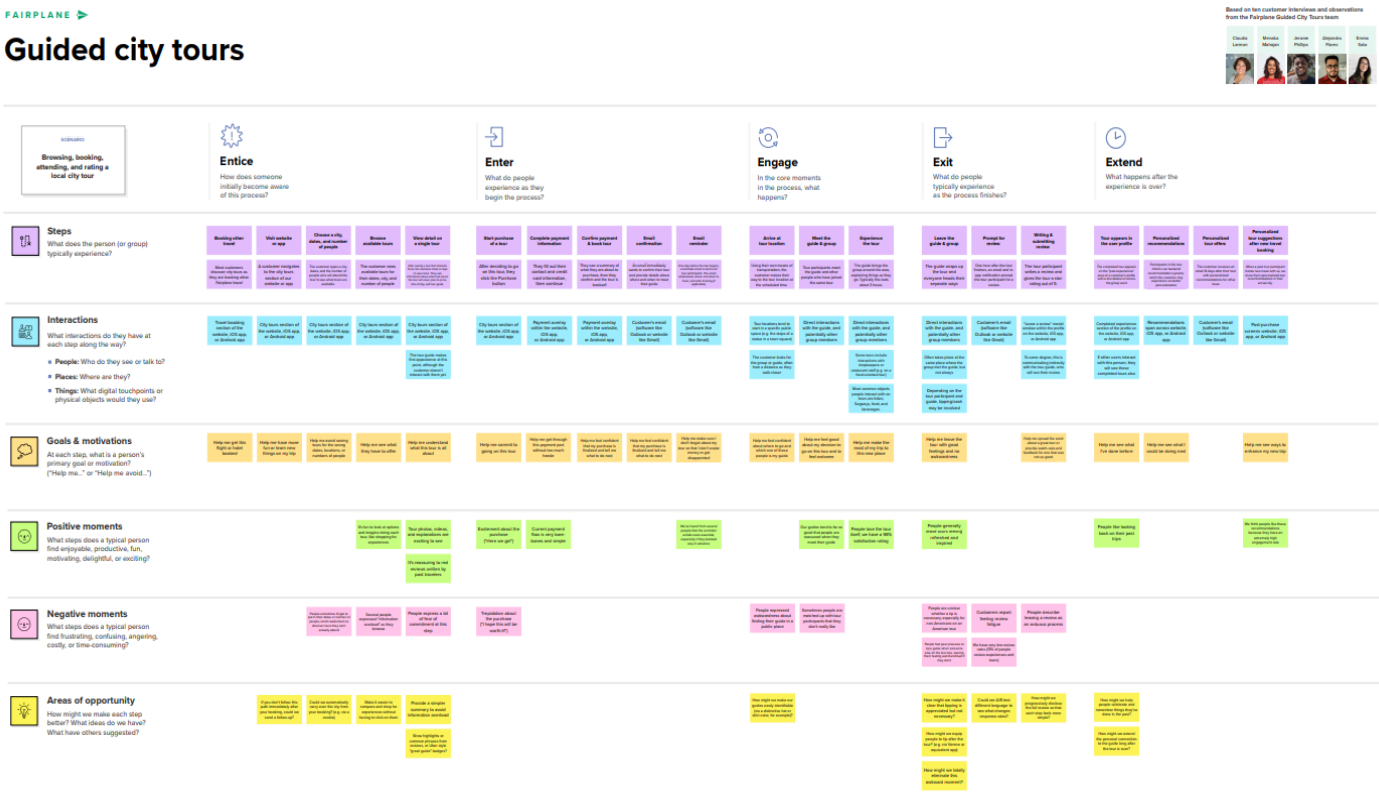



Guided city tours offer a unique and enriching way to explore a new city. These tours provide visitors with a deeper understanding of the city's history, culture, and landmarks, while also offering a convenient and stress-free way to navigate unfamiliar streets.

