Assignment 3

Nina Hartmann,

Clara Tschamon

Functional Requirements

- *Quality Scenario*
  - *Source*
  - *Stimulus*
  - *Artifact*
  - *Environment*
  - *Response*
  - *Response Measure*
- *Tactics*

1) The system must be scalable, able to handle increasing amounts of data and traffic.

Quality Scenario

Source: The increasing number of users and data input to the system.

Stimulus: The system receives a sudden spike in traffic, exceeding its capacity to handle it.

Artifact: The processors, communication channels, storage, and processes involved in handling the increased traffic.

Environment: Normal operation during high traffic periods.

Response: The system must prevent the fault from becoming a failure by scaling up its resources to handle the increased traffic. It must also detect any faults or performance issues and recover from them quickly to prevent downtime or data loss.

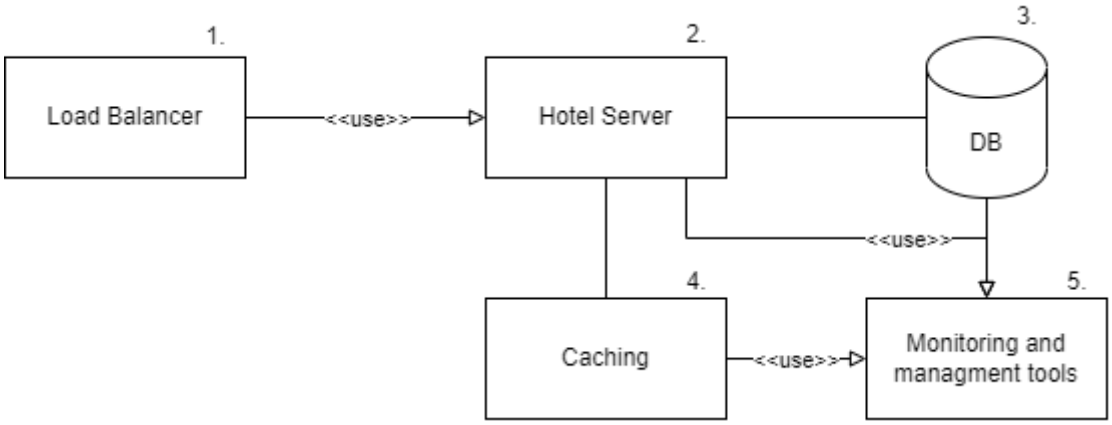
Response Measure: The system must maintain at least 99.9% availability during high traffic periods, and any faults or issues must be detected and resolved within 5 minutes.

Tactics

To ensure system scalability, the following actions should be taken:

1. Design the system to be modular and easily scalable by adding additional resources such as processors, storage, or network bandwidth.
2. Implement load balancing and automatic scaling to distribute traffic across multiple instances of the system and to scale up or down based on demand.
3. Use monitoring tools to detect any faults or performance issues and alert system administrators for quick resolution.
4. Conduct stress testing to simulate high traffic loads and ensure the system can handle them without failures.

By implementing these measures, the system can handle increasing traffic and data without causing downtime or failures, providing a reliable and scalable service to its users.



Here you can see a short ULM.

1. Load balancer: This component is responsible for distributing incoming requests from customers across multiple hotel servers. As the number of customers increases, the load balancer can adjust the distribution of requests to optimize performance and minimize downtime.
2. Hotel servers: These components represent the physical servers that run the hotel management software. As the number of customers increases, more servers can be added to handle the additional load.
3. Database: This component stores all the data related to the hotel management software, including customer information, reservations, and room availability. The database is designed to be scalable and can be partitioned or sharded to improve performance.
4. Caching: This component stores frequently accessed data in memory to reduce the load on the database and improve system performance.
5. Monitoring and management tools: These components provide real-time monitoring of system performance and capacity and allow administrators to adjust system resources and settings as needed.

2) The system must be maintainable, with easy updates and bug fixes.

Quality Scenario

Source: Stakeholder (such as software engineers, project manager, or business owner) responsible for maintaining the system.

Stimulus: The need to make updates or bug fixes to the system due to changes in the business requirements, security patches, or technical updates.

Artifact: The system's codebase, including the source code, libraries, and dependencies, as well as the system's documentation, such as technical specifications, user guides, and API documentation.

