**LoanEligibilityApiApplication.java**

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class LoanEligibilityApiApplication {

public static void main(String[] args) {

SpringApplication.run(LoanEligibilityApiApplication.class, args);

}

}

**LoanRequestDTO.java**

public class LoanRequestDTO {

public double monthlyIncome;

public double existingLoanObligations;

public int creditScore;

public double requestedLoanAmount;

}

**LoanResponseDTO.java**

public class LoanResponseDTO {

public boolean eligible;

public Double approvedLoanAmount;

public Map<String, Double> emiBreakdown;

public String reason;

public LoanResponseDTO(boolean eligible, Double approvedLoanAmount, Map<String, Double> emiBreakdown, String reason) {

this.eligible = eligible;

this.approvedLoanAmount = approvedLoanAmount;

this.emiBreakdown = emiBreakdown;

this.reason = reason;

}

}

**AdminStatisticsDTO.java**

public class AdminStatisticsDTO {

public double averageApprovalAmount;

public Map<String, Integer> topRejectionReasons;

public int totalProcessedRequests;

public AdminStatisticsDTO(double averageApprovalAmount, Map<String, Integer> topRejectionReasons, int totalProcessedRequests) {

this.averageApprovalAmount = averageApprovalAmount;

this.topRejectionReasons = topRejectionReasons;

this.totalProcessedRequests = totalProcessedRequests;

}

}

**LoanController.java**

@RestController

@RequestMapping("/loans")

public class LoanController {

private final LoanEligibilityService loanEligibilityService;

public LoanController(LoanEligibilityService loanEligibilityService) {

this.loanEligibilityService = loanEligibilityService;

}

@PostMapping("/eligibility")

public LoanResponseDTO checkLoanEligibility(@RequestBody LoanRequestDTO request) {

return loanEligibilityService.calculateEligibility(request);

}

@GetMapping("/admin/loan-statistics")

public AdminStatisticsDTO getLoanStatistics() {

return loanEligibilityService.getStatistics();

}

}

**LoanEligibilityService.java**

import org.springframework.web.bind.annotation.\*;

import java.util.\*;

@Service

public class LoanEligibilityService {

private final List<LoanRequestDTO> processedLoans = new ArrayList<>();

private final Map<String, Integer> rejectionReasons = new HashMap<>();

public LoanResponseDTO calculateEligibility(LoanRequestDTO request) {

if (request.monthlyIncome < 30000) {

incrementRejectionReason("lowMonthlyIncome");

return new LoanResponseDTO(false, null, null, "Monthly income is below the required minimum.");

}

if (request.existingLoanObligations > 0.4 \* request.monthlyIncome) {

incrementRejectionReason("highExistingObligations");

return new LoanResponseDTO(false, null, null, "Existing loan obligations exceed 40% of monthly income.");

}

if (request.creditScore < 700) {

incrementRejectionReason("lowCreditScore");

return new LoanResponseDTO(false, null, null, "Credit score is below the required minimum.");

}

double maxLoanAmount = 10 \* request.monthlyIncome;

double approvedLoanAmount = Math.min(request.requestedLoanAmount, maxLoanAmount);

Map<String, Double> emiBreakdown = calculateEmiBreakdown(approvedLoanAmount);

processedLoans.add(request);

return new LoanResponseDTO(true, approvedLoanAmount, emiBreakdown, null);

}

public AdminStatisticsDTO getStatistics() {

double totalApprovalAmount = processedLoans.stream()

.filter(loan -> loan.requestedLoanAmount <= 10 \* loan.monthlyIncome)

.mapToDouble(loan -> loan.requestedLoanAmount)

.sum();

int totalProcessed = processedLoans.size();

double averageApprovalAmount = totalProcessed > 0 ? totalApprovalAmount / totalProcessed : 0;

return new AdminStatisticsDTO(averageApprovalAmount, rejectionReasons, totalProcessed);

}

private void incrementRejectionReason(String reason) {

rejectionReasons.put(reason, rejectionReasons.getOrDefault(reason, 0) + 1);

}

private Map<String, Double> calculateEmiBreakdown(double loanAmount) {

Map<String, Double> emiBreakdown = new HashMap<>();

int tenure = 12; // 12 months

emiBreakdown.put("8%", calculateEmi(loanAmount, 8, tenure));

emiBreakdown.put("10%", calculateEmi(loanAmount, 10, tenure));

emiBreakdown.put("12%", calculateEmi(loanAmount, 12, tenure));

return emiBreakdown;

}

private double calculateEmi(double principal, double annualRate, int months) {

double monthlyRate = annualRate / (12 \* 100);

return (principal \* monthlyRate \* Math.pow(1 + monthlyRate, months)) /

(Math.pow(1 + monthlyRate, months) - 1);

}

}