## Types of Guided City Tours:

1. **Walking Tours**: Explore the city on foot with a knowledgeable guide, discovering hidden gems and local favorites.
2. **Bus Tours**: See the sights from the comfort of a bus, often with hop-on hop-off options for added flexibility.
3. **Food Tours**: Sample local cuisine and drinks while learning about the city's culinary scene.
4. **Bike Tours**: Pedal through the city with a guide, taking in the sights and sounds at a leisurely pace.
5. **Private Tours**: Customize your tour experience with a private guide, tailored to your interests and schedule.

## Benefits of Guided City Tours:

1. Local Insights: Gain a deeper understanding of the city's history, culture, and customs from a knowledgeable guide.
2. Time-Saving: Let someone else handle the logistics, ensuring you make the most of your limited time in the city.
3. Safety: Explore the city with confidence, knowing you're in good hands with a reputable tour operator.
4. Meeting New People: Join a group tour and meet fellow travelers from around the world.

## Popular Guided City Tours:

1. New York City: Explore the Big Apple's iconic landmarks, from the Statue of Liberty to Central Park.

2. Paris: Discover the City of Light's famous art museums, historic landmarks, and charming neighborhoods.

3. Tokyo: Experience the vibrant energy of Japan capital city from neon- lit slickpeary to ancient temples.

4. Rome: Uncover the Eternal City's ancient ruins, Vatican City, and delicious Italian cuisine.

5. Barcelona: Stroll along La Rambla, visit the iconic Sagrada Familia, and enjoy the city's stunning beaches.

## Guided city tours can also offer various specialized experiences, such as:

1. Ghost Tours: Explore the city's spooky side with a guide, visiting haunted sites and hearing eerie tales.

2. Street Art Tours: Discover the city's vibrant street art scene, learning about the artists and their works.

3. Architecture Tours: Examine the city's iconic buildings and landmarks, learning about their history and design.

4. Food and Wine Tours: Sample local cuisine and drinks, learning about the city's culinary scene and wine production.

5. Nighttime Tours: Experience the city's nightlife, visiting bars, clubs, and live music venues.

## Many guided city tours also offer additional amenities and services, such as:

1. Hotel Pickups: Convenient transportation from your hotel to the tour starting point.

2. Audio Guides: Multilingual audio guides to help you understand the tour commentary.

3. Skip-the-Line Access: Priority entry to popular attractions, saving you time and hassle.

4. Small Group Sizes: Intimate tour groups, ensuring a personalized experience.

5. Expert Guides: Knowledgeable and passionate guides, providing insightful commentary and answering your questions.

## When choosing a guided city tour, consider the following factors:

1. Tour Length: Select a tour that fits your schedule and interests.

2. Tour Style: Choose a tour that suits your preferences, such as walking, bus, or bike.

3. Guide Quality: Opt for a tour with expert, knowledgeable, and enthusiastic guides.

4. Group Size: Consider a small group tour for a more personalized experience.

5. Reviews and Ratings: Research the tour operator's reputation and read reviews from past customers.

## Some popular guided city tour companies include:

1. Viator: Offers a wide range of guided tours in cities worldwide.

2. GetYourGuide: Provides guided tours, activities, and experiences in over 7,000 destinations.

3. Big Bus Tours: Operates hop-on hop-off bus tours in major cities worldwide.

4. City Wonders: Offers guided tours and experiences in Europe, Asia, and the Americas.

5. Fat Tire Tours: Provides guided bike tours in cities worldwide.

6. Tour Customization: Some guided city tour companies offer customization options, allowing you to tailor the tour to your interests and preferences.

# Project Design Phase-II
# Data Flow Diagram & User Stories

| | |
|---|---|
| Date | 31 January 2025 |
| Team ID | |
| Project Name | |
| Maximum Marks | 4 Marks |

**Data Flow Diagrams:**

**What is a Data Flow Diagram?**

A Data Flow Diagram (DFD) is a graphical representation of the flow of data through a system or process. It's a tool used to model and analyze the flow of data, highlighting the relationships between different components and the data that flows between them.

