Unit 1

Q1. Explain the core concepts of Angular 17: Components, Services, and Routing.

Answer:
Angular is a TypeScript-based framework used to build single-page applications with a component-based architecture.

• **Components:** They are the basic building blocks of Angular that control a part of the user interface and consist of an HTML template, TypeScript logic, and CSS styles.
• **Services:** Services are used to share data and business logic across multiple components, promoting code reusability and separation of concerns.
• **Routing:** Routing enables navigation between different views without reloading the page and supports SPA behavior.
• Router outlet dynamically loads components, while RouterLink helps users navigate between routes.
• Together, these concepts improve application structure, user experience, and maintainability.

Q2. What is Lazy Loading in Angular? Explain how it improves performance with an example.

Answer:
Lazy Loading is a design pattern where feature modules are loaded only when required instead of loading the entire application at startup.

• It reduces initial load time and improves performance, especially for large applications.
• Feature modules such as AdminModule or UserModule are loaded when the user navigates to their routes.
• Implementation involves creating a module, defining routes with RouterModule.forChild(), and configuring loadChildren.
• Example: The admin module loads only when the user visits the /admin route.
• Benefits include faster initial load, better resource utilization, and improved scalability.
• Hence, lazy loading optimizes performance and enhances user experience.

Q3. Differentiate between Smart Components and Dumb Components in Angular.

Answer:
Angular follows separation of concerns by dividing components into Smart and Dumb components.

• **Smart Components:** Also called container components, they handle business logic, fetch data, communicate with services, and pass data to child components.
• **Dumb Components:** These are presentation components focused only on UI; they receive data via @Input() and emit events using @Output().

• Smart components have low reusability but control data flow, whereas dumb components are highly reusable.
• Smart components are not UI-focused, while dumb components primarily handle display.
• Best practice is to keep components small and let smart components manage logic while dumb components handle presentation.
• This approach improves scalability and maintainability.

Q4. What is FormArray in Angular Reactive Forms? Explain its use with an example scenario.

Answer:
FormArray is a special form control that stores an array of FormControls, FormGroups, or other FormArrays in reactive forms.

• It allows dynamic creation of form fields when the number of inputs is not known at compile time.
• Developers can add, remove, or insert controls dynamically based on user requirements.
• Common use cases include adding multiple addresses, contacts, tasks, or survey questions.
• Reactive forms define structure and validation in the component class, giving better control and scalability.
• Example: An order form where users can dynamically add multiple items.
• Thus, FormArray enables flexible and data-driven form design.

Q5. Explain the folder structure of a scalable Angular project created using Angular CLI.

Answer:
A well-organized folder structure is essential for scalability, maintainability, and faster development in large Angular projects.

• **.angular/** – Internal cache used by Angular CLI for faster builds.
• **node_modules/** – Contains installed npm packages and should not be modified manually.
• **public/** – Stores static assets like images and favicon files.
• **src/** – Main application source code, while **src/app/** contains core application logic.
• Feature folders such as **admin** or **login** organize components based on functionality.
• Important files include index.html (main page), main.ts (entry point), and configuration files like angular.json.
• This structure improves readability, supports lazy loading, and enables team collaboration.

Q6. What are Standalone Components in Angular 17?

Answer:
Standalone components are Angular components that operate independently without requiring an NgModule.

• They are declared using standalone: true in the component metadata.
• Dependencies can be imported directly inside the component, simplifying configuration.
• They reduce boilerplate code and make applications easier to manage.
• Standalone components support modern Angular architecture and faster development.
• They improve modularity and help create lightweight applications.
• Hence, they provide a simpler and more efficient way to build Angular apps.

Unit 2

Q1. What are Standalone Components in Angular 17? Explain their advantages.

Answer:
Standalone components are Angular components that function independently without requiring an NgModule, simplifying application architecture.

• They are declared using standalone: true in the component metadata, allowing direct usage.
• Dependencies such as directives and pipes can be imported directly inside the component.
• They reduce boilerplate code, making applications cleaner and easier to maintain.
• Standalone components improve modularity and support modern Angular development practices.
• They enable faster development because fewer configuration files are required.
• Applications become lightweight and easier to scale.
• Hence, standalone components enhance performance, readability, and developer productivity.

