1. Creating Microservices for account and loan

drizzle.config.ts -

```
import { defineConfig } from "drizzle-kit";
if (!process.env.DATABASE_URL) {
  throw new Error("DATABASE_URL, ensure the database is provisioned");
}

export default defineConfig({
  out: "./migrations",
  schema: "./shared/schema.ts",
  dialect: "postgresql",
  dbCredentials: {
    url: process.env.DATABASE_URL,
  },
});
```

Schema.ts -

```
import { pgTable, text, serial, integer, boolean, decimal } from "drizzle-orm/pg-core";
import { createInsertSchema } from "drizzle-zod";
import { z } from "zod";

export const accounts = pgTable("accounts", {
  id: serial("id").primaryKey(),
  name: text("name").notNull(),
  email: text("email").notNull().unique(),
  phone: text("phone"),
  balance: decimal("balance", { precision: 10, scale: 2 }).default("0.00"),
  status: text("status").notNull().default("active"),
});
```

```
export const loans = pgTable("loans", {
  id: serial("id").primaryKey(),
  accountId: integer("account_id").notNull(),
  type: text("type").notNull(),
  amount: decimal("amount", { precision: 10, scale: 2 }).notNull(),
  interestRate: decimal("interest_rate", { precision: 5, scale: 2 }).notNull(),
  termMonths: integer("term_months").notNull(),
  status: text("status").notNull().default("pending"),
});

export const insertAccountSchema = createInsertSchema(accounts).omit({
  id: true,
});

export const insertLoanSchema = createInsertSchema(loans).omit({
  id: true,
});

export type InsertAccount = z.infer<typeof insertAccountSchema>;
export type Account = typeof accounts.$inferSelect;
export type InsertLoan = z.infer<typeof insertLoanSchema>;
export type Loan = typeof loans.$inferSelect;
```

storage.ts –

```
import { accounts, loans, type Account, type Loan, type InsertAccount, type InsertLoan } from
"@shared/schema";

export interface IStorage {
  // Account methods
```

```
  getAccount(id: number): Promise<Account | undefined>;
  getAllAccounts(): Promise<Account[]>;
  createAccount(account: InsertAccount): Promise<Account>;
  updateAccount(id: number, account: Partial<InsertAccount>): Promise<Account | undefined>;
  deleteAccount(id: number): Promise<boolean>;

  // Loan methods
  getLoan(id: number): Promise<Loan | undefined>;
  getAllLoans(): Promise<Loan[]>;
  getLoansByAccountId(accountId: number): Promise<Loan[]>;
  createLoan(loan: InsertLoan): Promise<Loan>;
  updateLoan(id: number, loan: Partial<InsertLoan>): Promise<Loan | undefined>;
  deleteLoan(id: number): Promise<boolean>;

  // Statistics
  getAccountStats(): Promise<{ total: number; activeAccounts: number }>;
  getLoanStats(): Promise<{ total: number; activeLoans: number; totalValue: string }>;
}

export class MemStorage implements IStorage {
  private accounts: Map<number, Account>;
  private loans: Map<number, Loan>;
  private accountIdCounter: number;
  private loanIdCounter: number;

  constructor() {
    this.accounts = new Map();
    this.loans = new Map();
    this.accountIdCounter = 1;
```