**Components of a Data Flow Diagram**:

1. Entities: External sources or destinations of data, such as users, databases, or other systems.

2. Processes: Transformations or operations performed on the data, such as calculations, sorting, or formatting.

3. Data Flows: The movement of data between entities, processes, and data stores.

4. Data Stores: Repositories of data, such as databases, files, or data warehouses.

**Types of Data Flow Diagrams:**

1. Context Diagram: A high-level DFD that shows the overall system and its interactions with external entities.

2. Level 1 DFD: A more detailed DFD that breaks down the system into its major components and processes.

3. Level 2 DFD: An even more detailed DFD that shows the specific data flows and processes within each component.

**Benefits of Data Flow Diagrams:**

1. Improved Communication: DFDs provide a clear and concise visual representation of the system, facilitating communication among stakeholders.

2. System Analysis: DFDs help identify inefficiencies, bottlenecks, and areas for improvement in the system.

3. System Design: DFDs provide a foundation for designing new systems or modifying existing ones.

Tools for Creating Data Flow Diagrams:

1. Lucidchart: A popular online diagramming tool that supports DFD creation.

2. (link unavailable): A free online diagramming tool that allows you to create DFDs.

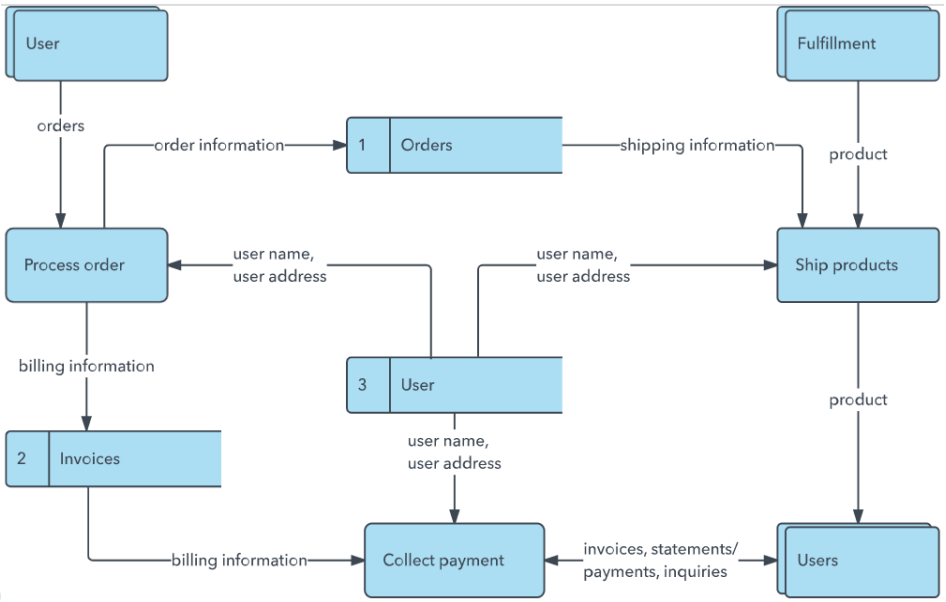
3. Microsoft Visio: A commercial diagramming tool that supports DFD creation.

**Best Practices for Creating Data Flow Diagrams:**

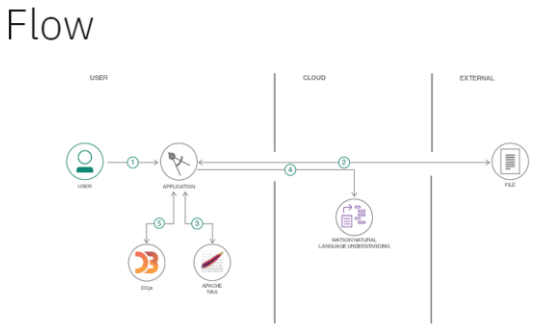
1. Keep it Simple: Avoid cluttering the diagram with too much detail.