Q2. Explain Data Binding in Angular. Give types with examples.

Answer:
Data binding is the mechanism that connects the component (TypeScript) with the template (HTML), allowing automatic synchronization of data between the model and view.

• It reduces manual DOM manipulation and improves application efficiency.
• **Interpolation:** Uses {{ }} to display component data in the template. Example: <h1>{{title}}</h1>.
• **Property Binding:** Binds component data to element properties. Example: <img [src]="imageUrl">.
• **Event Binding:** Handles user actions such as clicks. Example: <button (click)="save()">.
• **Two-way Binding:** Combines property and event binding using [(ngModel)] to keep data synchronized.
• It improves code readability and enables dynamic UI updates.
• Therefore, data binding is essential for building interactive Angular applications.

Q3. What is State Management in Angular? Explain its need in large applications.

Answer:
State management refers to the process of managing and maintaining application data so that the UI updates automatically when data changes. Angular commonly uses RxJS Subjects and BehaviorSubjects for reactive state handling.

• Subjects act as both observable and observer, allowing values to be emitted to multiple subscribers.
• BehaviorSubjects store the latest value so new subscribers immediately receive current data.
• It centralizes application data, making it easier to share information between components.
• State management is important in large applications to avoid inconsistent data.
• It helps manage asynchronous operations such as login status or API responses.
• Improves scalability, maintainability, and debugging.
• Hence, it ensures predictable data flow across the application.

Q4. How does Angular handle forms? Briefly explain Template-driven and Reactive forms.

Answer:
Angular provides two main approaches to handle forms: Template-driven forms and Reactive forms.

• **Template-driven forms** rely mainly on HTML templates to define form structure and validation.
• They are simple to use and suitable for small applications with basic validation.
• **Reactive forms** are model-driven, where form structure, validation, and state are defined in the component class.
• Reactive forms provide better control, scalability, and testability for complex applications.
• They use FormControl, FormGroup, and FormArray along with Observables to manage form data.
• Reactive forms support dynamic fields, custom validators, and advanced state tracking.
• Therefore, template-driven forms are ideal for simple forms, whereas reactive forms are preferred for enterprise-level applications.

Q5. Give the differences between Template-driven forms and Reactive forms.

Answer:

Feature	**Template-driven Forms**	**Reactive Forms**
Approach	Template-based	Model-driven
Form Logic	Defined in HTML	Defined in component class

Feature	Template-driven Forms	Reactive Forms
Complexity	Suitable for simple forms	Ideal for complex forms
Validation	Limited and mostly template-based	Powerful with custom and async validators
Scalability	Less scalable	Highly scalable
Testing	Harder to test	Easier to test
Control	Less control over form state	Full control over form lifecycle
Use Case	Small applications	Large, real-world applications

Reactive forms provide better control, scalability, and testability compared to template-driven forms.

Q6. How are Angular forms connected to Express APIs for data submission and retrieval?

Answer:
Angular forms communicate with Express APIs using HTTP requests to send and retrieve data from the backend.

• When a form is submitted, Angular sends a POST request containing form data to the server endpoint.
• Express middleware such as express.json() parses incoming request bodies.
• The controller processes the data, performs validation, and returns a JSON response.
• Angular subscribes to the response to display success messages or handle errors.
• Proxy configuration can be used during development to avoid CORS issues.
• This integration enables full-stack communication between frontend and backend.
• Thus, Angular forms and Express APIs work together to support efficient data submission and retrieval.

Unit 3

Q1. What is Node.js? Explain its features and advantages.

Answer:
Node.js is a server-side JavaScript runtime environment built on the V8 engine that allows developers to execute JavaScript outside the browser. It uses an event-driven, non-blocking I/O model for scalable applications.

• It supports asynchronous programming, enabling multiple requests to be handled efficiently.
• Built on the V8 engine, it provides high performance and fast execution.
• Node.js is ideal for real-time applications such as chat apps and streaming services.

