**Project Plan: TORM – A Lightweight ORM Library for Go**

1. **Project Overview**

**What Do I Want to Do?**

* **Objective**: Develop a new ORM library for Go that simplifies and improves upon existing solutions.
* **TORM** (working title).

**Why?**

* **Dissatisfaction with GORM**: Existing issues such as excessive complexity, performance overhead, or inflexible APIs have driven the need for a better alternative.
* **Performance**: A leaner, more efficient library that minimizes overhead.
* **Simplicity & Usability**: An intuitive API that aligns with Go’s idioms, making it easier for developers to perform common database operations.
* **Customizability**: Greater control over SQL queries and mappings without sacrificing ease of use.

**How Will I Do It?**

* **Research & Analysis**: Examine current ORM solutions (GORM, sqlx, xorm, ent, etc.) to identify strengths and weaknesses.
* **Design & Prototyping**: Define a clear, minimal API with focused features. Create design sketches and digital diagrams to visualize the architecture.
* **Incremental Development**: Build a Proof of Concept (PoC) for key functionalities (CRUD operations, struct mapping, transaction management) with testing and feedback loops.
* **Collaboration**: Engage with peers and experts (e.g., within Fontys ICT) for feedback throughout the project lifecycle.
* **Version Control & Documentation**: Use Git for code management and ensure comprehensive documentation and tests.

1. **Analysis**

**What's Already Available?**

* **Existing Libraries**:
  + **GORM**: Feature-rich but can be overly complex and heavy.
  + **xorm**, **ent**, **sqlboiler**, **sqlx**: Each with their own trade-offs in terms of performance, usability, and flexibility.

**Similar Examples & Best Practices**

* **API Design**: Look into how sqlx handles simple query executions with minimal overhead and how ent enforces schema integrity.
* **Best Practices**:
  + **Separation of Concerns**: Differentiate between the query builder, connection management, and schema migrations.
  + **Idiomatic Go**: Leverage Go’s native database/sql package and design patterns.
  + **Performance & Safety**: Ensure SQL injection prevention, efficient connection pooling, and easy transaction handling.

**Tools, Tutorials, and Resources**

* **Languages & Frameworks**: Go (latest stable release) with its standard library.
* **Supporting Libraries**: Use database/sql alongside popular drivers (e.g., for PostgreSQL, MySQL).
* **Development Tools**: Go modules, unit testing (go test), CI/CD pipelines (e.g., GitHub Actions), and documentation generators.
* **Tutorials/Documentation**: Official Go documentation, community-written guides on ORM design, and best practices for database interactions.

**Experts & Community Feedback**

* **Fontys ICT**: Reach out to internal experts and senior developers with experience in Go and database systems.
* **Peer Reviews**: Regularly schedule feedback sessions with peers to refine requirements and design choices.

**Requirements & Validation**

* **Functional Requirements**:
  + Seamless connection management.
  + Intuitive CRUD operations.
  + Automatic struct-to-table mapping.
  + Custom query building and transaction support.
* **Non-functional Requirements**:
  + **Performance**: Benchmark against GORM and similar libraries.
  + **Usability**: Simple, clear, and well-documented API.
* **Validation Strategy**:
  + Extensive unit and integration tests.
  + Performance benchmarks.
  + Continuous feedback from early adopters and internal experts.

1. **Design**

**Conceptual Sketches and API Wireframes**

* **Architecture Overview**:
  + **Core Package (orm)**: Manages connections, configurations, and basic CRUD operations.
  + **Query Builder (query)**: Provides a fluent interface for constructing SQL queries.
  + **Schema Management (migration)**: Handles migrations and schema updates.
  + **Database Drivers (driver)**: Integrates with various SQL drivers.

**Digital Design & Iterative Feedback**

* **Peer Reviews**: Share initial designs with peers and Fontys ICT experts to ensure the API is intuitive and meets performance goals.
* **Iteration**: Revise designs based on feedback, focusing on simplicity and clarity.

1. **Realisation**

**Development Phases**

1. **Setup & Initialization**:
   * Create a Git repository for version control (e.g. GitHub).
   * Set up initial project structure and Go modules.
2. **Core Functionality**:
   * Develop connection management and basic CRUD operations.
   * Implement automatic struct-to-table mapping.
3. **Extended Features**:
   * Build a flexible query builder.
   * Add support for transactions and error handling.
   * Develop schema migration tools.
4. **Testing & Documentation**:
   * Write comprehensive unit and integration tests.
   * Create example applications and documentation to demonstrate usage.
5. **Feedback & Iteration**:
   * Deploy a PoC (v0.1) for internal testing.
   * Gather feedback from peers and Fontys ICT experts.
   * Iterate on features and design based on test results.

**Demo & PoC**

* **Demo Goals**:
  + Showcase a complete flow: connecting to a database, mapping a Go struct to a table, performing CRUD operations, and executing a custom query.
  + Demonstrate performance improvements and usability over GORM.
* **Availability**: Make the codebase publicly available on Git, ensuring clear commit messages, documentation, and instructions for running the demo.

**Timeline (Example)**

* **Week 1**: Research, analysis, and initial design sketches.
* **Week 2**: Setup repository, establish project structure, and build the basic connection and CRUD functionality.
* **Week 3**: Develop extended features (query builder, transactions) and begin writing tests/documentation.
* **Week 4**: Collect feedback, perform performance benchmarking, and refine the API for v0.1 demo.