2. Use Consistent Symbols: Stick to a standard set of symbols and notation.

3. Label Clearly: Use clear and concise labels for entities, processes, and data flows.




**Example: [(Simplified)]**




1. User configures credentials for the Watson Natural Language Understanding service and starts the app.
2. User selects data file to process and load.
3. Apache Tika extracts text from the data file.
4. Extracted text is passed to Watson NLU for enrichment.
5. Enriched data is visualized in the UI using the D3.js library.

**Data Flow Diagram Symbols:**

1. Entity: Represented by a rectangle, entities are external sources or destinations of data.

2. Process: Represented by a circle or oval, processes are transformations or operations performed on the data.

3. Data Flow: Represented by an arrow, data flows show the movement of data between entities, processes, and data stores.

4. Data Store: Represented by a open-ended rectangle, data stores are repositories of data.

**Data Flow Diagram Levels**:

1. Context Diagram: A high-level DFD that shows the overall system and its interactions with external entities.

2. Level 1 DFD: A more detailed DFD that breaks down the system into its major components and processes.

3. Level 2 DFD: An even more detailed DFD that shows the specific data flows and processes within each component.

4. Improved Communication: DFDs provide a clear and concise visual representation of the system.

5. System Analysis: DFDs help identify inefficiencies, bottlenecks, and areas for improvement.

**User Stories**

Use the below template to list all the user stories for the product.

| User Type | Functional Requirement (Epic) | User Story Number | User Story / Task | Acceptance criteria | Priority | Release |
|---|---|---|---|---|---|---|
| Customer (Mobile user) | Registration | USN-1 | As a user, I can register for the application by entering my email, password, and confirming my password. | I can access my account / dashboard | High | Sprint-1 |
| | | USN-2 | As a user, I will receive confirmation email once I have registered for the application | I can receive confirmation email & click confirm | High | Sprint-1 |
| | | USN-3 | As a user, I can register for the application through Facebook | I can register & access the dashboard with Facebook Login | Low | Sprint-2 |
| | | USN-4 | As a user, I can register for the application through Gmail | | Medium | Sprint-1 |
| | Login | USN-5 | As a user, I can log into the application by entering email & password | | High | Sprint-1 |
| | Dashboard | | | | | |
| Customer (Web user) | | | | | | |
| Customer Care Executive | | | | | | |
| Administrator | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |

**What is a User Story?**

A User Story is a natural-language description of a software feature or requirement from the perspective of the end-user. It's a way to capture the user's needs and goals in a concise and understandable format.

**Components of a User Story:**

1. As a [type of user]: Describes the user's role or persona.

2. I want to [perform some task]: Describes the user's goal or desire.

3. So that [some reason or benefit]: Describes the user's motivation or expected outcome.

Example of a User Story:

"As a customer, I want to be able to view my order history, so that I can easily track my purchases and reorder items."

**Benefits of User Stories:**

1. Improved Communication: User stories help developers understand the user's needs and goals.

2. Increased Empathy: User stories encourage developers to think from the user's perspective.

**Project Design Phase-II**
**Solution Requirements (Functional & Non-functional)**

| | |
|---|---|
| Date | 31 January 2025 |
| Team ID | |
| Project Name | |
| Maximum Marks | 4 Marks |

**Functional Requirements:**

## What are Functional Requirements?

Functional Requirements (FRs) are detailed descriptions of the functions that a software system or application must perform. They define what the system must do, how it must behave, and what features it must have.

## Types of Functional Requirements:

1. Business Requirements: Define the business needs and goals that the system must support.

2. User Requirements: Define the needs and expectations of the system's users.

3. System Requirements: Define the technical requirements of the system, such as performance, security, and scalability.

## Characteristics of Good Functional Requirements:

1. Specific: Clearly and concisely define what the system must do.

2. Measurable: Quantifiable, so progress and success can be measured.

3. Achievable: Realistic and attainable, given the resources and constraints.

4. Relevant: Align with the business goals and user needs.

5. Time-bound: Define when the requirement must be met.

## Examples of Functional Requirements:

1. "The system must allow users to log in using their username and password."

2. "The system must generate a report of all sales transactions within the last 30 days."

3. "The system must send an email notification to the administrator when a new user account is created."