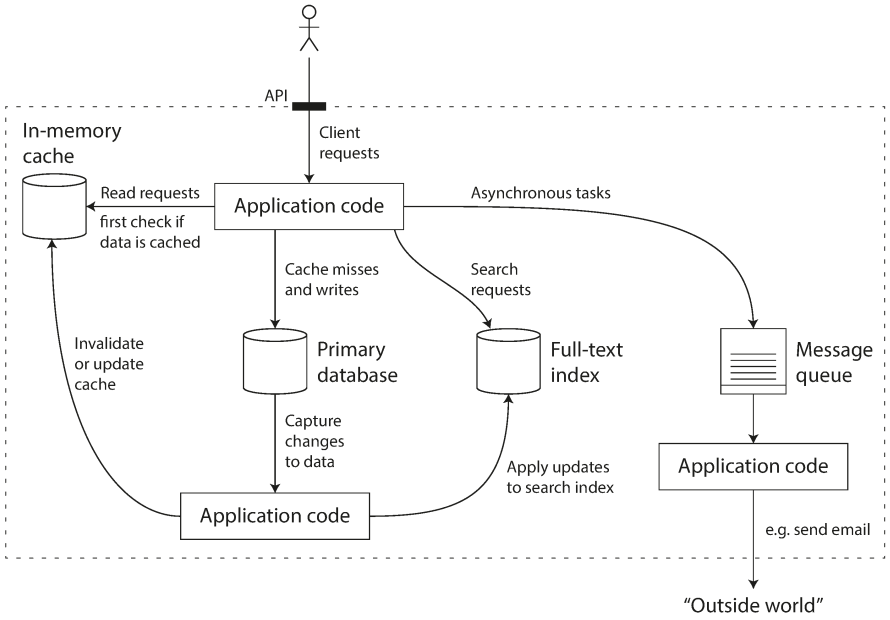
Environment: The development environment in which the maintenance is performed, including the tools and resources available.

Response: The system must be designed and implemented in a way that allows for easy updates and bug fixes. This may include modular architecture, clear and consistent coding standards, version control, and comprehensive documentation.

Response Measure: The maintainability of the system can be measured by factors such as the time it takes to make updates or bug fixes, the frequency and severity of bugs or errors, and feedback from the stakeholders responsible for maintaining the system.

Tactics

1. Modularity: The system should be designed with a modular architecture, where components can be easily added, removed, or modified without affecting other parts of the system. This makes it easier to isolate and fix bugs or make updates to specific features.
2. Code consistency and readability: The system should follow clear and consistent coding standards to make it easier to read and understand. This can help developers quickly identify and fix issues.
3. Version control: The system should use version control tools to manage changes to the codebase. This allows developers to roll back to a previous version if an update causes issues and helps track changes over time.
4. Comprehensive documentation: The system should have comprehensive documentation, including technical specifications, user guides, and API documentation. This makes it easier for developers and other stakeholders to understand the system and make updates or bug fixes.



HTTPS://WWW.OREILLY.COM/LIBRARY/VIEW/DESIGNING-DATA-INTENSIVE-APPLICATIONS/9781491903063/CH01.HTML

3) The system must be performant, with fast response times and minimal latency.

Quality Scenario

Source: End users and stakeholders who require a responsive and fast system.

Stimulus: The need to perform a specific action or task in the system, which requires the system to respond quickly and with minimal latency. This could be anything from loading a web page to executing a complex query.

Artifact: The system's architecture, including the hardware and software components, as well as the specific algorithms and data structures used to execute tasks and store data.

Environment: The environment in which the system is deployed, including factors such as the network infrastructure, available resources such as processing power and memory, and the number of concurrent users or requests.

Response: The system must be designed and implemented in a way that ensures fast response times and minimal latency. This may include optimizing the system's algorithms and data structures, using efficient network protocols, and caching mechanisms, and implementing load balancing and other performance optimization techniques.

Response Measure: The performance of the system can be measured by factors such as response time, latency, and throughput. These metrics can be benchmarked against industry standards or previous performance metrics to ensure that the system is meeting the required level of performance.

Tactics

1. Optimization: The system's algorithms and data structures should be optimized for performance to ensure that tasks are executed quickly and efficiently.
2. Network optimization: The system should use efficient network protocols and caching mechanisms to minimize latency and improve response times.
3. Load balancing: The system should use load balancing techniques to distribute requests and ensure that resources are used efficiently, even during peak usage periods.
4. Resource allocation: The system should be designed to make efficient use of available resources, such as processing power and memory, to ensure that tasks are executed quickly and without delays.

4) The system must be interoperable, able to communicate with other systems and software.

Quality Scenario

Source: The software development team responsible for designing and building the system.

