**Digital Payment Wallet (Paytm Clone) with MERN Technology**

Creating a **Digital Payment Wallet** (similar to Paytm) using the MERN stack involves building a full-stack application comprising:

1. **Frontend**: React.js for the user interface.
2. **Backend**: Node.js with Express.js for handling API requests.
3. **Database**: MongoDB for storing user data, wallet balances, transactions, etc.
4. **Authentication**: Secure user authentication with JWT or OAuth.
5. **Payment Gateway Integration**: Using APIs like Stripe or Razorpay for transaction processing.

Here's a breakdown of the process with theory and code.

**1. Project Setup**

**Backend Setup:**

* **Initialize the Node.js App**
* mkdir paytm-clone-backend
* cd paytm-clone-backend
* npm init -y
* npm install express mongoose cors body-parser dotenv bcryptjs jsonwebtoken
* **Create the File Structure**:
* paytm-clone-backend/
* ├── server.js
* ├── .env
* ├── models/
* │ ├── User.js
* │ └── Transaction.js
* ├── routes/
* │ ├── auth.js
* │ └── wallet.js
* ├── controllers/
* │ ├── authController.js
* │ └── walletController.js
* └── config/
* └── db.js

**Frontend Setup:**

* **Initialize React App**
* npx create-react-app paytm-clone-frontend
* cd paytm-clone-frontend
* npm install axios react-router-dom bootstrap

**2. Backend Code**

**Database Connection (MongoDB)**

**config/db.js**

const mongoose = require("mongoose");

const connectDB = async () => {

try {

await mongoose.connect(process.env.MONGO\_URI, {

useNewUrlParser: true,

useUnifiedTopology: true,

});

console.log("MongoDB Connected...");

} catch (error) {

console.error("Database connection failed:", error);

process.exit(1);

}

};

module.exports = connectDB;

**Models**

**models/User.js**

const mongoose = require("mongoose");

const bcrypt = require("bcryptjs");

const userSchema = new mongoose.Schema({

name: { type: String, required: true },

email: { type: String, unique: true, required: true },

password: { type: String, required: true },

walletBalance: { type: Number, default: 0 },

});

userSchema.pre("save", async function (next) {

if (!this.isModified("password")) return next();

const salt = await bcrypt.genSalt(10);

this.password = await bcrypt.hash(this.password, salt);

next();

});

module.exports = mongoose.model("User", userSchema);

**models/Transaction.js**

const mongoose = require("mongoose");

const transactionSchema = new mongoose.Schema({

userId: { type: mongoose.Schema.Types.ObjectId, ref: "User", required: true },

amount: { type: Number, required: true },

type: { type: String, enum: ["credit", "debit"], required: true },

timestamp: { type: Date, default: Date.now },

});

module.exports = mongoose.model("Transaction", transactionSchema);

**Routes & Controllers**

**Authentication Route** **routes/auth.js**

const express = require("express");

const { registerUser, loginUser } = require("../controllers/authController");

const router = express.Router();

router.post("/register", registerUser);

router.post("/login", loginUser);

module.exports = router;

**Authentication Controller** **controllers/authController.js**

const User = require("../models/User");

const jwt = require("jsonwebtoken");

exports.registerUser = async (req, res) => {

const { name, email, password } = req.body;

try {

const user = new User({ name, email, password });

await user.save();

res.status(201).json({ message: "User registered successfully" });

} catch (error) {

res.status(500).json({ message: "Registration failed", error });

}

};

exports.loginUser = async (req, res) => {

const { email, password } = req.body;

try {

const user = await User.findOne({ email });

if (!user) return res.status(404).json({ message: "User not found" });

const isMatch = await bcrypt.compare(password, user.password);

if (!isMatch) return res.status(400).json({ message: "Invalid credentials" });

const token = jwt.sign({ userId: user.\_id }, process.env.JWT\_SECRET, { expiresIn: "1h" });

res.status(200).json({ token, walletBalance: user.walletBalance });

} catch (error) {

res.status(500).json({ message: "Login failed", error });

}

};

**Wallet Route** **routes/wallet.js**

const express = require("express");

const { creditWallet, getTransactions } = require("../controllers/walletController");

const router = express.Router();

router.post("/credit", creditWallet);

router.get("/transactions/:userId", getTransactions);

module.exports = router;

**Wallet Controller** **controllers/walletController.js**

const Transaction = require("../models/Transaction");

const User = require("../models/User");

exports.creditWallet = async (req, res) => {

const { userId, amount } = req.body;

try {

const user = await User.findById(userId);

user.walletBalance += amount;

await user.save();

const transaction = new Transaction({ userId, amount, type: "credit" });

await transaction.save();

res.status(200).json({ walletBalance: user.walletBalance });

} catch (error) {

res.status(500).json({ message: "Transaction failed", error });

}

};

exports.getTransactions = async (req, res) => {

const { userId } = req.params;

try {

const transactions = await Transaction.find({ userId });

res.status(200).json(transactions);

} catch (error) {

res.status(500).json({ message: "Failed to fetch transactions", error });

}

};

**Server Entry Point** **server.js**

const express = require("express");

const connectDB = require("./config/db");

const cors = require("cors");

require("dotenv").config();

const app = express();

connectDB();

app.use(express.json());

app.use(cors());

app.use("/api/auth", require("./routes/auth"));

app.use("/api/wallet", require("./routes/wallet"));

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));

**3. Frontend Code**

Build components for:

* Registration/Login
* Viewing wallet balance
* Adding funds
* Viewing transaction history

**4. Run the Project**

1. **Backend**:
   * Run MongoDB (locally or via MongoDB Atlas).
   * Start the server:
   * node server.js
2. **Frontend**:
   * Start the React app:
   * npm start
3. Open http://localhost:3000 for the frontend and interact with your app