• It uses a single-threaded architecture, reducing server resource consumption.
• Large npm ecosystem provides reusable libraries and tools.
• Enables full-stack development using JavaScript on both client and server.
• Hence, Node.js simplifies development while improving scalability and performance.

Q2. Explain the role of Express.js in backend development.

Answer:
Express.js is a lightweight web framework built on top of Node.js that simplifies backend development by providing structured routing and middleware support.

• It helps create web servers and APIs quickly with minimal configuration.
• Provides routing features to handle HTTP requests such as GET, POST, PUT, and DELETE.
• Supports middleware for processing requests before sending responses.
• Makes request handling and response management easier compared to raw HTTP servers.
• Encourages organized project structure for scalable applications.
• Reduces development time by offering built-in utilities.
• Therefore, Express.js is widely used for building fast and efficient backend services.

Q3. How does middleware work in Express.js? Give a brief example.

Answer:
Middleware is a function that executes between receiving a request and sending a response in an Express application. It can modify request and response objects or pass control to the next function.

• Middleware flow follows: **Request → Middleware → Route Handler → Response.**
• Built-in middleware like express.json() parses JSON request bodies.
• Application-level middleware can log request details before processing.
• Router-level middleware works for specific routes only.
• Error-handling middleware captures and manages application errors.

Example:

```
app.use((req, res, next) => {

 console.log(`${req.method} ${req.url}`);

 next();

});
```

• This middleware logs request information and forwards control using next().

Q4. Explain RESTful APIs. What are the common HTTP methods used?

Answer:
RESTful APIs are web services that follow the Representational State Transfer (REST) architecture to enable communication between client and server using standard HTTP methods.

• They use structured URLs to perform operations on resources such as users or products.
• **GET:** Retrieves data from the server.
• **POST:** Creates new data or resources.
• **PUT:** Updates existing resources.
• **DELETE:** Removes resources from the server.
• REST APIs usually return responses in JSON format for easy data exchange.
• They support scalable and platform-independent communication.
• Hence, RESTful APIs are essential for modern web and mobile applications.

Q5. Explain the role of environment variables in Express.js applications.

Answer:
Environment variables are configuration values stored outside the application code to manage sensitive data and settings securely.

• They store information such as PORT, database URIs, and API keys.
• The .env file is commonly used along with the dotenv package to load variables.
• Example variables include PORT=3000 and NODE_ENV=development.
• Using environment variables improves security by preventing secrets from being hardcoded.
• Helps maintain different configurations for development and production environments.
• They are usually added to .gitignore to avoid exposure.
• Thus, environment variables make applications flexible, secure, and easier to manage.

Q6. Explain the role of Node.js in building server-side applications and mention any two advantages.

Answer:
Node.js plays a crucial role in building server-side applications by enabling developers to create fast, scalable network programs using JavaScript.

• It uses an event-driven, non-blocking I/O model that efficiently handles concurrent requests.
• Suitable for APIs, real-time apps, and data-intensive platforms.
• Allows the same language (JavaScript) for both frontend and backend development.
• **Advantage 1:** High performance due to the V8 engine.
• **Advantage 2:** Scalable architecture capable of handling many connections simultaneously.
• Reduces development complexity and improves productivity.

• Therefore, Node.js is a powerful choice for modern server-side development.

Q7. What is nodemon? Explain its purpose and advantages.

Answer:
Nodemon is a development tool that automatically restarts a Node.js application whenever file changes are detected, improving developer efficiency.

• Installed as a development dependency using npm install --save-dev nodemon.
• Eliminates the need to manually restart the server after every change.
• Watches files such as server.js, routes, and controllers for updates.
• Speeds up development and testing processes.
• Helps identify errors quickly during coding.
• Supports customizable configuration through nodemon.json.
• Hence, nodemon enhances productivity and streamlines backend development.

Unit 4

Q1. What is Firebase? Explain its main services used in web development and how collections and subcollections are structured.

Answer:
Firebase is Google's platform for building web and mobile applications with services such as real-time databases, authentication, hosting, and cloud functions.