Stimulus: The need for the system to be able to communicate effectively with other systems and software. This could be driven by several factors, such as the need to integrate with existing systems, or the desire to create a more seamless user experience by allowing data to be shared between different applications.

Artifact: The system under development, including any hardware, software, and network components. This might include APIs, communication protocols, data formats, and other technical specifications that enable the system to interact with other systems and software.

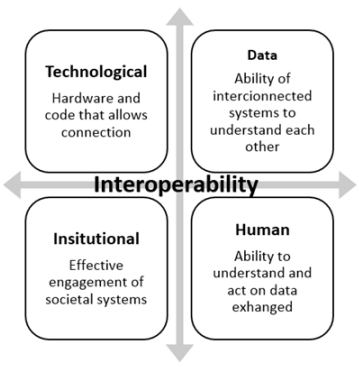
Environment: The development environment in which the software development team is working. This might include the hardware and software tools used to build and test the system, as well as any third-party systems or software that the system needs to integrate with.

Response: The system is designed and implemented to ensure interoperability and effective communication with other systems and software.

Response Measure: The software development team will conduct interoperability and communication testing to ensure that the system can successfully interact with other systems and software. They will use standard communication protocols and interfaces to facilitate integration with other systems. The team will also collaborate with other development teams to identify and address any interoperability issues.

Tactics:

1. Use industry-standard communication protocols and interfaces to facilitate integration with other systems and software.
2. Conduct thorough interoperability and communication testing to identify and address any issues.
3. Collaborate with other development teams to ensure compatibility and effective communication between systems.
4. Implement clear documentation and guidelines for integration with other systems.



HTTPS://ITLAW.FANDOM.COM/WIKI/INTEROPERABILITY

5) The system must be transparent, providing clear visibility into all processes and transactions.

Quality Scenario

Source: Software development team responsible for designing and building the system.

Stimulus: The need for the system to be transparent, providing clear visibility into all processes and transactions. This might be driven by factors such as regulatory compliance, a desire for greater accountability and oversight, or a need to improve user trust and confidence in the system.

Artifact: The system under development, including any hardware, software, and network components. This might include features such as audit trails, transaction logs, and other mechanisms that allow users to track and monitor the system's activity.

Environment: The operational environment in which the system will be used. This might include the hardware and software tools used to deploy and operate the system, as well as any third-party systems or software that the system needs to interact with.

Response: The software development team designs and implements the system in a way that provides clear visibility into all processes and transactions. This might involve implementing features such as audit trails, transaction logs, and other mechanisms that allow users to track and monitor the system's activity. The team will also ensure that the system is easy to use and understand, so that users can quickly and easily access the information they need.

Response Measure: The success of the system's transparency will be measured by how well it is able to provide clear visibility into all processes and transactions. The software development team will conduct testing to ensure that the system is able to capture and store all relevant data and will work closely with users to identify any areas where greater transparency is needed.

Tactics

1. Implement features such as audit trails and transaction logs to provide clear visibility into all processes and transactions.
2. Use clear and consistent terminology and labelling to ensure that users can understand the information provided by the system.
3. Provide user-friendly interfaces that allow users to access the information quickly and easily they need.
4. Conduct testing to ensure that the system can capture and store all relevant data.
5. Work closely with users to identify any areas where greater transparency is needed and implement changes to address these issues.

ReadMe

Coding Guidelines

Follow 1 Edit

Johannes RIEDMANN 9. März

1. Git

1.1 Branches

Es gibt pro User Story einen eigenen Branch, der nach dem Schema `[US_Nr.]-[US_Name]` benannt wird. Alle Teammitglieder, die an dieser User Story arbeiten, committen ihre fertigen Features auf diesen Branch.

Wenn unfertige Code-Schnipsel zwischengespeichert werden sollen, bitte vom User Story Branch wegbranchen und darauf die unfertigen Änderungen committen.
Namensgebung für diesen Branch: `[US_Nr.]-[Vorname]`

2. Variablenbenennung

2.1 Case-Definitionen:

Camel Case: `firstName, name, myFirstPhone`
Pascal Case: `FirstName, Name, MyFirstPhone`
Snake Case: `first_name, name, my_first_phone`
Kebab Case: `first-name, name, my-first-phone`
Caps: `FIRST_NAME, NAME, MY_FIRST_PHONE`

2.2 Generelle Variablenbenennung:

Variable type	Example	Case information
local variable	`myVariable`	camelCase
local *final* variable	`MY_CONSTANT`	CAPS
private attribute	`_myAttribute`	camelCase mit Unterstrich-Präfix
public attribute	`myAttribute`	camelCase
public *static* variable	`MY_ATTRIBUTE`	CAPS
class name	`MyClass`	PascalCase
SQL relation/attribute	`first_name`	snake_case*

