|  |
| --- |
| Assignment 1 |
|  |
| |  |  | | --- | --- | | **Submitted to: Kunwar Taimur** | | |  | | | **Submitted by: Muhammad Usman** | | |  | | | **Class: BS SE (EVE)** | | |  | | | **Roll no: 2521340** | | |  | | |  |  | |  |  | |

|  |
| --- |
|  |



|  |  |
| --- | --- |
|  | **GOVERNMENT COLLEGE UNIVERSITY FAISALABAD** |

|  |  |
| --- | --- |
| **Description: Gc uni logo.jpg** | **Department of Software Engineering GCUF** |

**Assignment 5: Case Study: GCUF Website System’s**

**Organization Architecture**

**Read the following case study of a security system and answer the questions**

**below. You can make necessary assumptions. But state them clearly.**

**Description:**

Draw an architecture diagram to show the organization of;

1. GCUF website system.

2. Then suggest a better architecture for this website. How can it be

improved if we have issues of managing lots of data, 24/7 availability and

space.

3. Suggest another design which should accommodate/cater the possibility of

“change in language of website from English to Urdu.”

**Question 1: GCUF website system?**

**Answer:**

The architecture of the Government College University Faisalabad (GCUF) website system can be illustrated as follows:

**Components:**

**Frontend:** Includes the user interface accessible via web browsers. It encompasses HTML, CSS, JavaScript, and frontend frameworks like React, Angular, or Vue.js.

**Backend:** Comprises the server, application logic, and database. It can involve technologies such as Node.js, Express.js, and a database like MongoDB or MySQL.

**Database:** Stores website data, including user information, academic details, news, and other relevant content.

**Server:** Hosts the website and handles requests from users. It communicates with the backend, retrieves data from the database, and sends responses to users.

**Connections:**

**Frontend-Backend Interaction:** Utilizes RESTful APIs or GraphQL for communication between the frontend and backend.

**Backend-Database Connection:** Uses appropriate drivers or ORMs (Object-Relational Mapping) to manage data operations.

**Hosting/Deployment:**

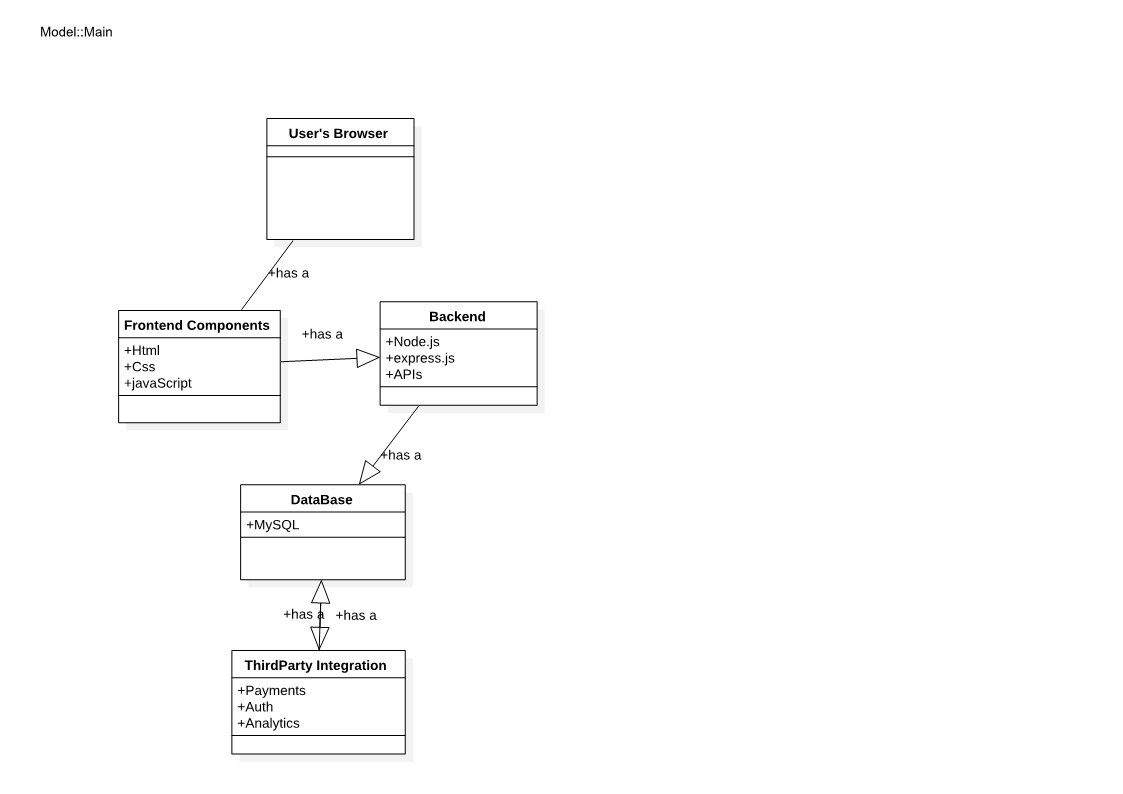
The system may be hosted on cloud services like AWS, Azure, or on-premises servers.

