



HandyConnect360

Software Architecture Documentation

Team Member

Israt Jahan Jhumu (BKH2025020F)
MD. Sanwar Hossain (MUH2025018M)
Hasanur Rahman Shishir (MUH2025022M)
Irfanul Haque Nabil (ASH1925021M)
Khos Mahmuda Akter(BKH2025036F)

Course Teacher

Dipok Chandra Das
Assistant Professor
Institute of Information Technology
Noakhali Science and Technology University



Submission Date : 13.2.2024

1. Introduction	4
1.1 Purpose	4
1.2 Scope	4
1.3 Definitions	4
1.4 Acronyms and Abbreviation	6
1.4 References	7
1.5 Overview	7
2. Architecturally Significant Requirements	7
3. Quality Attributes	8
3.1. Quality attributes scenarios for my system:	9
1. Reliability	9
2. Availability	9
3. Usability	9
4. Performance	10
5. Security	10
6. Scalability	10
7. Maintainability	11
8. Interoperability	11
9. Testability	12
10. Adaptability	12
3.2. QA Utility Tree	12
3.3 Design Tactics	14
3.3.1 Availability Tactics:	14
3.3.2 Performance Tactics:	16
3.3.3 Usability Tactics:	17
3.3.4 Interoperability tactics:	18
3.3.5 Security tactics:	20
3.3.6 Maintainability Tactics	21
3.3.7 Adaptability Tactics	24
3.3.8 Testability Tactics	26
4. Architectural Representation	27
4.1 Context diagrams	27
4.2 Use-case Diagram	28
4.2.1 Use Case Description	29
4.3 Logical View	30
4.3.1 Entities	30
4.3.2 Relationships	31
4.3.3 Attributes	31
4.3.4 Constraints	32
Logical View Diagram	32
4.4 Process View	33
4.4.1 Search Serviceman	33
4.4.2 Find Info of Serviceman	33
4.4.3 Send Hire Request	33

4.4.4 Accept Hire Request	33
4.4.5 Cancellation Process	33
4.4.6 Communication Channels	34
4.4.7 Database	34
Process View Diagram	35
Sequence Diagram	36
4.5 Deployment View	37
4.5.1 Client Devices:	37
4.5.2 Web Servers:	38
4.5.3 C-Web Servers:	38
4.5.4 Database Servers:	38
4.5.5 Connections & Interactions:	38
Deployment View Diagram	39
4.6 Implementation view	40
5. Architectural Goal and Constraints	41
Appendix	43
A. Design Processes	43
B. Architecture Patterns	43
MVC (Model-View-Controller) Pattern	43
Client-Server Architecture	45
Broker Pattern	45
Pipe-Filter Pattern	46

Table of Figures

Figure 1 : QA utility tree	15
Figure 2 : Availability Tactics.....	17
Figure 3:Performance Tactics.....	18
Figure 4:Usability Tactics.....	19
Figure 5:Interoperability Tactics.....	20
Figure 6:Security Tactics	22
Figure 7:Maintainability Tactics.....	23
Figure 8:Scalability Tactics	25
Figure 9:Adaptability Tactics	26
Figure 10:Testability Tactics	27
Figure 11:Context diagrams.....	28
Figure 12:: Use-case diagram	29
Figure 13:Logical View of HandyConnect360	34
Figure 14:Process View of HandyConnect360	37
Figure 15:Sequence Diagram of HandyConnect360	39
Figure 16:Deployment View of HandyConnect360	41
Figure 17:Implementation View of HandyConnect360	42
Figure 18:Design Process of HandyConnect360	45
Figure 19:: MVC pattern of HandyConnect360.....	49
Figure 20:: Client-Server Architecture Pattern of HandyConnect360	50
Figure 21:Broker Pattern of HandyConnect360	50
Figure 22:Pipe-Filter Pattern of HandyConnect360	51

Revision History

Date	Version	Description	Author
<13/02/2024>	<1.0>	Initialization of first version of Software Architecture Report of HandyConnect360	< Israt Jahan Jhumu MD. Sanwar Hossain Hasanur Rahman Shishir Irfanul Haque Nabil Khos Mahmuda Akter >

1. Introduction

1.1 Purpose

The purpose of a software architecture document is to provide the architectural overview of the software system. The document contains different architectural views to show different aspects of the system. It helps the developer in making decisions regarding the full-stack development of the system. It is the communication medium between the software architect and the developer.