**SQL macht keinen Unterschied zwischen Groß-/Kleinschreibung, daher bitte Unterstriche zwischen Wörtern verwenden!*

2.3 JavaFX:

FXML-Komponenten, welche im ViewController verwendet werden, bekommen generell IDs im *camelCase* (ohne Unterstrich-Präfix). Da es jedoch oft mehrere UI-Komponenten mit demselben Namen gibt (z.B. fristName als Label und als TextField), legen wir fest, dass hinter den deskriptiven Namen der FXML-Komponente immer mit einem verbindenden Unterstrich der Klassenname der Komponente im *PascalCase* angehängt wird.

Struktur: `[desctriptiveName]_[ClassName]`
Beispiele: `firstName_TextField`, `customer_TableView`, `myFirstComponent_Label`

6) The system must be fault-tolerant, able to recover from failures and errors.

Quality Scenario

Scenario: Fault-tolerant System Design for Recovery from Failures and Errors

Source: System faults and errors.

Stimulus: The system experiences a failure or error.

Artifact: The components of the system, including hardware, software, power sources and communication channels.

Environment: Normal operation.

Response: The system should be designed with fault-tolerant measures to recover from failures and errors. These measures may include redundant components, automated failover, and data backups.

Response Measure: The system should be able to recover from failures and errors within a specified time frame, and data integrity must be maintained throughout the process.

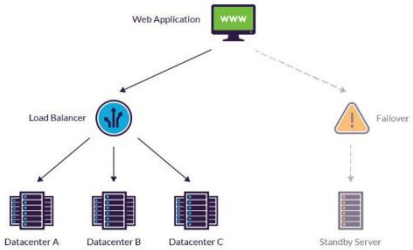
Software Tactics

1. Code Reviews
2. Testing
3. Monitoring (of tracking and analysing the performance and behaviour of a system. Monitoring tools, such as log analysers, can help identify and diagnose faults in production systems)

HTTPS://WWW.GEEKSFORGEEKS.ORG/INTRODUCTION-TO-FAULTS-IN-SOFTWARE-ENGINEERING/

Hardware Tactics

1. Use redundancy to provide backup components for critical functions. For example, redundant servers, storage devices, and network connections can ensure that the system remains operational even if one component fails.
2. Implement load balancing to allow an application to run on multiple network nodes, removing the concern about a single point of failure.
3. Use data backups to ensure that data is not lost in the event of a failure or error. Regular backups should be taken and stored in a secure location.
4. Implement fault detection and isolation mechanisms to identify failing components and isolate them from the rest of the system to prevent cascading failures.



HTTPS://WWW.IMPERVA.COM/LEARN/AVAILABILITY/FAULT-TOLERANCE/

7) **The system must be designed with a loosely coupled architecture, promoting flexibility and scalability.**

Quality Scenario

Source: The need for flexibility and scalability in the system.

Stimulus: A change is made to one component of the system.

Artifact: The software components of the system.

Environment: The system is in normal operation.

Response: The system should be designed with a loosely coupled architecture to ensure that changes to one component do not affect the functioning of other components. The components should communicate through standardized interfaces, and changes to one component should be isolated from other components.