• **Firestore:** A document-oriented NoSQL database with real-time synchronization.
• **Realtime Database:** An older JSON-based database still used for simple real-time apps.
• **Firebase Hosting:** CDN-based static hosting for fast and secure deployment.
• **Firebase Authentication:** Provides built-in user management and login support.
• Firestore organizes data in a hierarchy: **Collections → Documents → Fields**, allowing structured storage.
• Subcollections help organize related data (e.g., users/{userId}/posts) and improve query clarity.
• This structure ensures scalability, better data organization, and efficient retrieval.

Q2. Explain Firestore Database. How is it different from Realtime Database?

Answer:
Firestore is a flexible, scalable NoSQL database designed for modern app development with real-time data synchronization.

• It follows a **document-oriented model** with JSON-like data.
• Supports rich queries including filtering, sorting, and pagination.
• Offers better scalability for large datasets compared to Realtime Database.

Difference:

Feature	**Firestore**	**Realtime Database**
Data Model	Document-oriented	JSON structure
Querying	Rich query support	Limited queries
Pricing	Pay per read/write	Pay per GB stored
Scaling	Better for large datasets	Slower at massive scale
Use Case	Structured data apps	Messaging & simple CRUD

Firestore is generally preferred for structured and scalable applications.

Q3. How are CRUD operations performed in Firestore? Explain briefly.

Answer:
Firestore supports CRUD operations (Create, Read, Update, Delete) to manage application data efficiently.

• **Create:** Use addDoc() for auto-generated IDs or setDoc() for custom IDs when adding documents.
• **Read:** Retrieve documents using getDocs() or apply queries with filters.
• **Update:** Modify existing documents using updateDoc().
• **Delete:** Remove documents with deleteDoc() to maintain data consistency.
• Batch writes allow multiple operations in a single request for better performance.
• Data is stored inside collections such as "students," where each document represents one record.
• These operations enable efficient data handling in full-stack applications.

Q4. Explain how Firebase is integrated with Angular applications.

Answer:
Firebase can be integrated with Angular using the Firebase SDK and AngularFire to fetch and store data securely.

• Install dependencies using npm install firebase @angular/fire.
• Configure Firebase in **app.config.ts** using provideFirebaseApp() and initializeApp().
• Provide services like Firestore and Authentication using provideFirestore() and provideAuth().
• Create Angular services to perform operations such as adding and retrieving data.
• Authentication features like registration and login can be implemented with Angular services.

• Firebase Admin SDK can be connected with Express for secure backend access.
• This integration enables real-time data handling and supports full-stack development.

Q5. Give a brief explanation of Firebase Hosting and deployment process.

Answer:
Firebase Hosting is a fast and secure hosting service designed for deploying modern web applications, especially single-page apps.

• It uses a global CDN to deliver content quickly to users.
• Provides HTTPS by default for secure communication.
• Supports SPA routing, making it ideal for Angular projects.

Deployment Steps:
• Build the Angular project using ng build --configuration=production.
• Install Firebase CLI.
• Initialize Firebase in the project.
• Deploy using the command firebase deploy.

• Firebase Hosting simplifies deployment while ensuring speed, reliability, and scalability.

Q6. What is React? Explain in detail with advantages and disadvantages.

Answer:
React is a popular JavaScript library used for building interactive user interfaces through reusable components. It allows developers to create fast and scalable frontend applications.

• Uses **JSX**, a syntax that looks like HTML but compiles into JavaScript.
• Supports **functional components** that promote reusable UI design.
• Hooks like **useState** enable local state management within components.
• **useEffect** handles side effects such as API calls and subscriptions.
• Encourages component composition for modular architecture.

Advantages:
• High performance due to efficient rendering.
• Reusable components reduce development time.
• Strong community support and ecosystem.
• Suitable for large, scalable applications.

Disadvantages:
• Requires additional tools for routing and state management.
• JSX syntax may have a learning curve for beginners.
• Frequent updates can make maintenance challenging.

Thus, React is widely adopted for modern frontend development due to its flexibility and efficiency.