**System Architectures:**

* **N-Tier Architecture**
* **Microservices Architecture**

**N-Tier Architecture (Monolithic)**

**What is it?**

N-Tier involves dividing the application into different layers:

1. **Presentation Layer**: The front end (React).
2. **Business Logic Layer**: The core functionality (Node.js).
3. **Data Layer**: The database (postgresql, firebase).

**Advantages:**

* **Simpler to set up**: Easier for beginners and smaller teams. All components are in a single application.
* **Less complexity**: You have one application to deploy and manage.
* **Easier to maintain**: Less overhead compared to microservices. You don’t need to manage multiple services.
* **Faster development**: Good for small-to-medium-sized applications.

**Disadvantages:**

* **Harder to scale**: As the app grows, it becomes difficult to scale specific parts independently.
* **Limited flexibility**: Changes in one part might affect others, and everything is tightly coupled.

**Microservices Architecture**

**What is it?**

Microservices involve breaking down the application into smaller, independent services that communicate with each other via APIs or messaging systems. Each service focuses on a specific feature (e.g., user management, product listing).

**Advantages:**

* **Scalability**: You can scale specific parts of your app (like user management or orders) independently based on demand.
* **Flexibility**: Each service can be developed and deployed separately, and you can use different technologies for different services.
* **Better for large apps**: Suitable for applications that are complex and need independent scaling or continuous deployment.

**Disadvantages:**

* **Complex to set up**: It requires managing multiple services, each with its own deployment, database, etc.
* **More overhead**: Managing communication between services, service discovery, and consistency can be challenging.
* **Requires more resources**: Running multiple services increases infrastructure and maintenance costs.

**Which is Easier?**

* **N-Tier Architecture** is simpler, more straightforward, and easier to maintain, especially for smaller teams or smaller applications.

**Comparison: N-Tier vs. Microservices**

| **Aspect** | **N-Tier** | **Microservices** |
| --- | --- | --- |
| **Architecture** | Monolithic, with layers (Presentation, Business, Data) | Decentralized, with independent services |
| **Fault Tolerance** | Single failure in one layer can affect the entire system | Isolated failures: other services continue functioning |
| **Development Flexibility** | Tightly coupled components | Independent, loosely coupled services |
| **Communication** | Direct communication between layers | API calls or messaging queues between microservices |

**Implementation of N-Tier:**

* **Frontend (React)**: The frontend calls the backend APIs, sends user input, and receives responses to update the UI. For example, customers browse products, submit orders, and artists manage their galleries. React handles the presentation of this data.
* **Backend (Node.js)**: This layer consists of the business logic that handles:
  + User registration/login (Artist/Customer/Admin)
  + Artist product management (Add/Edit/Delete)
  + Order management (Add to cart, checkout)
  + Auction management (Creating and viewing bids)
  + customization options
  + Sales tracking and analytics
* **Database (MySQL)**: A relational database where all the data is stored, including:
  + User accounts
  + Products (with details like price, descriptions, images)
  + Orders (including statuses like pending, shipped, delivered.)
  + Auction bids and items
  + Reviews and ratings

**Example flow in N-Tier:**

1. **Customer browses products on client.**
2. **client (Frontend) makes an HTTP request to the Node.js server.**
3. **Node.js processes the request (e.g., check available stock, fetch product details, etc.).**
4. **Node.js queries MySQL database to fetch or update data (e.g., fetch products).**
5. **Node.js sends the data back to client, which updates the UI.**

**Microservices Breakdown for Your Website:**

1. **User Service**:
   * Handles user authentication, profile management, registration, login, etc.
   * Stores user-related data such as email, username, password (hashed), and profile details in a separate database (or table).
   * Can expose APIs like POST /register, POST /login, GET /profile.
2. **Product Service**:
   * Manages products, including adding, editing, deleting, and listing products.
   * Stores product data (e.g., name, description, price, images, availability) in a separate database.
   * Can expose APIs like POST /product, PUT /product/{id}, DELETE /product/{id}, GET /products.
3. **Order Service**:
   * Handles cart management, order creation, order updates, and order tracking.
   * Manages order details, including which products have been ordered, the status of each order (e.g., shipped, delivered), and payments.
   * Can expose APIs like POST /order, PUT /order/{id}, GET /orders/{userId}.
4. **Auction Service**:
   * Manages auction listings, bids, and bid tracking.
   * Stores auction-related data, including auction start and end times, bids, and winning bids.
   * Can expose APIs like POST /auction, PUT /auction/{id}, GET /auctions.
5. **Analytics Service**:
   * Tracks the sales, views, and visitor count for artists.
   * Generates reports and insights (e.g., total sales, best-selling products).
   * Can expose APIs like GET /analytics/sales.
6. **Review Service**:
   * Handles product reviews and ratings.
   * Allows customers to leave feedback for products they've purchased.
   * Can expose APIs like POST /review, GET /reviews/{productId}.

Resources

https://www.geeksforgeeks.org/when-to-choose-which-architecture-for-system-design/