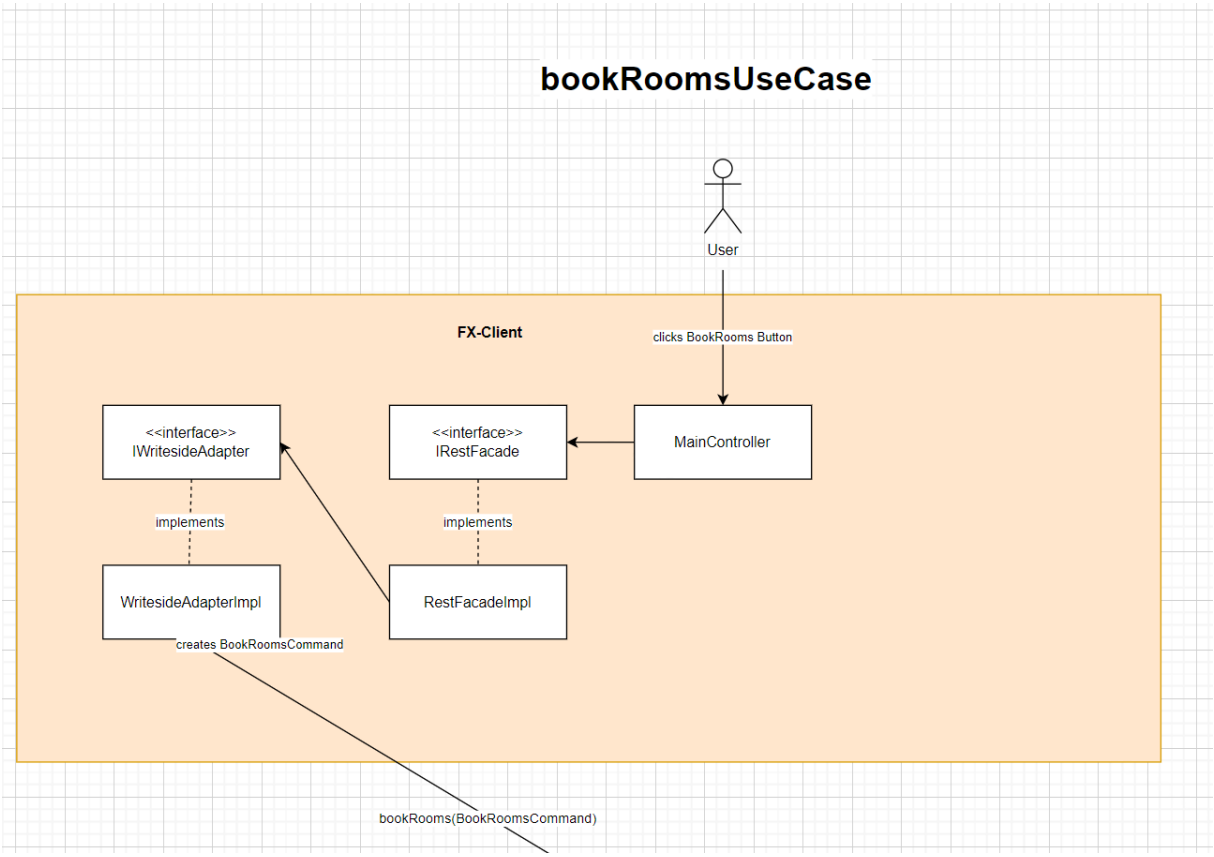
Response Measure: The system should be able to accommodate changes to individual components without affecting the performance of the entire system. The time required to make changes to one component should be minimal, and the system's performance should not degrade during the change process.

Tactics

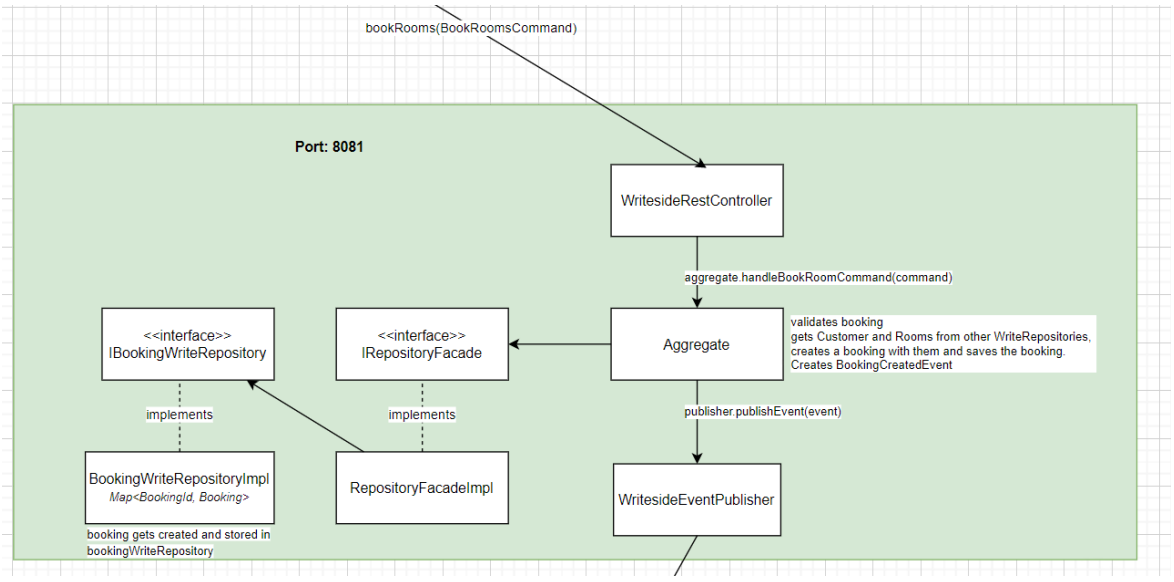
1. Use standard interfaces and protocols to ensure compatibility between components.
2. Use message queues or event-driven architectures to enable asynchronous communication between components.
3. Use containerization or microservices to isolate components and enable easy deployment and scaling. RESTful web services have the advantage of platform independence, meaning that they can be used with different programming languages and on different operating systems and platforms. By using RESTful web services, systems can be loosely coupled, as they can function independently of each other without a strong dependency. A RESTful service provides uniform interfaces to exchange data between different applications without requiring direct access to the underlying systems.