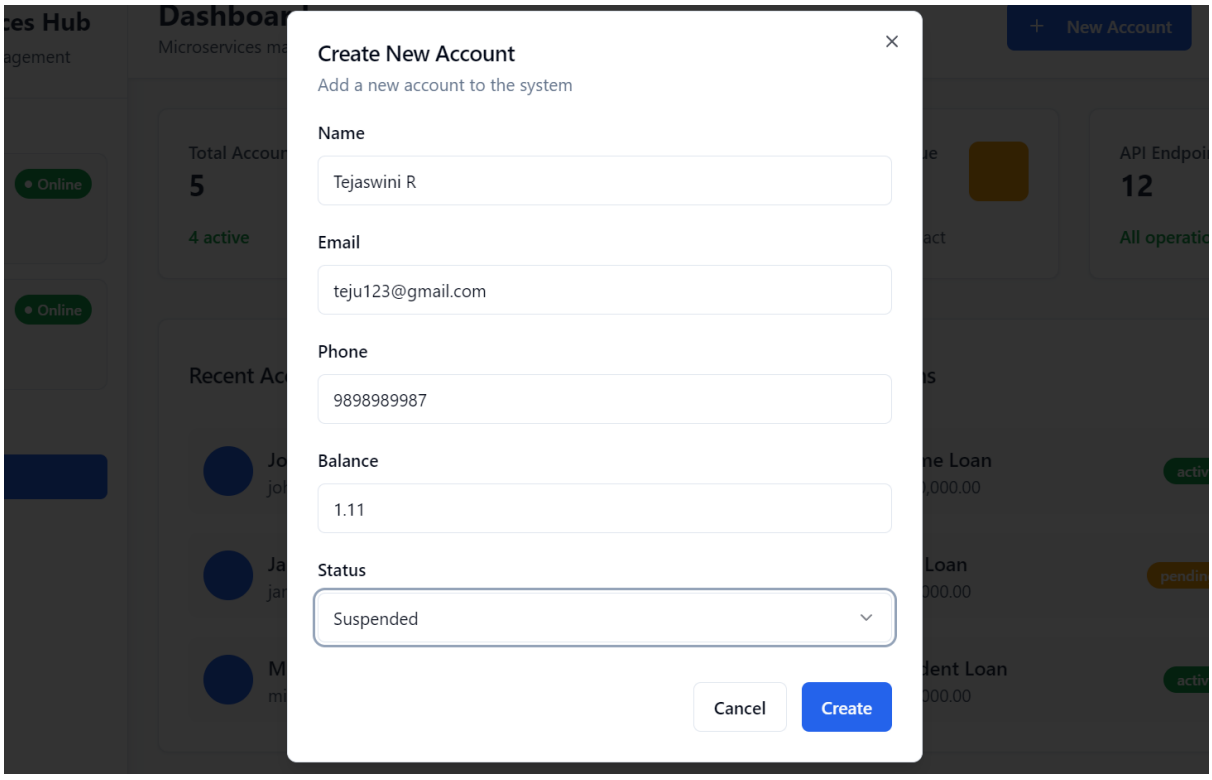
```
    this.loanIdCounter = 1;

    // Initialize with some sample data
    this.initializeData();
  }

  private initializeData() {
    // Sample accounts
    const sampleAccounts = [
      { name: "John Doe", email: "john.doe@example.com", phone: "+1-555-0123", balance: "15000.00", status: "active" },
      { name: "Jane Smith", email: "jane.smith@example.com", phone: "+1-555-0124", balance: "25000.00", status: "active" },
      { name: "Mike Johnson", email: "mike.johnson@example.com", phone: "+1-555-0125", balance: "8500.00", status: "active" },
    ];

    sampleAccounts.forEach(account => {
      const id = this.accountIdCounter++;
      this.accounts.set(id, { ...account, id });
    });

    // Sample loans
    const sampleLoans = [
      { accountId: 1, type: "Home Loan", amount: "250000.00", interestRate: "3.5", termMonths: 360, status: "active" },
      { accountId: 2, type: "Car Loan", amount: "45000.00", interestRate: "4.2", termMonths: 60, status: "pending" },
      { accountId: 3, type: "Student Loan", amount: "85000.00", interestRate: "2.8", termMonths: 120, status: "active" },
    ];
```

