goto

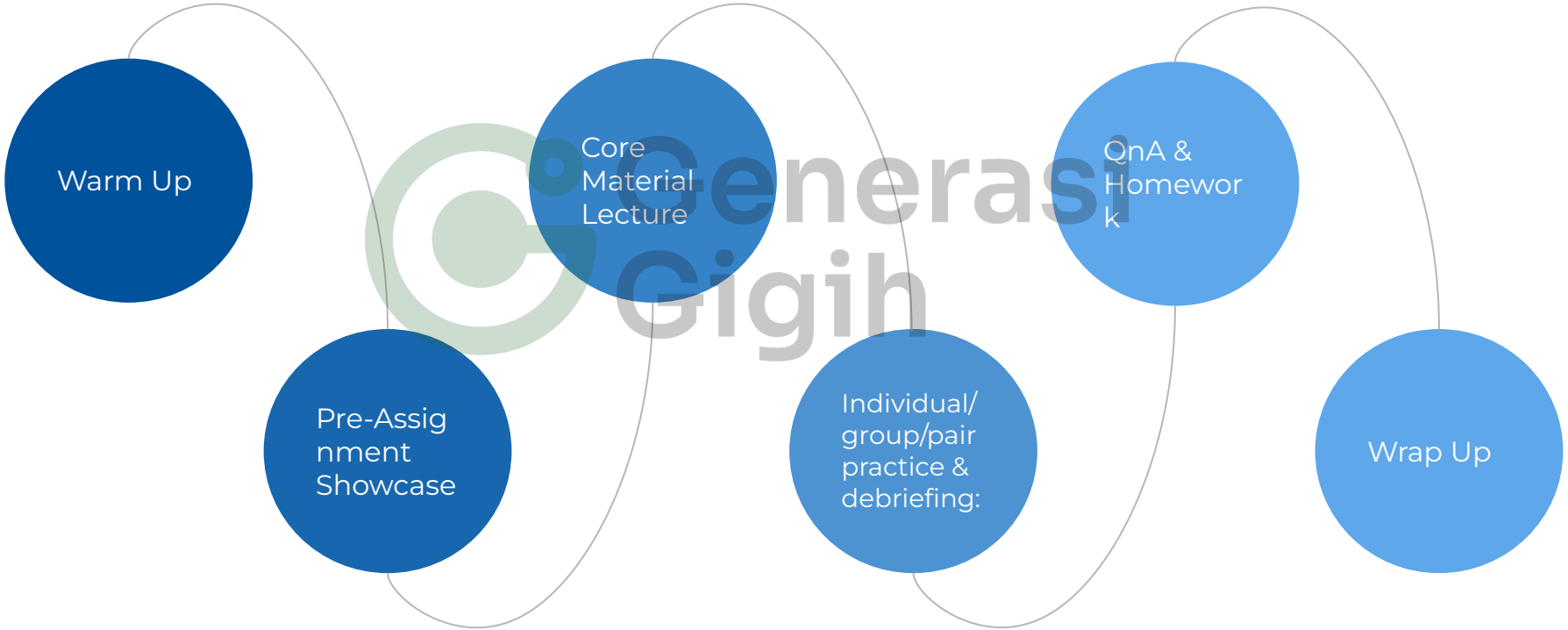
goto impact foundation

Generasi Gigih

Full Stack Engineer

Module 2.3: Intro To REST API and Software Architecture

Our Agenda


Warm Up
Pre-Assignment Showcase
Core Material Lecture
Individual/ group/pair practice & debriefing:
QnA & Homework
Wrap Up

Generasi Gigih

Let’s Warm Up!

Let’s Discuss

[Session outline]