CQRS Example:

WritesideAdapter could easily be replaced...technology which is used to communicate with the server could be changed, etc. The Main Controller will have to change if the adapter changes.



Instead of the BookingWriteRepositoryImplClass which stores the bookings in lists, we could exchange to BookingWriteRepositoryImpl with a JpaAdapter Class which communicates with a PostgreSQL Database. The only thing in the application that would have to be changed would be the Injected class in the IBookingWriteRepository Interface. The Façade would not have to change.



8) **The system must be secure from external threats, such as hacking and data breaches.**

Quality Scenario

Scenario: The system must be secure from external threats such as hacking and data breaches, ensuring the confidentiality, integrity, and availability of the system and its data.

Source: The need for system security from external threats.

Stimulus: An external attacker attempts to gain unauthorized access to the system or data.

Artifact: The system's security measures, including access control, authentication, encryption, and intrusion detection/prevention. Environment: Normal operation, degraded operation, run time.

Response: The system should be designed with robust security measures, including access control, authentication, encryption, and intrusion detection/prevention, to prevent unauthorized access, data breaches, and other security threats. The system should also have a clear incident response plan in case of a security breach.

Response Measure: The system's security measures should be regularly tested and audited to ensure their effectiveness and compliance with relevant security standards and regulations.

Tactics

1. Use secure authentication mechanisms, such as two-factor authentication, to ensure that only authorized users can access the system.
2. Implement strong access controls, to restrict access to sensitive data and functions.
3. Use encryption to protect sensitive data in transit and at rest.
4. Implement intrusion detection/prevention systems (IDS/IPS) to detect and block suspicious activity.
5. Regularly update and patch the system's software and hardware to address known vulnerabilities.
6. Conduct regular security audits and penetration testing to identify and address potential security weaknesses.
7. Have an incident response plan in place to respond quickly and effectively to security breaches.

HTTPS://WWW.IMPERVA.COM/LEARN/DATA-SECURITY/DATA-ENCRYPTION/

9) The system must be internationalized, able to support multiple languages and cultural norms.

Quality Scenario

Scenario: Internationalization for Multi-language and Cultural Support

The system must be internationalized, able to support multiple languages and cultural norms, to ensure that users from different regions can use the system effectively.

Source: The need to support users from different regions with varying languages and cultural norms.

Stimulus: The user interacts with the system and selects their preferred language.

Artifact: The user interface, including text, images, and other multimedia.

Environment: Normal operation.

Response: The system should be internationalized to support multiple languages and cultural norms. The user interface should be designed to accommodate different languages and cultural norms, with appropriate text, images, and other multimedia.

Response Measure: The system should be able to support different languages and cultural norms, providing a seamless experience for users regardless of their region.

