DESIGN DOCUMENT

This project can described using the following components:

1.Data Design:

The database is populated using a SQL script which builds all the tables based on the schema of the library specified as a part of project requirement. It also inserts data into those tables based on the data provided in the form of .csv files.

2. Architecture Design:

I used a Javascript as a language of choice for both the stacks, namely Frontend and the Backend. NodeJs and specifically Express was chosen for the backend, with Sequelize as the chosen ORM to interface with MySQL and Angular 6 in the frontend to build the library management interface.

Following are the components of the Project:

1. Backend
2. Frontend

1. Backend :

The backend is a typical NodeJs project, exposing the following API endpoints for the front end to use and build a GUI on top of.

```
    app.post('/api/customers', customers.search);
    app.post('/api/customers/addBorrower', customers.addBorrower);
    app.post('/api/customers/addCustomer', customers.addCustomer);
    app.post('/api/customers/checkinBook', customers.checkinBook);
    app.get('/api/customers/refreshFines', customers.refreshFines);
    app.post('/api/customers/getFineList', customers.getFineList);
    app.post('/api/customers/payFine', customers.payFine);
```

All the API endpoint methods are implemented in the customer.controller.js file and these methods interact with the MySQL db using Sequelize ORM. All the API endpoints send data to the frontend in form of typical JSON objects.

2. Frontend :

The frontend is a typical Angular6 projects, with a modular and component based structure with 6 components and the routing of the Application is managed on the frontend.
These are the following valid routes for the application –

```
const routes: Routes = [
   {
     path: 'books',
     component: CustomerComponent
   },
   {
     path: 'books/add',
     component: AddCustomerComponent
   },
   {
    path: 'books/checkout',
    component: CheckBookComponent
  },
  {
    path: 'books/fines',
    component: FinesComponent
  },
   {
     path: 'customers/:id',
     component: CustomerDetailsComponent
   },
   {
     path: '',
     redirectTo: 'books',
     pathMatch: 'full'
   },
];
```

The Application features standard validations and error handling and can considered to be quite robust in its current state. The component folders contain the route specific component code, the html partials and the style sheets.