## Tools for Managing Functional Requirements:

1. Requirements Management Tools: Such as IBM Rational DOORS, or Jama Connect.

2. Agile Project Management Tools: Such as Jira, or Trello.

3. Spreadsheets: Such as Microsoft Excel, or Google Sheets.

Best Practices for Writing Functional Requirements

**Functional Requirements:**

Following are the functional requirements of the proposed solution.

| FR No. | Functional Requirement (Epic) | Sub Requirement (Story / Sub-Task) |
|---|---|---|
| FR-1 | User Registration | Registration through Form<br>Registration through Gmail<br>Registration through LinkedIN |
| FR-2 | User Confirmation | Confirmation via Email<br>Confirmation via OTP |
| FR-3 | | |
| FR-4 | | |
| | | |
| | | |

Characteristics of Good Functional Requirements |

| --- |

| Specific: Clearly and concisely define what the system must do. |

| Measurable: Quantifiable, so progress and success can be measured. |

| Achievable: Realistic and attainable, given the resources and constraints. |

| Relevant: Align with the business goals and user needs. |

| Time-bound: Define when the requirement must be met. |

| Characteristics of Good Functional Requirements |

| --- |

| Unambiguous: Clearly defined and easy to understand, with no room for misinterpretation. |

| Consistent: Align with other requirements and do not conflict with them. |

| Complete: Include all necessary details and do not omit important information. |

| Verifiable: Can be verified through testing, inspection, or other means. |

| Traceable: Can be traced back to the original source or requirement. |

| Prioritized: Prioritized based on business value, user needs, and technical feasibility. |

Characteristics of Good Functional Requirements

| Specific | Clearly and concisely define what the system must do. |

| Measurable | Quantifiable, so progress and success can be measured. |

| Achievable | Realistic and attainable, given the resources and constraints. |

**Non-functional Requirements:**

Following are the non-functional requirements of the proposed solution.

| FR No. | Non-Functional Requirement | Description |
|---|---|---|
| NFR-1 | **Usability** | |
| NFR-2 | **Security** | |
| NFR-3 | **Reliability** | |
| NFR-4 | **Performance** | |
| NFR-5 | **Availability** | |
| NFR-6 | **Scalability** | |

Non-Functional Requirements (NFRs) are constraints or qualities that a system or application must possess, beyond its functional requirements. NFRs describe how the system should behave, rather than what it should do. Examples of NFRs include performance requirements (e.g., response time, throughput), security requirements (e.g., authentication, authorization), usability requirements (e.g., user interface, accessibility), and scalability requirements (e.g., handling increased traffic or data). NFRs are crucial in ensuring that the system meets the desired quality attributes, is reliable, and provides a good user experience.

**Types of Non-Functional Requirements:**

1. **Performance Requirements**: Define the system's speed, efficiency, and responsiveness, such as response time, throughput, and latency.

2. **Security Requirements**: Define the system's security features, such as authentication, authorization, encryption, and access control.

3. **Usability Requirements:** Define the system's user interface, accessibility, and user experience, such as navigation, search functionality, and error handling.

4. **Scalability Requirements**: Define the system's ability to handle increased traffic, data, or user growth, such as horizontal scaling, load balancing, and caching.

5. **Reliability Requirements**: Define the system's availability, fault tolerance, and recovery capabilities, such as uptime, downtime, and mean time between failures.

6. **Maintainability Requirements**: Define the system's ease of maintenance, modification, and repair, such as code quality, testing, and documentation.

7**. Portability Requirements**: Define the system's ability to operate on different platforms, devices, and environments, such as operating systems, browsers, and hardware.

