Design Document for IBAN Validation Web Application

1. Overview

This document describes the architecture, components, and design decisions for the IBAN Validation Web Application.

2. System Architecture

- Frontend: Built using Vue.js for a single-page application (SPA) design.
- Backend: Built using PHP to handle authentication, IBAN validation, and data management.
- Database: A relational database (e.g., MySQL, PostgreSQL) is used to store user information, IBAN numbers, and logs.
- API Communication: RESTful API handles communication between the frontend and backend.

3. Functional Components

Frontend

- User Registration Page: Allows users to sign up.
- Login Page: Authenticates users.
- IBAN Validation Page:
  - Input field for IBAN entry.
  - Results in alert (Valid/Invalid).
- Dashboard:
  - Paginated list of user created IBANs, list all for Admin.
  - Delete option for Admin.

Backend

- Authentication:
  - Register as a general user.
  - User and admin authentication.
  - Token-based security (Laravel/Sanctum used).
- IBAN Validation Module:
  - Algorithm for validating IBAN based on checksum and country code rules.
- Data Management:
  - Stores valid IBANs in the database.

Database Design

- Users Table:
  - id, name, email, password, role (user/admin), created_at, updated_at.
- IBAN Table:
  - id, iban, user_id, created_at, updated_at.

4. Design Decisions

- Validation Logic: The IBAN validation will implement the MOD-97-10 algorithm to ensure compliance without relying on external libraries. Check the explained IBAN validation logic at ./IBAN.md
- API Structure:
  - POST /api/register: User registration.
  - POST /api/login: User login.
  - POST /api/iban: Validate and save IBAN.
  - GET /api/ibans/list: Fetch paginated IBAN list (admin only).
- Error Handling: Returning error responses in JSON format.

5. Technologies Used

- Frontend:
  - Vue.js with Vue Router and Vuex.
- Backend:
  - PHP (Laravel framework recommended for structure and scalability).
- Database:
  - MySQL or PostgreSQL.

6. Security Considerations

- Passwords hashed using a secure algorithm (e.g., bcrypt).
- Input validation to prevent SQL injection and XSS attacks.
- Role-based access control for admin and user functionalities.