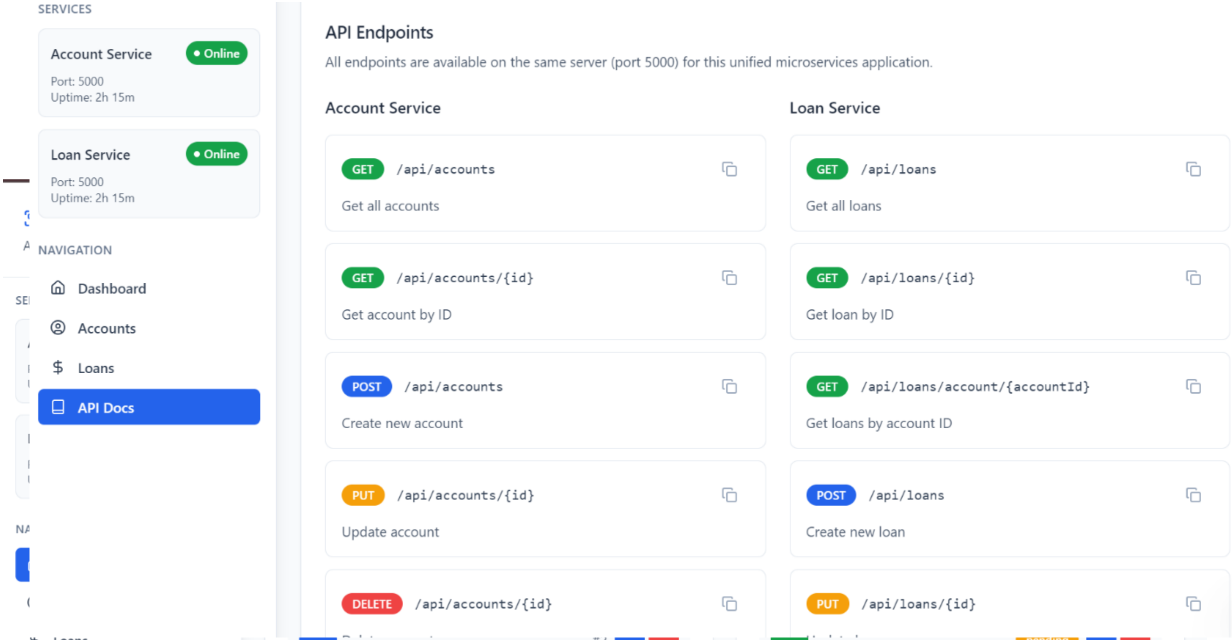
```
  sampleLoans.forEach(loan => {
    const id = this.loanIdCounter++;
    this.loans.set(id, { ...loan, id });
  });
}

// Account methods
async getAccount(id: number): Promise<Account | undefined> {
  return this.accounts.get(id);
}

async getAllAccounts(): Promise<Account[]> {
  return Array.from(this.accounts.values());
}

async createAccount(insertAccount: InsertAccount): Promise<Account> {
  const id = this.accountIdCounter++;
  const account: Account = {
    ...insertAccount,
    id,
    phone: insertAccount.phone || null,
    balance: insertAccount.balance || "0.00",
    status: insertAccount.status || "active"
  };
  this.accounts.set(id, account);
  return account;
}
```

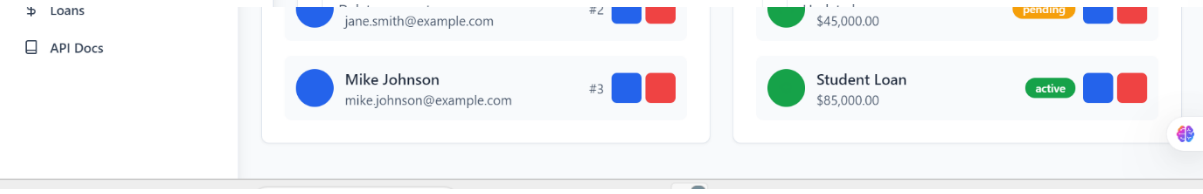
```
async updateAccount(id: number, updateData: Partial<InsertAccount>): Promise<Account | undefined> {
  const account = this.accounts.get(id);
  if (!account) return undefined;

  const updatedAccount = { ...account, ...updateData };
  this.accounts.set(id, updatedAccount);
  return updatedAccount;
}

async deleteAccount(id: number): Promise<boolean> {
  return this.accounts.delete(id);
}

// Loan methods
async getLoan(id: number): Promise<Loan | undefined> {
  return this.loans.get(id);
}

async getAllLoans(): Promise<Loan[]> {
  return Array.from(this.loans.values());
}

async getLoansByAccountId(accountId: number): Promise<Loan[]> {
  return Array.from(this.loans.values()).filter(loan => loan.accountId === accountId);
}

async createLoan(insertLoan: InsertLoan): Promise<Loan> {
  const id = this.loanIdCounter++;
  const loan: Loan = {
```

