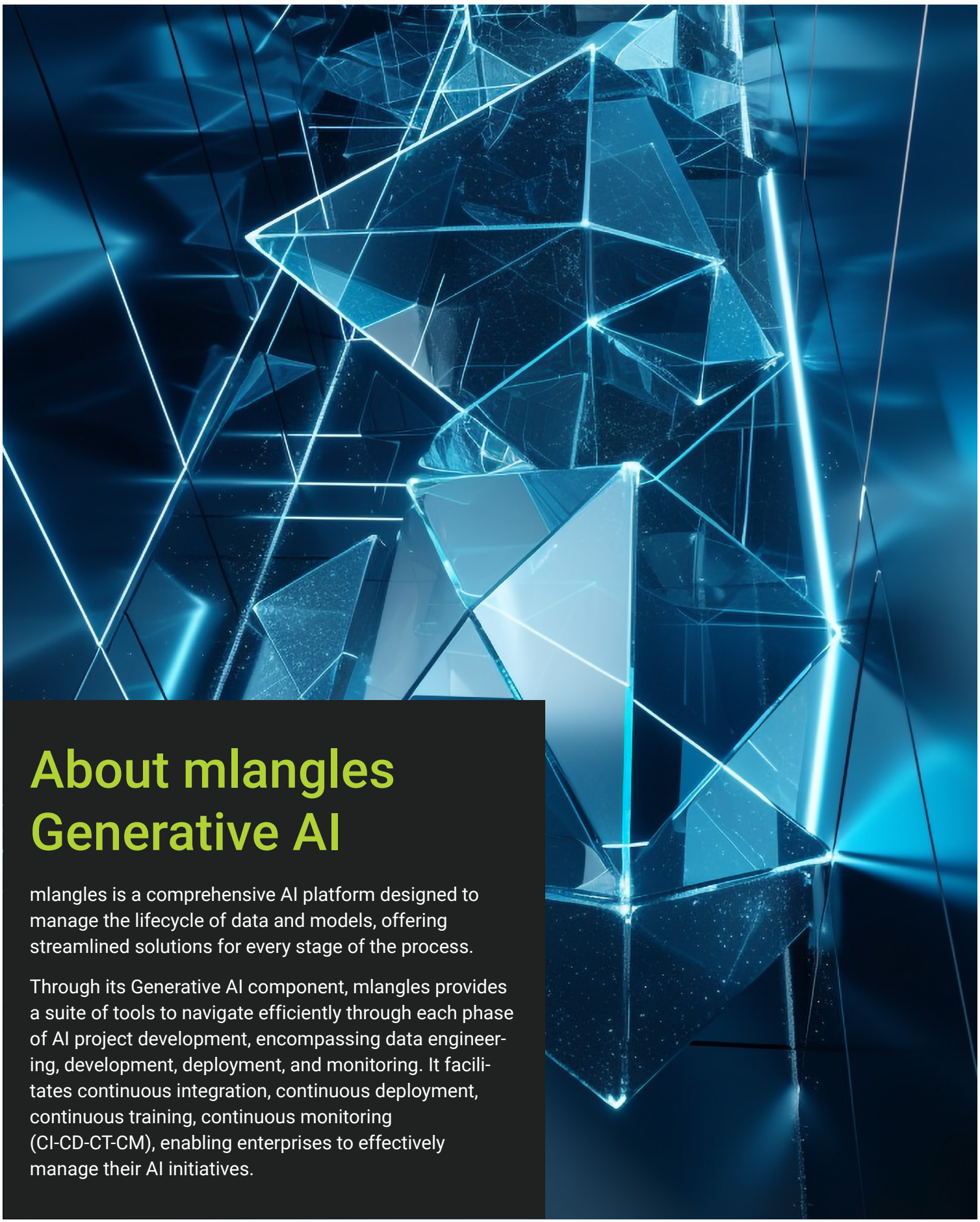
mlangles Generative AI

Java Code Generator

Use Case



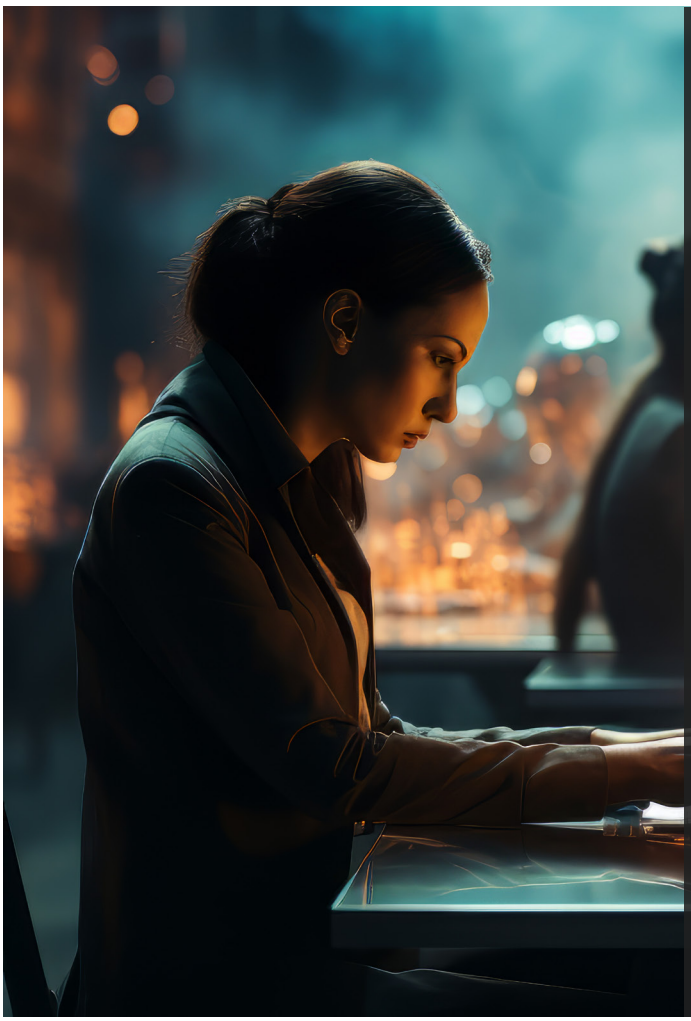
About mlangles Generative AI

mlangles is a comprehensive AI platform designed to manage the lifecycle of data and models, offering streamlined solutions for every stage of the process.

Through its Generative AI component, mlangles provides a suite of tools to navigate efficiently through each phase of AI project development, encompassing data engineering, development, deployment, and monitoring. It facilitates continuous integration, continuous deployment, continuous training, continuous monitoring (CI-CD-CT-CM), enabling enterprises to effectively manage their AI initiatives.

Challenge

Many enterprise applications leverage PL/SQL stored procedures for database interactions. However, as application landscapes evolve and Java becomes a dominant language, maintaining these procedures can be cumbersome. Migrating them to Java offers several advantages.



Solution

This application streamlines the conversion process of PL/SQL stored procedures into well-structured Java code. It automates tedious tasks, reducing development time and effort.

Conversion Process

Upload PL/SQL File: The user selects and uploads the PL/SQL stored procedure file for conversion.

Preprocessing with LLMs: The app may optionally utilize LLMs to preprocess the PL/SQL code. This could involve:

- Code Normalization: LLMs can identify and correct inconsistencies in code formatting or naming conventions, ensuring a more uniform structure for the conversion process.
- Dependency Identification: LLMs can analyze the PL/SQL code to identify dependencies on other stored procedures, functions, or packages. This information can be used to ensure proper handling of these dependencies during conversion.

Parsing and Analysis: The app parses the PL/SQL code, identifying elements like variables, data types, SQL statements, control flow, and function calls.

