Loan Eligibility & EMI Calculator - Technical Report

Group Members: Thavamohan Thananchayan, Member 2 (update), Member 3 (update), Member 4 (update). Replace the placeholders with the remaining team members before submission.

Project Title: Functional Loan Eligibility & EMI Calculator with a concurrent scenario simulator.

Problem Statement & Industrial Motivation

Retail lenders and NBFCs must evaluate thousands of unsecured and secured loan requests each day. Lending policies focus on the borrower's cash flows, debt burden, and credit band to avoid defaults. The project delivers a deterministic decisioning microservice that collects applicant and loan details, validates the request, estimates the EMI, predicts debt-to-income (DTI) ratios, and emits an approval or rejection explanation.

The domain is industrially relevant because straight-through processing engines and loan origination systems demand auditable logic. Functional programming strengths—pure functions, algebraic data types, and ability to evaluate multiple scenarios in parallel—mirror the resilience and throughput needs of such production systems.

Functional Design

The system is decomposed into small, pure functions housed in dedicated modules:

```
emiAmount :: LoanRequest -> Double    -- pure EMI computation using recursion
emiSummary :: LoanRequest -> EmiResult    -- aggregates EMI, total payback, interest
validateApplicant :: Applicant -> [String]    -- returns structural issues only
validateLoan :: LoanRequest -> [String]    -- guards unrealistic rates/tenure
decideEligibility :: Applicant -> LoanRequest -> (Decision, Maybe EligibilityMetrics)    --
batchEvaluate :: [LoanScenario] -> [ScenarioEvaluation]    -- parallel map over immutable s
pow :: Double -> Int -> Double    -- recursive exponentiation used by emiAmount
```

DataTypes.hs captures the domain via algebraic data types such as Applicant, LoanRequest, Decision, EmiResult, EligibilityMetrics, LoanScenario, and ScenarioEvaluation. These ADTs make illegal states unrepresentable while also deriving NFData for parallel evaluation.

Application of FP Concepts

- Purity & Referential Transparency: Processing.emiSummary, Processing.decideEligibility, and Processing.batchEvaluate are pure and deterministic. Given the same Applicant and LoanRequest they always return the same decision and metrics.
- Recursion & Higher-order Functions: pow demonstrates explicit recursion, while batchEvaluate relies on the higher-order parMap from Control.Parallel.Strategies to evaluate multiple scenarios concurrently.
- Algebraic Data Types & Pattern Matching: Decision encapsulates Approved vs. Rejected outcomes together with explanatory strings, enabling the IO layer to pattern match and produce human-readable narratives.
- Immutability & Pipeline Composition: Validation, EMI calculation, affordability metrics, and DTI checks form a pipeline where each stage receives immutable inputs and produces new values without side effects.
- Parallelism: batchEvaluate orchestrates parMap rdeepseq so that demo scenarios can be evaluated independently, showcasing the ease of exploiting multicore CPUs from pure code.

Expected Outputs & Workflow

Interactive Flow: The CLI collects monthly income, expenses, existing EMIs, and credit band information, followed by the desired loan amount, interest rate, and tenure. The system prints approval/rejection together with EMI, total payoff, disposable income before/after the loan, and the projected DTI percentage.

Parallel Demo: Users can trigger a scenario simulator that evaluates four contrasting applicants simultaneously. The output lists each scenario label, applicant profile, result, and affordability metrics, proving that parallel evaluation yields the same deterministic answers as the single-scenario pipeline.

Possible Extensions

- Persist applicant histories in a SQLite/PostgreSQL layer and stream them through the same Processing.decideEligibility pipeline for audit trails.
- Import CSV batches and reuse batchEvaluate to produce entire day-level decisions.
- Integrate risk-based pricing by tweaking annualRate using scorecards derived from the Applicant and LoanRequest ADTs.
- Expose the pure logic as a servant/Yesod web API while keeping Testing via HSpec or QuickCheck to assert EMI correctness.

Reliability & Concurrency Discussion

Pure functions make correctness reasoning straightforward: EMI calculations and DTI ratios rely solely on their inputs, so reproducibility and unit testing are trivial. Because the processing pipeline is side-effect free, it can be exercised concurrently without locks or shared state. The parallel batch evaluator illustrates how immutable LoanScenario values can be mapped across CPU cores (parMap rdeepseq) to deliver higher throughput without changing business logic.

Rejection reasons are deterministic lists, yielding auditable artifacts that align with industrial compliance expectations. FP's structural typing and ADTs prevent malformed cases—tenure cannot be negative, credit bands are constrained to a closed set, and validation functions detect inconsistent numbers before the EMI math runs.