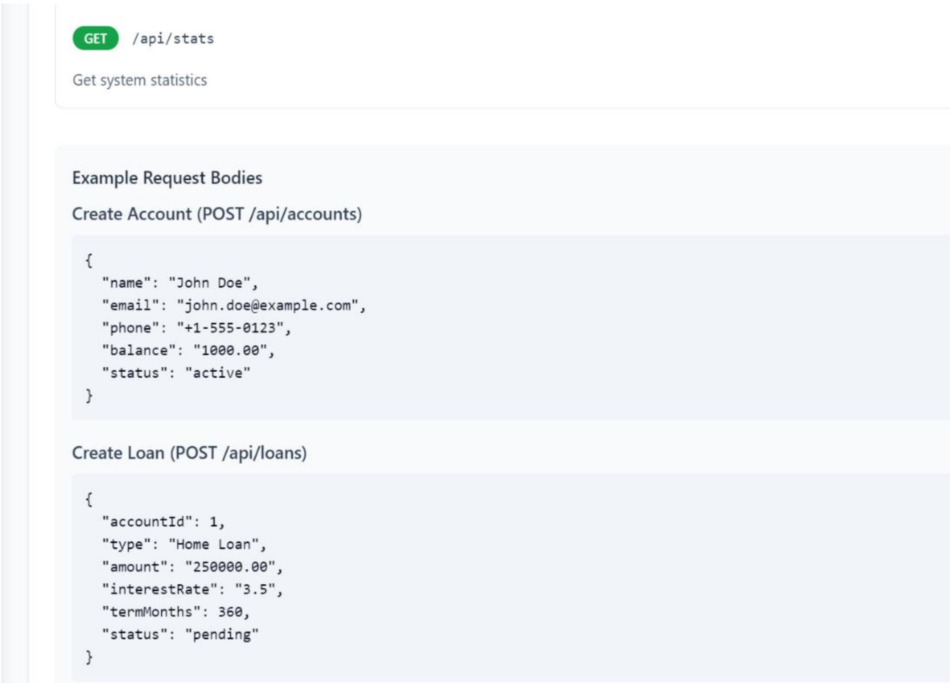
```
    ...insertLoan,
    id,
    status: insertLoan.status || "pending"
  };
  this.loans.set(id, loan);
  return loan;
}

async updateLoan(id: number, updateData: Partial<InsertLoan>): Promise<Loan | undefined> {
  const loan = this.loans.get(id);
  if (!loan) return undefined;

  const updatedLoan = { ...loan, ...updateData };
  this.loans.set(id, updatedLoan);
  return updatedLoan;
}

async deleteLoan(id: number): Promise<boolean> {
  return this.loans.delete(id);
}

// Statistics
async getAccountStats(): Promise<{ total: number; activeAccounts: number }> {
  const allAccounts = Array.from(this.accounts.values());
  return {
    total: allAccounts.length,
    activeAccounts: allAccounts.filter(account => account.status === "active").length,
  };
}
```