Here the purpose of this document is to provide a comprehensive overview of the architecture of HandyConnect360, a platform designed to simplify the process of connecting users with reliable skilled service providers for home repairs and other services. This document will outline the system's components, interactions, and key considerations such as scalability, security, and deployment.

1.2 Scope

This Document provides the architectural overview of the Goods delivery mobile application and its backend panel. This architectural document covers the high-level design and structure of HandyConnect360 software. It outlines the major components of the system, their interactions, and the technologies used to implement them. However, detailed implementation specifics and low-level design decisions are out of scope for this document. The document guides the developer about the compatible technology, databases, and development techniques required for this mobile application development.

1.3 Definitions

User Account Creation

Feature: User Registration

Description: The essential process enabling users to create personalized accounts within the HandyConnect360 ecosystem. This functionality encompasses secure storage of user credentials, such as usernames and passwords, ensuring data integrity and privacy through encryption protocols and robust authentication mechanisms.

Location Setting

Feature: Location Identification

Description: This pivotal feature empowers users to specify their geographical coordinates or address accurately within the HandyConnect360 platform. It involves seamless integration of geolocation services, facilitating efficient service recommendations and streamlined interactions between users and service providers.

Service Search

Feature: Service Discovery

Description: An indispensable tool allowing users to explore and discover various services offered within HandyConnect360. Leveraging advanced indexing techniques and intuitive search algorithms, this feature ensures swift and precise retrieval of relevant services tailored to user preferences and requirements.

Service Listing

Feature: Service Presentation

Description: This feature presents users with a comprehensive list of available services within the HandyConnect360 platform. It encompasses efficient retrieval and elegant presentation of service offerings, enhancing user experience and facilitating informed decision-making.

Problem Posting

Feature: Service Request Submission

Description: Facilitating the seamless submission of service requests or problems within HandyConnect360, this feature underpins user engagement and service provider interaction. Robust backend APIs and secure data storage mechanisms ensure the integrity and reliability of user-submitted requests.

Emergency Posting

Feature: Emergency Service Request

Description: A critical functionality designed to prioritize and expedite emergency service requests within HandyConnect360. This feature employs specialized workflows and notification systems, ensuring prompt attention and swift resolution of urgent user needs.

Scheduled Posting

Feature: Scheduled Service Requests

Description: Empowering users to schedule service requests or appointments for future dates and times, this feature fosters convenience and flexibility within HandyConnect360. It includes intuitive scheduling interfaces and efficient backend processes to manage and coordinate scheduled service activities.

Work Request Management

Feature: Request Management

Description: Central to user interaction, this feature enables users to track and manage their service requests seamlessly within HandyConnect360. From initiation to completion, robust backend processes and intuitive user interfaces ensure efficient request handling and user satisfaction.

Rating Provision

Feature: Service Rating

Description: Enabling users to provide feedback and ratings on services received, this feature enhances transparency and accountability within HandyConnect360. It encompasses user-friendly rating interfaces and sophisticated backend algorithms for aggregating and presenting service ratings.

Profile Updating

Feature: Profile Management

Description: Crucial for maintaining accurate user information, this feature allows users to update their profiles within HandyConnect360 easily. With intuitive profile editing interfaces and secure backend processes, users can ensure their profiles reflect current information accurately.