LLM-Assisted Conversion: Here, LLMs play a crucial role:

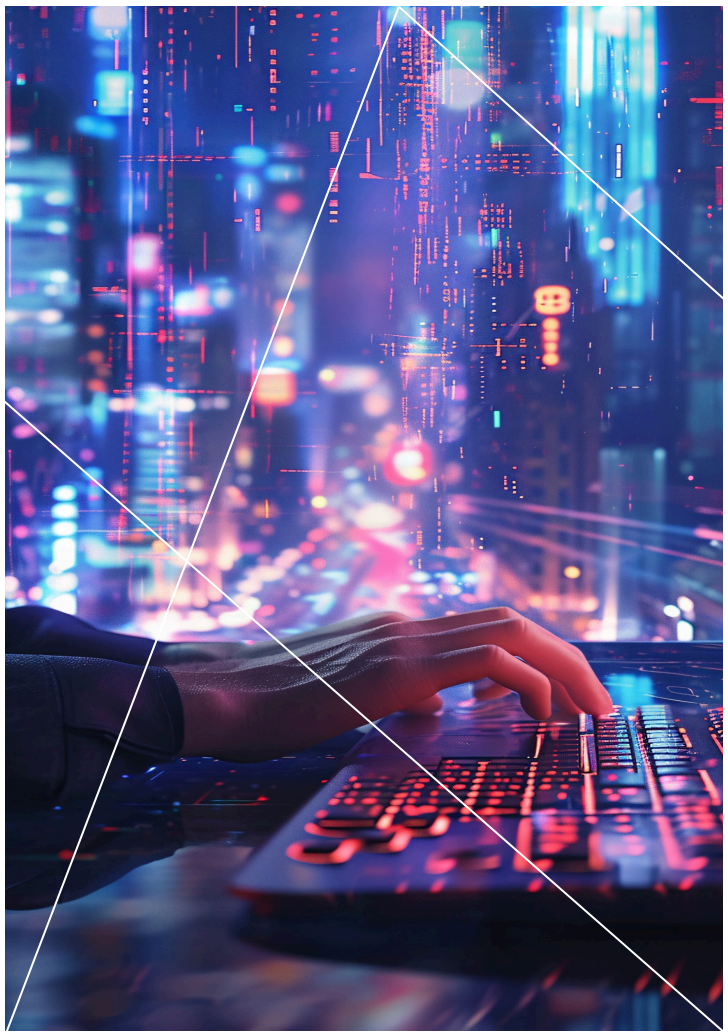
- Code Comprehension: LLMs, trained on vast amounts of code, can understand the logic and intent of the PL/SQL code. This understanding is vital for accurate conversion.
- Syntax Translation: LLMs can translate PL/SQL syntax and semantics into equivalent Java constructs, considering data type mappings, loops, conditional statements, and database interactions.
- Generative AI for Code Generation: Generative AI techniques within the LLMs can be used to create well-structured and idiomatic Java code, adhering to common coding practices and conventions. This promotes readability and maintainability.

Post-Processing:

- Error Handling and Code Improvement: The app analyzes the generated Java code, identifying potential errors or areas for improvement. This may involve suggestions for enhanced error handling, code optimization, or adherence to specific coding standards.

Human Review: While LLMs and Generative AI strive for high accuracy, a human developer may optionally review the generated code for additional quality assurance and potential manual adjustments.

Output Generation: The final, well-formatted Java code is presented within the app's interface for review, along with any suggestions for improvement. A download option allows the user to save the code for integration into their application.



