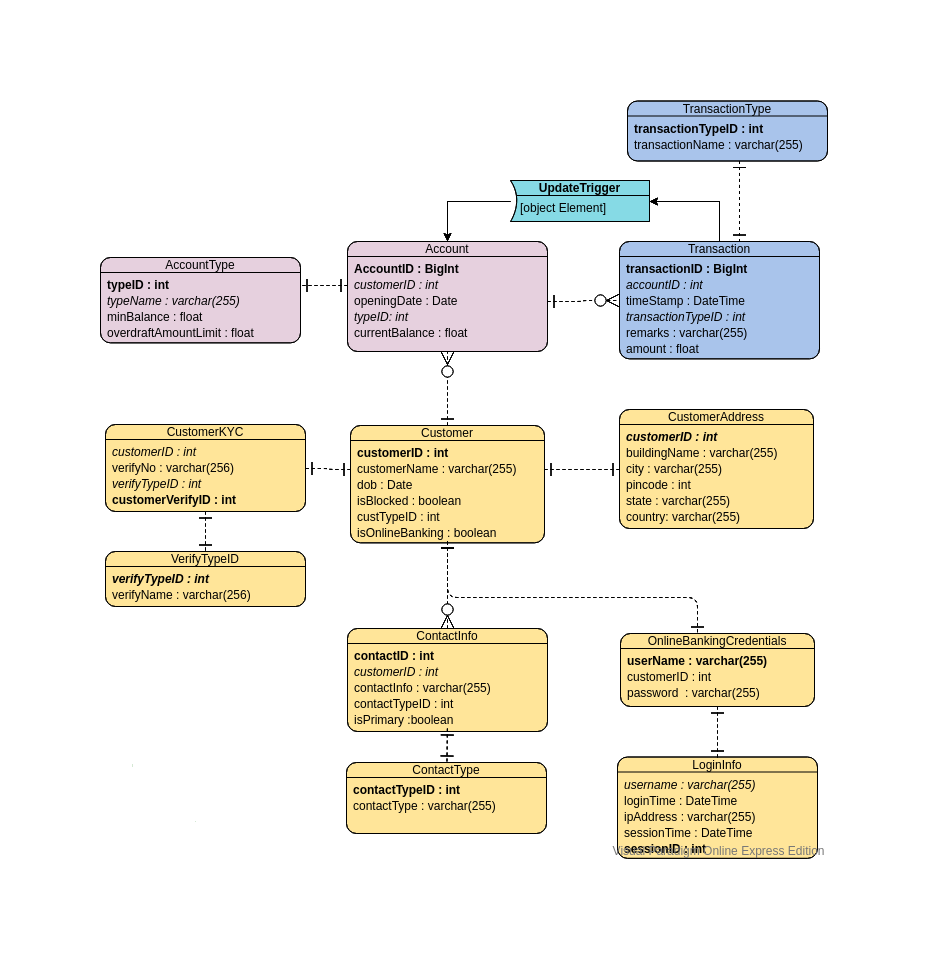


D: ER Diagram



The project contains 13 tables storing the data in a highly normalised manner in the Derby database. The tables contain information about the customers, their accounts with the bank, the transactions that are performed and the logging information. The specification has been normalized up to 3NF to eliminate all the anomalies that may arise due to the database transaction that are executed by the customers and the bank employees. The table design is being done to provide higher consistency and reliability for the data storage. At all proper levels care is taken to check that the system manages the data consistency with proper business rules and authentication.