Generasi Gigih

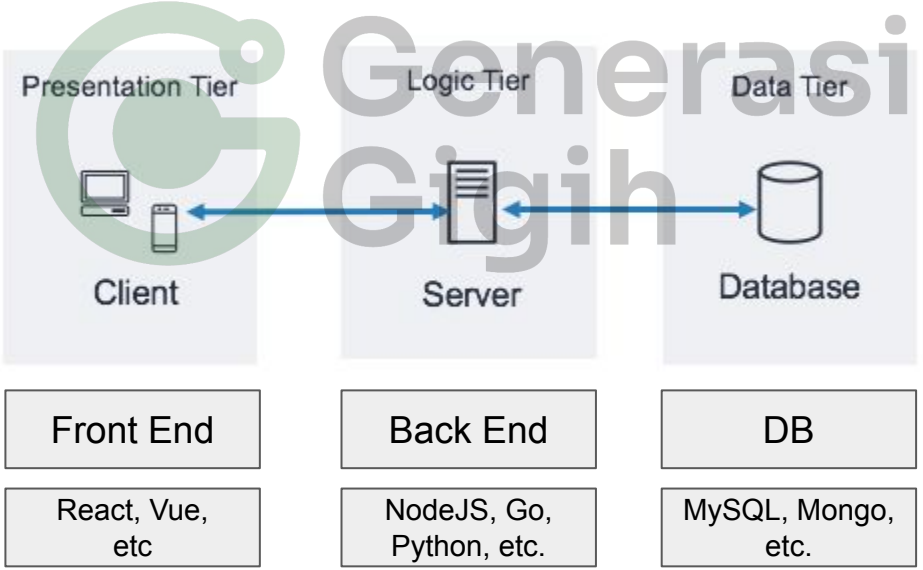
Let’s Talk About The Materials

Generasi Gigih

Software Architecture

The N-Layer Architecture

The most popular and basic architecture in web development is **The Three-Layer Architecture.**



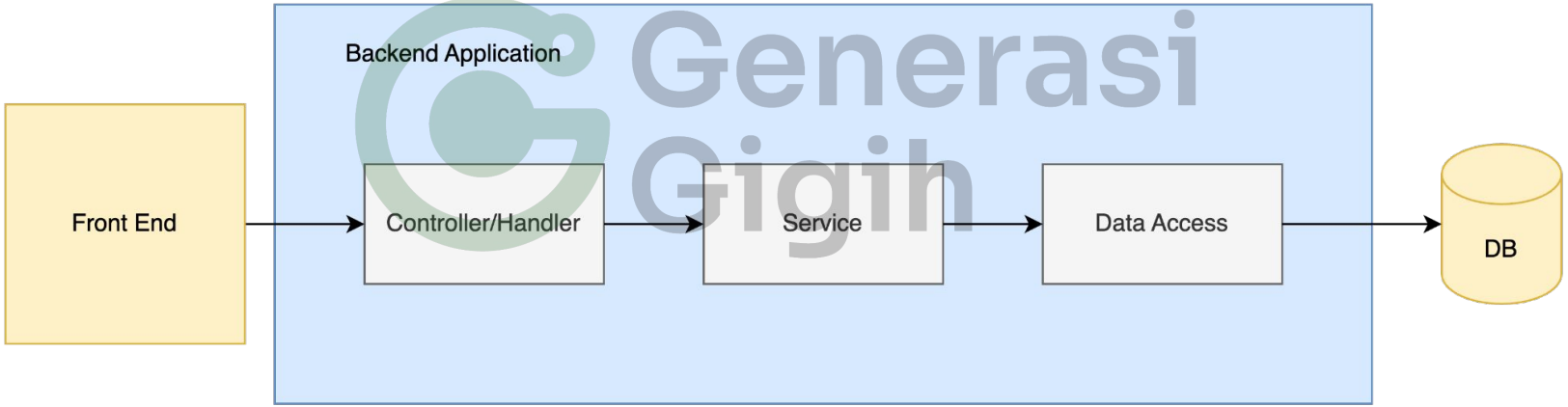
The N-Layer Architecture

Why?

1. Modularity and Separation of Concerns
2. Code Reusability
3. Scalability and Maintainability
4. Flexibility and Agility
5. Testing and Debugging
6. Team Collaboration
7. Interoperability

The N-Layer Architecture, Backend-Side

Backend development itself use the N-Layer concept.



Controller/Handler Layer

Controller acts as an interface between the client and the rest of the application, facilitating the flow of data and coordinating the appropriate actions to be taken. Controller Layer includes the implementation of APIs

Responsibilities:

1. Request Handling
2. Input Validation
3. Service Invocation
4. Response Generation
5. Error Handling

Business Logic / Service Layer

The primary purpose of the business logic layer is to implement and orchestrate the specific operations and workflows required to fulfill the requirements of the application.