For better scalability, load balancing, and redundancy, multiple servers might be employed.

**Security Measures:**

SSL/TLS protocols for secure data transmission.

Firewalls, encryption, and access controls to safeguard the system.

****

**Question 2: Then suggest a better architecture for this website. How can it be improved if we have issues of managing lots of data, 24/7 availability and space?**

**Answer:** When considering the improvement of the website's architecture to handle issues of managing extensive data, ensuring 24/7 availability, and optimizing space, it's imperative to delve deeper into Software Design and Architecture (SDaA) principles. Here's an expanded and more detailed breakdown:

Architecture for Managing Lots of Data:

1. Data Partitioning Strategies:
   * Design Principle: Implement horizontal partitioning to distribute data across multiple servers based on criteria (e.g., date ranges, geographical location) to improve query performance and scalability.
   * Implementation: Use consistent hashing or range-based sharding to partition data effectively.
2. Big Data Technologies:
   * Design Principle: Utilize technologies designed for handling large datasets efficiently.
   * Implementation: Introduce Big Data solutions such as Hadoop, Spark, or NoSQL databases to process, analyze, and manage extensive data volumes effectively.
3. Optimized Indexing and Query Optimization:
   * Design Principle: Enhance database performance through efficient indexing strategies and optimized queries.
   * Implementation: Utilize composite indexes, covering indexes, and query optimization techniques to improve data retrieval speed and efficiency.

Architecture for 24/7 Availability:

1. Geographical Redundancy and Disaster Recovery:
   * Design Principle: Ensure data redundancy across multiple geographical regions for disaster recovery and high availability.
   * Implementation: Implement multi-region deployments on cloud platforms and establish failover mechanisms to ensure continuous service availability.
2. Auto-Scaling and Elasticity:
   * Design Principle: Enable the system to scale resources dynamically based on demand.
   * Implementation: Use cloud services with auto-scaling features to automatically adjust resources to handle fluctuating traffic while maintaining optimal performance.
3. Microservices and Service Mesh:
   * Design Principle: Break down the system into microservices for improved resilience and fault tolerance.
   * Implementation: Use service mesh architecture to manage communication between microservices, allowing better control and visibility into inter-service communication.

Architecture for Optimizing Space:

1. Data Archiving and Lifecycle Management:
   * Design Principle: Implement data archiving and lifecycle management policies.
   * Implementation: Archive infrequently accessed or historical data to cheaper and slower storage solutions while keeping frequently accessed data in faster storage.
2. Content Distribution and Edge Computing:
   * Design Principle: Utilize edge computing to reduce latency and optimize content delivery.
   * Implementation: Employ edge computing techniques along with CDNs to cache and serve content closer to end-users, minimizing server load and optimizing space.

**Question 3: Suggest another design which should accommodate/cater the possibility of “change in language of website from English to Urdu.”?**

**Answer:** When designing a system to accommodate a change in language from English to Urdu in the context of Software Design and Architecture (SDaA), it involves considering several architectural and design principles. Here's a detailed breakdown focusing on SDaA concepts:

**Internationalization (I18n) and Localization (L10n):**

**Separation of Language-Specific Content:**

**Design Principle:** Apply the principle of separation of concerns. Separate language-specific content (text, labels, messages) from the codebase into resource files or databases.

**Implementation:** Utilize resource files, databases, or content management systems to store and manage language-specific content separately from the application logic.

**Support for Multiple Languages:**

**Design Principle:** Embrace open-closed principle. Design the system to be open for extension but closed for modification.

**Implementation:** Create language-specific modules or components that can be easily extended to support new languages without modifying existing code extensively.

**Design Patterns and Strategies:**

**Factory Method Pattern:**

**Design Principle:** Apply design patterns to allow flexibility and easy addition of new languages.

Implementation: Implement the Factory Method pattern to create language-specific factories responsible for generating language-specific content.

**Decorator Pattern:**

**Design Principle:** Use patterns that facilitate adding functionalities without altering existing code.

**Implementation:** Apply the Decorator pattern to dynamically add language-specific decorators to components without modifying their core functionalities.

**System Architecture and Components:**

**Language-Neutral UI Components:**

**Design Principle:** Encapsulate language-specific elements separately from core UI components.

**Implementation:** Develop UI components that can dynamically fetch language-specific content based on user preferences or system settings.

**Language Selection Mechanism:**

**Design Principle:** Provide user-friendly language selection features without compromising user experience.

**Implementation:** Implement a language switcher component or setting accessible across the website to allow users to choose their preferred language easily.

**Testing and Validation:**

**Localization Testing:**

**Design Principle:** Incorporate testing strategies for language-specific functionalities.

**Implementation:** Conduct rigorous localization testing to ensure accurate translation, proper rendering, and alignment of text and UI elements in Urdu without impacting the functionality.

**User Feedback and Iterative Improvement:**

**Design Principle:** Follow agile principles for continuous improvement based on user feedback.

**Implementation:** Collect user feedback on language preferences and usability in Urdu. Use agile methodologies to iteratively improve the language switch feature and ensure a seamless transition between English and Urdu.