Example

1. Input PL/SQL Code:

```
CREATE OR REPLACE PROCEDURE register_buyer (
    email       IN   VARCHAR,
    fname       IN   VARCHAR,
    lname       IN   VARCHAR,
    password    IN   VARCHAR
) AS
BEGIN
    INSERT INTO amz_user VALUES (
        email,
        fname,
        lname,
        password,
        0
    );

    INSERT INTO buyer VALUES (
        email,
        0,
        NULL
    );

END register_buyer;

-----------------------------------------------------------------------------------------------

CREATE OR REPLACE PROCEDURE register_seller (
    email              IN   VARCHAR,
    fname              IN   VARCHAR,
    lname              IN   VARCHAR,
    password           IN   VARCHAR,
    company_name       IN   VARCHAR,
```

```
BEGIN
    UPDATE buyer
    SET
        is_prime = 0,
        prime_expiry_date = NULL
    WHERE
        buyer_id = buyer_id_input;

END cancel_membership;

-----------------------------------------------------------------------------------------

CREATE OR REPLACE PROCEDURE place_order (
    order_id        IN    INTEGER,
    buyer_id_var    IN    VARCHAR
) AS

    card_id_var             INTEGER;
    address_id_var          INTEGER;
    total_price_var         NUMBER := 0;
    curr_price_var          NUMBER;
    total_qty_var           NUMBER := 0;
    available_units_var     NUMBER(1);
    shipping_price_var      NUMBER := 10;
    is_prime_var            NUMBER := 0;
    CURSOR products_cur IS
    SELECT
        product_id
    FROM
        product_shoppingcart
```

2. Ouput Java Code:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class DatabaseOperations {

    private static final String URL = "jdbc:your_database_url";
    private static final String USER = "your_database_username";
    private static final String PASSWORD = "your_database_password";

    public static void registerBuyer(String email, String fname, String lname, String password) {
        Connection conn = null;
        PreparedStatement pstmt1 = null;
        PreparedStatement pstmt2 = null;

        try {
            conn = DriverManager.getConnection(URL, USER, PASSWORD);
            conn.setAutoCommit(false);

            String sql1 = "INSERT INTO amz_user VALUES (?, ?, ?, ?, ?)";
            pstmt1 = conn.prepareStatement(sql1);
            pstmt1.setString(1, email);
            pstmt1.setString(2, fname);
            pstmt1.setString(3, lname);
            pstmt1.setString(4, password);
            pstmt1.setInt(5, 0);
            pstmt1.executeUpdate();

            String sql2 = "INSERT INTO buyer VALUES (?, ?, ?)";
            pstmt2 = conn.prepareStatement(sql2);
            pstmt2.setString(1, email);
```

```java
public class DatabaseOperations {
    public static void registerSeller(String email, String fname, String lname, String password, String companyName, String url, String
        Connection conn = null;
        PreparedStatement pstmt1 = null;
        PreparedStatement pstmt2 = null;

        try {
            conn = DriverManager.getConnection(URL, USER, PASSWORD);

            String sql1 = "INSERT INTO amz_user VALUES (?, ?, ?, ?, ?)";
            pstmt1 = conn.prepareStatement(sql1);
            pstmt1.setString(1, email);
            pstmt1.setString(2, fname);
            pstmt1.setString(3, lname);
            pstmt1.setString(4, password);
            pstmt1.setInt(5, 1);
            pstmt1.executeUpdate();

            String sql2 = "INSERT INTO seller VALUES (?, ?, ?, ?, ?, ?)";
            pstmt2 = conn.prepareStatement(sql2);
            pstmt2.setString(1, email);
            pstmt2.setString(2, companyName);
            pstmt2.setString(3, url);
            pstmt2.setString(4, descriptionVar);
            pstmt2.setDouble(5, 2.5);
            pstmt2.setInt(6, 0);
            pstmt2.executeUpdate();

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
```

How it Helps Enterprises

Reduced Development Time: This app automates a significant portion of the conversion process, saving developers valuable time that can be focused on more complex tasks and application enhancements.

Improved Maintainability: By migrating to Java, code becomes more familiar to a wider pool of developers, simplifying maintenance and future modifications.

Enhanced Integration: Java's broader ecosystem fosters easier integration with other Java-based components within the application landscape.

Modernization: This conversion aligns your application with current technology trends, potentially improving performance and facilitating future upgrades.

Reduced Reliance on Proprietary Technologies: Migrating away from PL/SQL can lessen dependence on specific database platforms, promoting greater portability.