**Best Practices for Non-Functional Requirements**:

1. Prioritize NFRs: Prioritize NFRs based on business value, user needs, and technical feasibility.

2. Make NFRs Measurable: Make NFRs measurable by defining quantifiable metrics and thresholds.

3. Include NFRs in Testing: Include NFRs in testing to ensure that the system meets the desired quality attributes.

**Project Design Phase-II**
**Technology Stack (Architecture & Stack)**

| | |
|---|---|
| Date | 31 January 3035 |
| Team ID | |
| Project Name | |
| Maximum Marks | 4 Marks |

**Technical Architecture:**

A Technical Architecture (TA) is a detailed, high-level description of the overall architecture of a system or application, focusing on the technical aspects of its design and implementation. It provides a comprehensive framework for understanding the system's components, interactions, and infrastructure, including hardware, software, networks, and data storage. The TA defines the technical standards, guidelines, and best practices for the system's development, deployment, and maintenance, ensuring that it meets the required functional and non-functional requirements. It also provides a roadmap for the system's evolution and scalability, enabling organizations to make informed decisions about technology investments and resource allocation. By documenting the technical architecture, organizations can ensure consistency, reduce complexity, and improve communication among stakeholders, ultimately leading to a more efficient, effective, and sustainable system.

**Example: Order processing during pandemics for offline mode**

**Reference:** [https://developer.ibm.com/patterns/ai-powered-backend-system-for-order-processing-during-pandemics/](https://developer.ibm.com/patterns/ai-powered-backend-system-for-order-processing-during-pandemics/)

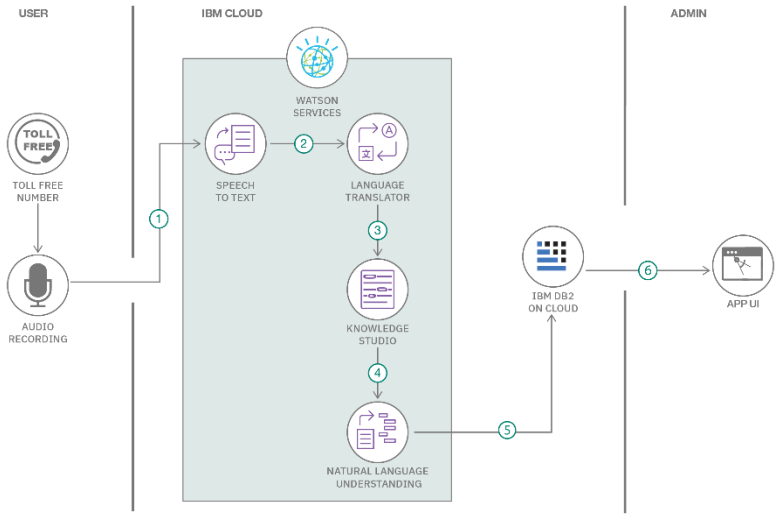



**Table-1 : Components & Technologies:**

| S.No | Component | Description | Technology |
|---|---|---|---|
| 1. | User Interface | How user interacts with application e.g. Web UI, Mobile App, Chatbot etc. | HTML, CSS, JavaScript / Angular Js / React Js etc. |
| 2. | Application Logic-1 | Logic for a process in the application | Java / Python |

| | | | |
|---|---|---|---|
| 3. | Application Logic-2 | Logic for a process in the application | IBM Watson STT service |
| 4. | Application Logic-3 | Logic for a process in the application | IBM Watson Assistant |
| 5. | Database | Data Type, Configurations etc. | MySQL, NoSQL, etc. |
| 6. | Cloud Database | Database Service on Cloud | IBM DB2, IBM Cloudant etc. |
| 7. | File Storage | File storage requirements | IBM Block Storage or Other Storage Service or Local Filesystem |
| 8. | External API-1 | Purpose of External API used in the application | IBM Weather API, etc. |
| 9. | External API-2 | Purpose of External API used in the application | Aadhar API, etc. |
| 10. | Machine Learning Model | Purpose of Machine Learning Model | Object Recognition Model, etc. |
| 11. | Infrastructure (Server / Cloud) | Application Deployment on Local System / Cloud<br>Local Server Configuration:<br>Cloud Server Configuration : | Local, Cloud Foundry, Kubernetes, etc. |

The Components & Technologies layer of a Technical Architecture refers to the specific software and hardware components that make up the system, as well as the technologies used to integrate them. This layer includes components such as web servers, application servers, databases, and messaging systems, as well as technologies like Java, .NET, and Python. The choice of components and technologies has a significant impact on the system's performance, scalability, and maintainability, and must be carefully evaluated to ensure alignment with the system's functional and non-functional requirements. Additionally, the Components & Technologies layer must also consider factors such as vendor support, licensing costs, and compatibility with existing systems and infrastructure. By carefully selecting and integrating the right components and technologies, organizations can build a robust, efficient, and adaptable system that meets their business needs and supports their long-term goals.