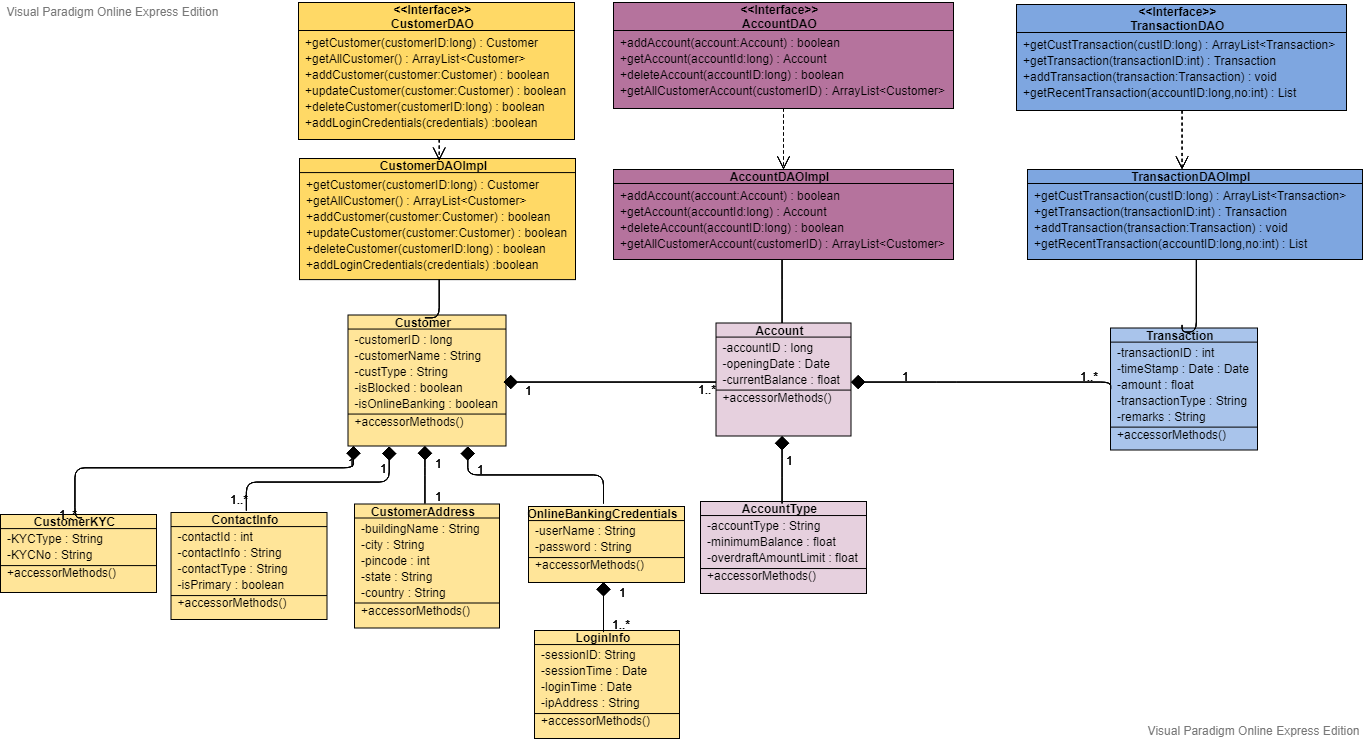
E: Table Schema Diagram



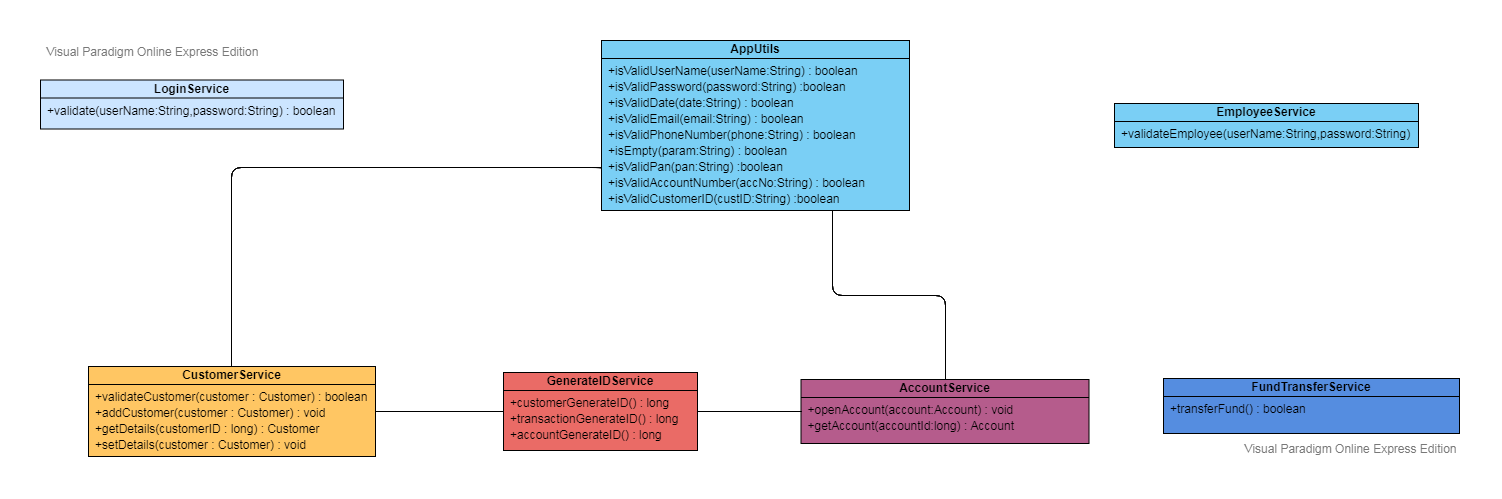
F: Class Diagram

In our design we have divided the whole application into three layers namely the data access layer, service layer and controller layer. It makes it easier to maintain, enhance, and debug by centralizing core application functions into their own chunks of code. The data access layer provides a way to design an application with a clean separation of code into their functional areas within an application. It is being used for the CRUD (Create, Read, Update, and Delete) operations performed on the Derby Database. We have created interfaces for CustomerDAO, AccountDAO and TransactionDAO and their implementations in CustomerDAOImpl, AccountDAOImpl and TransactionDAOImpl respectively. The Controller layer is being used to translate incoming requests into outgoing responses after processing. In order to do this, the controller take request data and pass it into the Service layer. Service layer is