1.4 Acronyms and Abbreviation

- **UI:** User Interface
- **API:** Application Programming Interface
- **DB:** Database
- **DMBS:** Database Management System
- **HTTP:** Hypertext Transfer Protocol
- **HTTPS:** Hypertext Transfer Protocol Secure.
- **CI/CD:** Continuous Integration/Continuous Deployment
- **SQL:** Structured Query Language
- **IoT:** Internet of Things
- **DNS:** Domain Name System
- **MVC:** Model-View-Controller Pattern.
- **SLA:** Service Level Agreement.
- **QA:** Quality Assurance.

1.4 References

1. IEEE. *IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications*. IEEE Computer Society, 1998.
2. *Software Architecture: Foundations, theory and practice*. Richard N. Taylor, Nenad
3. *Medvidovic, Eric Dashofy*. Wiley, 2010
4. *EBW Software Design: From Programming to Architecture*. Eric Braude. Wiley, 2004
5. *FRH Pattern Oriented Software Architecture: A System of Patterns*. Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal. Wiley, 1995
6. <https://www.edrawmax.com/online/app.html?>
7. <https://www.edrawmax.com/online/app.html?>

1.5 Overview

HandyConnect360 stands as a comprehensive platform, reshaping the landscape of connecting users with proficient service providers for a diverse array of home repairs and services. This sophisticated system's architectural blueprint, delineated in this document, unveils a multifaceted high-level overview encapsulating the intricacies of its design and functionalities. The software architecture document delves into the intricacies of the user journey, encompassing an array of capabilities such as service searching, problem posting, emergency posts, scheduled posts, work request management,

rating provisions, profile updates, and an array of interactive features aimed at enriching the overall user experience.

A pivotal focus of HandyConnect360's architectural design is on the architecturally significant requirements vital for its robust operation and scalability. The platform is engineered to seamlessly scale, adeptly handling a substantial user base and a myriad of service requests. This scalability is achieved through the implementation of distributed web servers, ensuring optimal performance even in the face of concurrent usage, with a target capacity of accommodating up to 1 lakh users concurrently. Security and authentication mechanisms take centre stage, with the platform employing robust encryption protocols and fortified defences against prevalent threats like SQL injection and cross-site scripting. This commitment to security extends to user data and interactions, ensuring a trustworthy digital environment. Reliability and availability are paramount considerations, steering architectural decisions towards the deployment of distributed web servers and databases. In essence, HandyConnect360 aspires to be more than a service connection platform; it aims to be a sophisticated ecosystem that not only meets but exceeds user expectations. The confluence of scalability, security, reliability, performance, and user experience within its architectural framework underscores its commitment to providing a seamless and enriching experience for both users and service providers.

1.6. Functional requirement

- 1.** Create an account(Both the house owner and service provider have to create an account)
- 2.** Sets Location- (Both users have to set their location)
- 3.** Customer Search a Service(Customer Search a Service or select Category)
- 4.** Show Service List
- 5.** Post a problem
- 6.** Emergency post
- 7.** Schedule post
- 8.** Sent work request
- 9.** Cancel work request
- 10.** Provide rating
- 11.** Ability to update profile
- 12.** User should be filtering search
- 13.** User should be able to communicate serviceman using System
- 14.** User should be able to view history and upcoming service list
- 15.** User should be able to request multiple service at same time
- 16.** User should be notify about any update of their work
- 17.** User should be able to see real time location.
- 18.** User should be able to see estimated arrival time of serviceman
- 19.** User should be able to access system using multiple device.
- 20.** Serviceman should be able to view reviews and reply them.
- 21.** Customer should be able to modify his review
- 22.** Serviceman should be able to change his availability status
- 23.** Serviceman should be able to set his service area
- 24.** User can submit report against illegal activities
- 25.** User should be able to direct communicate with system authority

2. Architecturally Significant Requirements

Architecturally Significant Requirements (ASRs) are a subset of system requirements that have a **measurable impact on the overall architecture** of the system. They influence key decisions about the system's structure, components, technologies, and interfaces. Identifying and addressing these requirements early in the development process is crucial to ensure the chosen architecture effectively meets all needs.