Responsibilities:

1. Business Rules Implementation
2. Data Processing and Transformation
3. Workflow Orchestration
4. Interaction with Data Access Layer
5. Business Logic Validation

Data Access Layer

The Data Access Layer is a component or layer in backend development that is responsible for interacting with the underlying data storage mechanisms, such as databases, file systems, or external APIs

Responsibilities include:

1. Data Retrieval
2. Data Manipulation
3. Connection Management
4. Data Mapping
5. Transaction Management

Case Study: Banking System

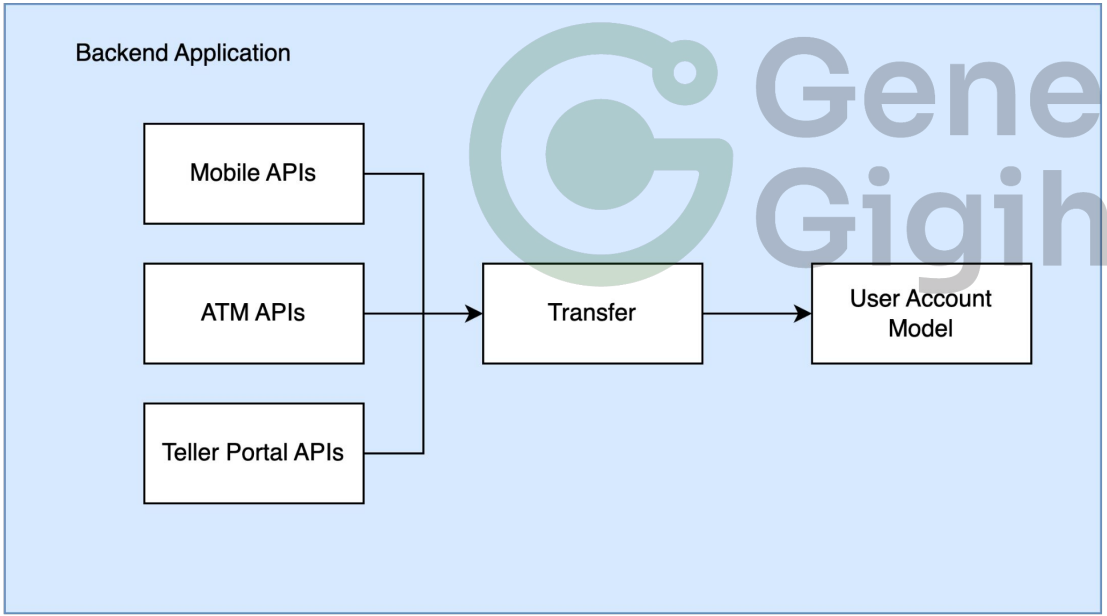
Imagine you're developing a backend application for GIGIH Bank. Let's say there's only one function: transfer money.

Now, there are multiple *user interface* for us to do this:

1. Mobile App
2. ATM
3. Via Bank's Teller

Case Study: Banking System

We'll end up implementing 3 APIs for 3 different *user interface*



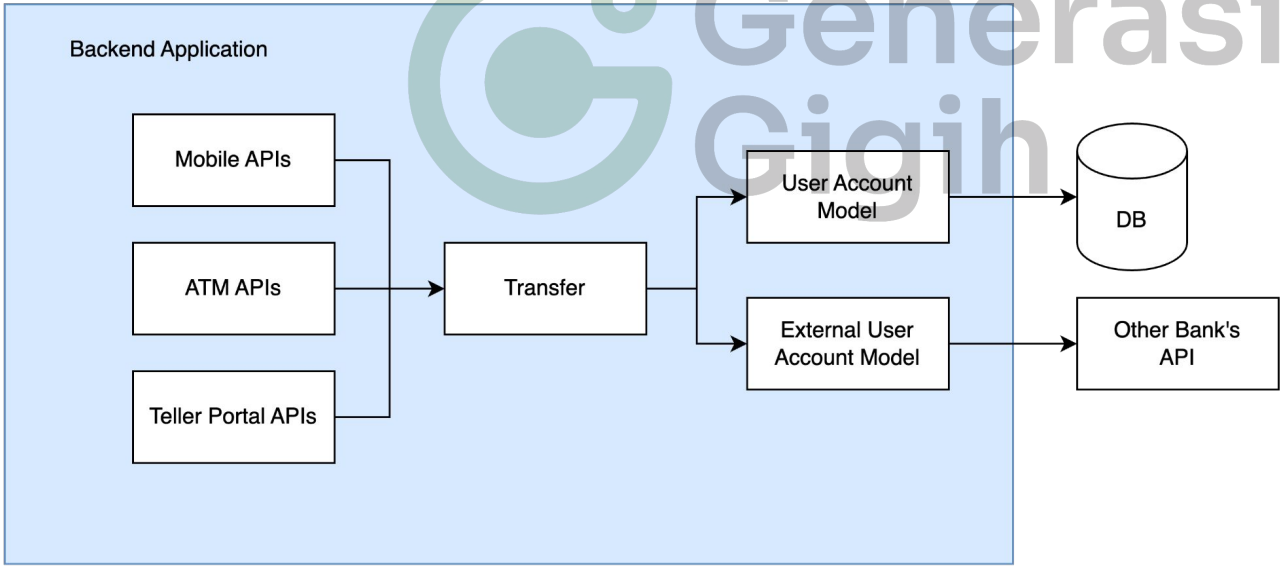
Mobile API → Verify Phone Num, PIN, OTP

ATM API → Verify ATM Card, PIN

Teller Portal API → Manual Verification by Teller

Case Study: Banking System

Then adding features being able to transfer to other bank's, then our system needs to have access to other bank's system.



Generasi Gigih

Hands-on: Transfer Function on Mobile Banking System

Model: Customer

For simplicity, no database will be used used in this sample

To simplify, let's say user can only have one bank account. With these attributes

1. Customer ID
2. Name
3. Email
4. Balance

```
let customers = [
  {
    customerId: '12345',
    name: 'John Doe',
    email: 'johndoe@example.com',
    balance: 5000.00
  }
];
```

Model: Customer

```
function getAllCustomers() {
  return customers;
}
function getCustomer(customerId) {
  return customers.find((c) => c.customerId === customerId);
}
function createCustomer(name, email, initialBalance) {
  let newCustomer = {
    customerId: generateCustomerId(),
    name: name,
    email: email,
    balance: initialBalance
  }
  return newCustomer
}
```

```
function generateCustomerId() {
  return Math.random().toString(10).substr(2,6)
}
```

Model: Transaction

Transaction model will have these attributes:

1. Transaction ID
2. Source ID
3. Destination ID
4. Amount
5. Timestamp

```
let transactions = [
  {
    transactionId: '123456789',
    sourceId: "12345",
    destinationId: "67890",
    amount: 5.0,
    timestamp: "2023-06-01T00:00:00.000Z"
  }
];
```

Model: Transaction

```
function createTransaction(sourceId, destinationId, amount) {
  const transaction = {
    transactionId: generateTransactionId(),
    sourceId,
    destinationId,
    amount: ,
    timestamp: new Date().toISOString()
  };
  transactions.push(transaction);
  return transaction;
}

// Helper function to generate a unique transaction ID
function generateTransactionId() {
  // Generate a random string or use a unique ID generation algorithm
  return Math.random().toString(36).substr(2, 9);
}
```

Service: Transaction

```
function transfer(sourceId, destinationId, amount) {
  sourceAccount = getCustomer(sourceId);
  destinationAccount = getCustomer(destinationId);
  if(!sourceAccount || !destinationAccount) {
    throw new Error("Invalid source or destination account");
  }
  if (sourceAccount.balance < amount) {
    throw new Error("Insufficient balance in the source account")
  }
  sourceAccount.balance -= amount;
  destinationAccount.balance += amount;
  createTransaction(sourceAccount.customerId, destinationAccount.destinationId, amount);
  return
}
```

Controller: Transaction

```
app.post("/transactions", (req,res) => {
  try {
    const { sourceAccount, destinationAccount, amount } = req.body;
    if(!sourceAccount || !destinationAccount || !amount) {
      throw new Error("Insufficient Parameter")
    }
    transfer(sourceAccount, destinationAccount, amount)
    res.status(201).json({message: "Transaction created successfully"})
  } catch(e) {
    //For example we'll always use code 500 (Internal Server Error)
    res.status(500).json({error: e.message})
  }
});
```

Generasi Gigih

Q&A!

Generasi Gigih

Homework

Simple Spotify Playlist Server

Continuing previous session homework with this additional rule:

1. Make playlist as a model
2. Track song play count in the playlist
3. Add feature to Get list of songs to be sorted by most played

Generasi Gigih

Finally, Let’s Wrap Up!