a design pattern, applied within the service-orientation design paradigm, which is being used to organize the services, within a service inventory, into a set of logical layers. We have implemented Login Service, Customer Service, Employee Service, FundTransfer Service, Account Service and App Utils for various housekeeping tasks and Exception Handling. The different classes work together as well as independently to implement the business logic.

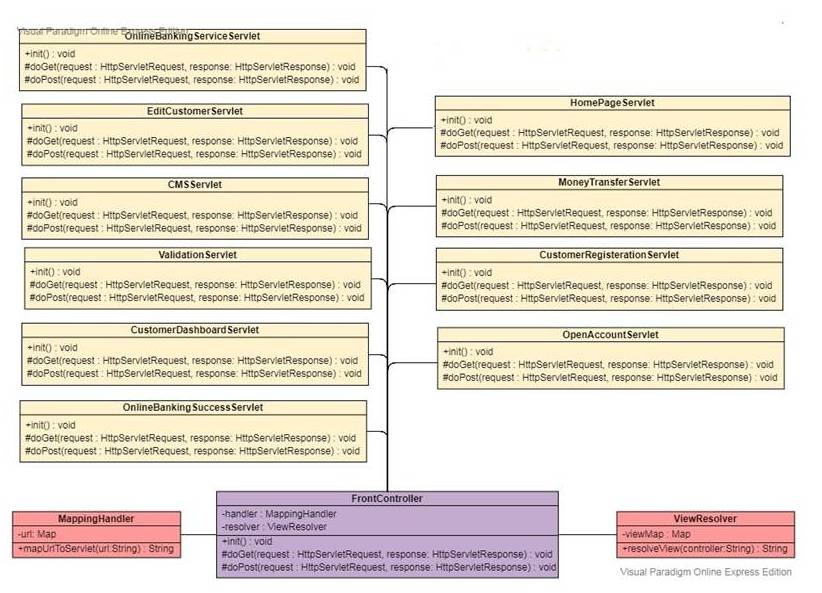
F.1 Data Access Layer Classes and Beans



F.2 Service Layer Classes



F.3 Controller Classes



G: List of Custom Exceptions Handled

We have implemented these CustomExceptions that are thrown by the appropriated classes not shown in the diagram due to space constraints.

|  |
| --- |
| InvalidDataEnteredException CustomerIDNotFoundException InvalidLoginCredentialsException AccountNotFoundException AccountNotActiveException InsufficientBalanceException OverdraftAmountException InvalidTransactionException |

H: Use case diagram