The Components & Technologies layer is responsible for providing the necessary functionality to support the system's business processes and user interactions. This includes components such as user interface (UI) frameworks, business logic frameworks, and data access frameworks, as well as technologies like service-oriented architecture (SOA), microservices, and cloud computing. The choice of components and technologies must be based on a thorough evaluation of the system's requirements, as well as the organization's technical strategy and direction. Additionally, the Components & Technologies layer must also consider factors such as integration with existing systems, scalability, and maintainability, as well as the skills and expertise of the development team. By selecting the right components and technologies, organizations can build a system that is efficient, effective, and adaptable to changing business needs.

Some common components and technologies used in modern systems include:

- Front-end frameworks like React, Angular, and Vue.js

- Back-end frameworks like Node.js, Django, and Ruby on Rails

- Databases like relational databases (RDBMS), NoSQL databases, and graph databases

- Messaging systems like Apache Kafka, RabbitMQ, and Amazon SQS

- Cloud platforms like Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP)

1. Modularity and Reusability: Components and technologies should be designed to be modular and reusable, allowing for greater flexibility and efficiency in system development and maintenance.

**Table-2: Application Characteristics:**

| S.No | Characteristics | Description | Technology |
|---|---|---|---|
| 1. | Open-Source Frameworks | List the open-source frameworks used | Technology of Opensource framework |
| 2. | Security Implementations | List all the security / access controls implemented, use of firewalls etc. | e.g. SHA-256, Encryptions, IAM Controls, OWASP etc. |
| 3. | Scalable Architecture | Justify the scalability of architecture (3 – tier, Micro-services) | Technology used |
| 4. | Availability | Justify the availability of application (e.g. use of load balancers, distributed servers etc.) | Technology used |
| 5. | Performance | Design consideration for the performance of the application (number of requests per sec, use of Cache, use of CDN's) etc. | Technology used |

**References:**


**https://c4model.com/**

**https://developer.ibm.com/patterns/online-order-processing-system-during-pandemic/**

**https://www.ibm.com/cloud/architecture**

**https://aws.amazon.com/architecture**

**https://medium.com/the-internal-startup/how-to-draw-useful-technical-architecture-diagrams-2d20c9fda90d**


**Reference:**


- "Software Architecture: Patterns, Principles, and Practices" by Mark Richards

- "Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions" by Gregor Hohpe and Bobby Woolf

- "Cloud Computing: Concepts, Technology & Architecture" by Thomas Erl


**Conclusion:**

In conclusion, the Components & Technologies layer is a critical aspect of a system's technical architecture. By carefully selecting and integrating the right components and technologies, organizations can build a robust, efficient, and adaptable system that meets their business needs and supports their long-term goals. It is essential to consider factors such as modularity, reusability, scalability, and performance when designing and implementing the Components & Technologies layer. By doing so, organizations can ensure that their system is able to evolve and adapt to changing business needs, while also providing a solid foundation for future growth and innovation.

Additionally, staying up-to-date with emerging trends and technologies, such as serverless computing, artificial intelligence, and the Internet of Things, can help organizations remain competitive and innovative in their respective markets. By leveraging these technologies and following best practices for component design and technology selection, organizations can build systems that are not only efficient and effective but also scalable, secure, and sustainable.

In conclusion, the Components & Technologies layer is critical to a system's technical architecture. Careful selection and integration of components and technologies can build a robust, efficient, and adaptable system. Considering modularity, reusability, scalability, and performance is essential when designing and