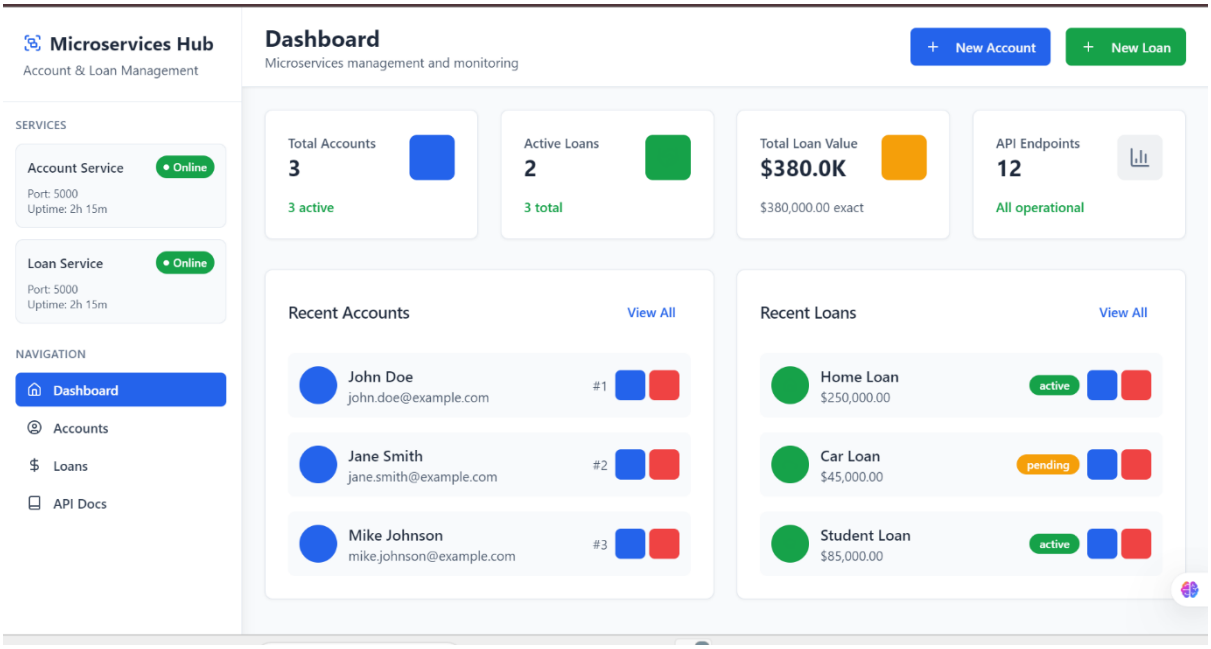
```
  async getLoanStats(): Promise<{ total: number; activeLoans: number; totalValue: string }> {
    const allLoans = Array.from(this.loans.values());
    const activeLoans = allLoans.filter(loan => loan.status === "active");
    const totalValue = allLoans.reduce((sum, loan) => sum + parseFloat(loan.amount), 0);

    return {
      total: allLoans.length,
      activeLoans: activeLoans.length,
      totalValue: totalValue.toFixed(2),
    };
  }
}

export const storage = new MemStorage();
```




SERVICES
Account Service
Online
Port: 5000
Uptime: 2h 15m
Loan Service
Online
Port: 5000
Uptime: 2h 15m
NAVIGATION
Dashboard
Accounts
Loans
API Docs
API Endpoints
All endpoints are available on the same server (port 5000) for this unified microservices application.
Account Service
GET /api/accounts
Get all accounts
GET /api/accounts/{id}
Get account by ID
POST /api/accounts
Create new account
PUT /api/accounts/{id}
Update account
DELETE /api/accounts/{id}
Loan Service
GET /api/loans
Get all loans
GET /api/loans/{id}
Get loan by ID
GET /api/loans/account/{accountId}
Get loans by account ID
POST /api/loans
Create new loan
PUT /api/loans/{id}


Loans
API Docs
jane.smith@example.com
Mike Johnson
mike.johnson@example.com
#3
$45,000.00
pending
Student Loan
$85,000.00
active


GET /api/stats
Get system statistics
Example Request Bodies
Create Account (POST /api/accounts)
{
"name": "John Doe",
"email": "john.doe@example.com",
"phone": "+1-555-0123",
"balance": "1000.00",
"status": "active"
}
Create Loan (POST /api/loans)
{
"accountId": 1,
"type": "Home Loan",
"amount": "250000.00",
"interestRate": "3.5",
"termMonths": 360,
"status": "pending"
}



Video link : https://drive.google.com/file/d/1eLoSAaUz5SlLAq-YBposJzqJEQokiYdg/view?usp=drivesdk