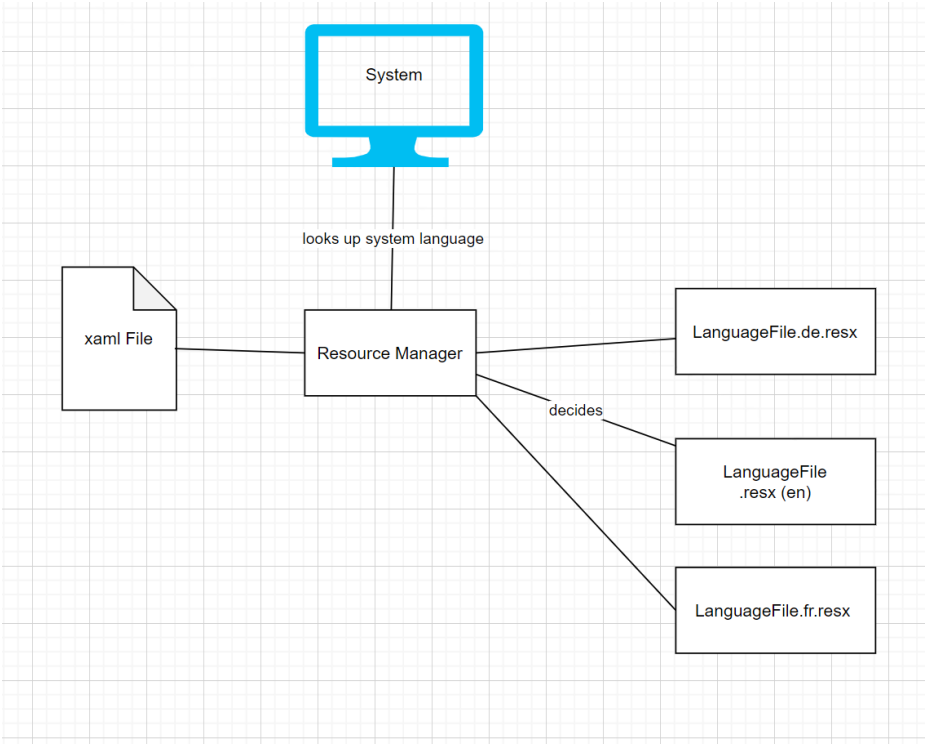
Tactics

1. Use a framework that supports multi-language and cultural norms.
2. Use the system language per default.
3. Use Unicode for encoding text to support different languages and character sets.
4. Use appropriate images and multimedia to accommodate different cultural norms and preferences.
5. Test the system with users from different regions to ensure that it functions correctly with different languages and cultural norms.

This graphic shows how multilanguage support would work in C#.

The xaml File (like fxml in Java) would use placeholders instead of Text as String. During runtime the Resource Manager would look up the System Language and use the corresponding .resx File.

The .resx File contains key - value pairs. The key is the same in all files and is the placeholder in the xaml File. The value is the translation of the word.



10) The system must be designed with a service-oriented architecture, promoting interoperability and reuse.

"SOA is a loosely coupled architecture. It means when the System talks with each other the dependency which is required to talk is minimal.

Failure of one system doesn't impact other system(s)."

Quality Scenario

Scenario: Service-Oriented Architecture for Interoperability and Reuse

The system must be designed with a service-oriented architecture to promote interoperability and reuse, ensuring that the system is modular and flexible enough to accommodate changing requirements.

Source: The need for interoperability and reuse in the system.

Stimulus: The user interacts with the system, requiring access to data or functionality from another system.

Artifact: The system architecture, including services, interfaces, and components.

Environment: Normal operation.

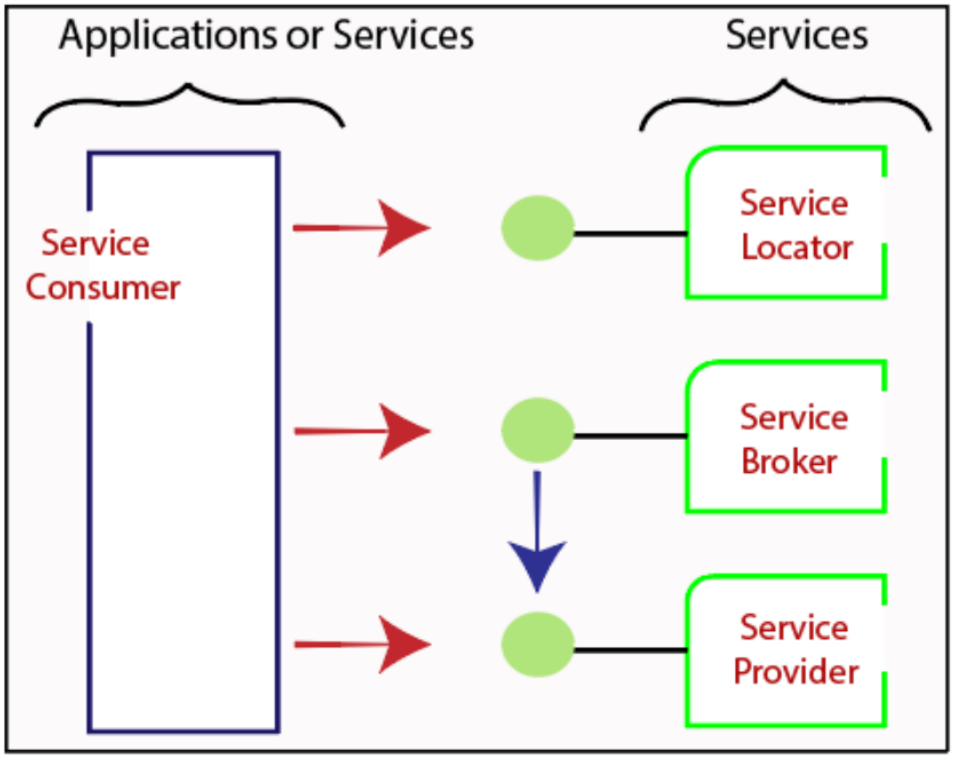
Response: The system should be designed with a service-oriented architecture to promote interoperability and reuse. The system should be modular and flexible, with clearly defined services and interfaces that can be reused by other systems.