Architecturally significant requirements of HandyConnect360 System:

- ❖ **Scalability:** The system should be able to handle a large number of users, servicemen, and service requests without compromising performance by using distributed web servers. The system must be able to handle 1 lakh concurrent users.
- ❖ **Security and Authentication:** Ensuring secure access for users and servicemen is essential. Robust authentication mechanisms, data encryption, and protection against common security threats (such as SQL injection or cross-site scripting) are architecturally significant.
- ❖ **Reliability and Availability:** The system must be highly available, minimizing downtime using distributed web servers and database. Architectural decisions related to redundancy, failover mechanisms, and fault tolerance play a critical role.
- ❖ **Performance:** The platform should respond promptly to user requests. ASRs related to response times, throughput, and resource utilization are vital. The system must respond to user requests within 1 second. Search results are shown within 2 seconds (< 2s).
- ❖ **Search Functionality:** The platform must include a robust search functionality that allows homeowners to easily search for service providers based on location and specialty. This requires efficient indexing and search algorithms to deliver relevant results quickly.
- ❖ **Reviews and Ratings System:** The system should incorporate a reviews and ratings system that enables previous customers to provide feedback on service providers' work. This helps homeowners evaluate the quality of service and make informed hiring decisions.
- ❖ **Messaging and Appointment Scheduling:** A messaging system should facilitate communication between homeowners and service providers, allowing them to discuss job requirements and schedule appointments seamlessly within the platform.
- ❖ **Data Management:** Architectural choices regarding data storage (e.g., databases, caching), retrieval, and consistency are significant. Consider factors like data partitioning, indexing, and data integrity.
- ❖ **Service Integration:** The system needs to integrate with external services (e.g., payment gateways, location services). Architectural decisions around APIs, protocols, and communication patterns are crucial.
- ❖ **User Experience (UX):** ASRs related to the user interface, navigation, and responsiveness impact the overall user experience. Designing an intuitive and efficient UI is architecturally significant.
- ❖ **Scalable Infrastructure:** Decisions about hosting, cloud services, load balancers, and auto-scaling are critical for handling varying workloads.
- ❖ **Service Discovery and Load Balancing:** Architectural components that enable dynamic service discovery and efficient load balancing are essential for scalability and fault tolerance.
- ❖ **Auditability and Logging:** Ensuring traceability of user actions, system events, and errors is architecturally significant. Proper logging mechanisms are crucial for troubleshooting and compliance.

3. Quality Attributes

Quality attributes are characteristics of a system that describe its non-functional properties. They are used to judge the overall quality of a system from the user's perspective.

Quality attributes of HandyConnect360 System:

1. **Reliability:** Ensuring that the HandyConnect360 platform consistently performs its intended functions without unexpected failures. Users should be able to rely on it for finding reliable servicemen.
2. **Availability:** The system should be accessible and operational whenever users need it. Downtime should be minimized to provide uninterrupted service.
3. **Usability:** The user interface should be intuitive, easy to navigate, and efficient. Users should find it straightforward to search for servicemen, read reviews, and make informed decisions.
4. **Performance:** The system's responsiveness and speed are crucial. Queries for servicemen should yield results promptly ($< 2s$), especially during peak usage times.
5. **Security:** Protecting user data, ensuring privacy, and preventing unauthorized access are critical. Safeguarding against potential threats is essential.
6. **Scalability:** As the user base grows, the system should handle increased load without compromising performance. Scalability ensures it can accommodate more users and servicemen.
7. **Maintainability:** The system should be designed for ease of maintenance, updates, and enhancements. Clear documentation and modular architecture contribute to maintainability.
8. **Interoperability:** HandyConnect360 should seamlessly integrate with other systems or platforms, allowing data exchange and collaboration.
9. **Testability:** HandyConnect360 should ensure that the system is designed and implemented in a way that facilitates effective testing, leading to reliable and maintainable software.
10. **Adaptability:** The system should be flexible enough to accommodate changes in requirements, business rules, or regulations.

