## **Full Stack Developer Assessment**

**NOTE**: You can complete this assignment using any Javascript framework/library.

### **Part 1: Coding Challenge**

#### **Task: Build a Simple "Transaction Management" API**

**Objective:**

You are required to create a Django application to manage financial transactions. The system should support operations like creating transactions, viewing transaction history, and updating transaction status.

### **1. Transaction Model**

Create a Django model for a Transaction that represents a financial transaction with the following fields:

* amount: A DecimalField that stores the monetary value of the transaction.
* transaction\_type: A CharField with the values DEPOSIT or WITHDRAWAL to indicate the type of transaction.
* user: A ForeignKey to the built-in User model, indicating the user associated with the transaction.
* timestamp: A DateTimeField that stores the date and time of the transaction.
* status: A CharField with values:
  + PENDING
  + COMPLETED
  + FAILED

### **2. API Endpoints**

Use Django REST Framework (DRF) to implement the following API endpoints for the Transaction model:

1. **POST /api/transactions/**
   * **Description**: Create a new transaction.
   * **Request Body**:
     + amount: The monetary value of the transaction (e.g., 150.00).
     + transaction\_type: Either DEPOSIT or WITHDRAWAL.
     + user: The ID of the user who initiated the transaction (e.g., user\_id).
   * **Response**: Return the newly created transaction, including fields such as transaction\_id, amount, transaction\_type, status, and timestamp.
2. **GET /api/transactions/**
   * **Description**: Retrieve all transactions for a specific user.
   * **Query Parameter**: user\_id: The ID of the user whose transactions are being fetched.
   * **Response**: Return a list of transactions for that user, showing transaction\_id, amount, transaction\_type, status, and timestamp.
3. **PUT /api/transactions/{transaction\_id}/**
   * **Description**: Update the status of an existing transaction.
   * **Request Body**:
     + status: The new status, which must be either COMPLETED or FAILED.
   * **Response**: Return the updated transaction, including transaction\_id, amount, transaction\_type, status, and timestamp.
4. **GET /api/transactions/{transaction\_id}/**
   * **Description**: Retrieve the details of a specific transaction by its transaction\_id.
   * **Response**: Return the transaction’s details, including transaction\_id, amount, transaction\_type, status, and timestamp.

### **Example Input and Output**

#### **1. POST /api/transactions/**

**Request**:

{

"amount": 100.00,

"transaction\_type": "DEPOSIT",

"user": 1

}

**Response**:

{

"transaction\_id": 1,

"amount": 100.00,

"transaction\_type": "DEPOSIT",

"status": "PENDING",

"user": 1,

"timestamp": "2024-11-16T10:30:00Z"

}

#### **2. GET /api/transactions/?user\_id=1**

**Response**:

{

"transactions": [

{

"transaction\_id": 1,

"amount": 100.00,

"transaction\_type": "DEPOSIT",

"status": "PENDING",

"timestamp": "2024-11-16T10:30:00Z"

},

{

"transaction\_id": 2,

"amount": 50.00,

"transaction\_type": "WITHDRAWAL",

"status": "COMPLETED",

"timestamp": "2024-11-15T15:00:00Z"

}

]

}

#### **3. PUT /api/transactions/1/**

**Request**:

{

"status": "COMPLETED"

}

**Response**:

{

"transaction\_id": 1,

"amount": 100.00,

"transaction\_type": "DEPOSIT",

"status": "COMPLETED",

"timestamp": "2024-11-16T10:30:00Z"

}

#### **4. GET /api/transactions/1/**

**Response**:

{

"transaction\_id": 1,

"amount": 100.00,

"transaction\_type": "DEPOSIT",

"status": "COMPLETED",

"timestamp": "2024-11-16T10:30:00Z"

}

### **Part 2: Deployment (Optional: Preferred but not required)**

Once the API is implemented, deploy it to a cloud platform which is available for free. The goal is to make the API accessible from a public URL. Feel free to share the hosted link with the below submission.

### **Evaluation Criteria**

* **Code Quality**: Clean and understandable code with good use of Django and DRF features.
* **Django & DRF Knowledge**: Proper use of Django models, serializers, views, and DRF for building the API.
* **API Validation**: Correct handling of inputs, including proper validation and error responses.
* **Simplicity**: The solution should be simple, clear, and easy to follow.