Response Measure: The system's service-oriented architecture should promote interoperability and reuse, with clearly defined services and interfaces that can be reused by other systems.

Tactics

Use a microservices architecture to break down the system into smaller, more manageable components.

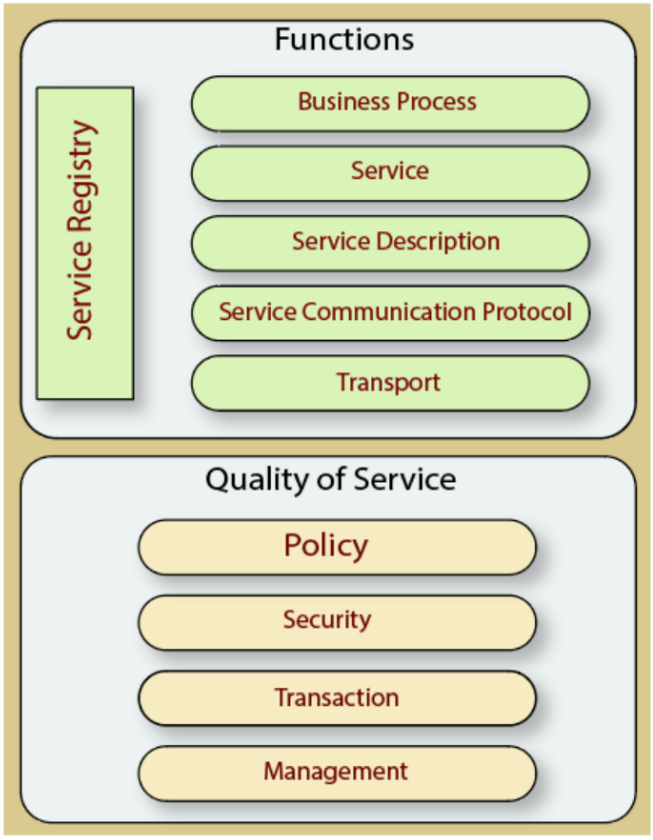
HTTPS://WWW.JAVATPOINT.COM/SERVICE-ORIENTED-ARCHITECTURE



- **Services -** The services are the logical entities defined by one or more published interfaces.
- **Service provider -** It is a software entity that implements a service specification.
- **Service consumer -** It can be called as a requestor or client that calls a service provider. A service consumer can be another service or an end-user application.
- **Service locator -** It is a service provider that acts as a registry. It is responsible for examining service provider interfaces and service locations.
- **Service broker -** It is a service provider that pass service requests to one or more additional service providers.

Components of service-oriented architecture

The service-oriented architecture stack can be categorized into two parts - functional aspects and quality of service aspects.



11) The system must be designed with a responsive design approach, promoting usability and accessibility on various screen sizes and resolutions.

Quality Scenario

Scenario: Responsive Design Approach for Usability and Accessibility on Various Screen Sizes and Resolutions

The system must be designed with a responsive design approach, promoting usability and accessibility on various screen sizes and resolutions, to ensure that users can access and use the system on any device.

Source: The need for the system to be accessible and usable on various screen sizes and resolutions.

Stimulus: The user interacts with the system on a device with a different screen size or resolution.

Artifact: The user interface, including layout, text, images, and other multimedia.

Environment: Normal operation.

Response: The system should be designed with a responsive design approach to promote usability and accessibility on various screen sizes and resolutions. The user interface should be designed to adapt to different screen sizes and resolutions, with appropriate layout, text, images, and other multimedia.

Response Measure: The system's responsive design approach should promote usability and accessibility on various screen sizes and resolutions, providing a seamless experience for users regardless of their device.

Tactics

1. Use a responsive design framework, such as Bootstrap or Foundation, to ensure that the user interface can adapt to different screen sizes and resolutions.
2. Use appropriate layout, text, images, and other multimedia to accommodate different screen sizes and resolutions.
3. Test the system on different devices with varying screen sizes and resolutions to ensure that it functions correctly.