3.1. Quality attributes scenarios of HandyConnect360:

1. Reliability

- ✚ **Scenario:** Handling a critical service request during peak hours when a user attempts to hire a serviceman for urgent repair work.
- ✚ **Source:** User attempting to hire a serviceman for urgent repair work.
- ✚ **Stimulus:** Critical service request during peak hours.
- ✚ **Environment:** Normal operation with high demand.
- ✚ **Artifact:** Service request processing module.
- ✚ **Response:** The system should promptly assign a qualified serviceman and notify the user.
- ✚ **Response measure:** Service request fulfilment time (e.g., within 2 hours).

2. Availability

- + **Scenario:** Ensuring the system remains accessible without downtime when users access the platform with frequent login attempts and service requests. Aim for high availability (e.g., 99.9% uptime).
- + **Source:** User accessing the platform.
- + **Stimulus:** Frequent login attempts and service requests.
- + **Environment:** Normal operation or peak load.
- + **Artifact:** Web server and database.
- + **Response:** The system should remain accessible without downtime.
- + **Response measure:** System uptime (e.g., 99.9% availability).

3. Usability

- + **Scenario:** New User Searching for a Specific Type of Serviceman.
- + **Source:** New user navigating the platform.
- + **Stimulus:** Attempting to search for a specific type of serviceman.
- + **Environment:** Normal operation.
- + **Artifact:** User interface (search filters, reviews section).
- + **Response:** The system should present relevant search results and user-friendly options.
- + **Response measure:** User satisfaction score (e.g., average rating of 4 out of 5).

4. Performance

- + **Scenario:** Each query should be less than 2 seconds, ensuring that users receive search results promptly even during peak load conditions.
- + **Source:** Multiple users searching for servicemen simultaneously.
- + **Stimulus:** High query volume.
- + **Environment:** Peak load conditions.
- + **Artifact:** Search functionality and database.
- + **Response:** The system should provide search results within an acceptable time frame.
- + **Response measure:** Response time (e.g., < 2 seconds per query).

5. Security

- + **Scenario:** Preventing unauthorized access attempts and protecting user data.
- + **Source:** Unauthorized access attempt.
- + **Stimulus:** Malicious login or data manipulation.
- + **Environment:** Normal operation.
- + **Artifact:** Authentication module and user data.
- + **Response:** The system should prevent unauthorized access and protect user data.
- + **Response measure:** Number of detected security breaches (aim for zero).

6. Scalability

- ✚ **Scenario:** The system is able to handle a large number (1 lakh) of concurrent user requests.
- ✚ **Source:** One million concurrent users accessing the platform.
- ✚ **Stimulus:** High volume of search queries, service requests, and reviews.
- ✚ **Environment:** Peak load conditions.
- ✚ **Artifact:** Overall system architecture, web servers, and database clusters.
- ✚ **Response:** The system should efficiently distribute and manage user requests without performance degradation.
- ✚ **Response measure:** Concurrent users supported (e.g., successfully handling one lakh concurrent users).

7. Maintainability

- ✚ **Scenario:** The development team adding a new service category (e.g., HVAC repair) to the platform
- ✚ **Source:** Development team adding a new service category (e.g., HVAC repair).
- ✚ **Stimulus:** Extending the platform's offerings.
- ✚ **Environment:** Development phase.
- ✚ **Artifact:** Codebase and database schema.
- ✚ **Response:** The system should allow seamless integration of new service categories.
- ✚ **Response measure:** Time taken to add a new category (< 1 week).

8. Interoperability

- ✚ **Scenario:** Count the successful integrations with other systems (e.g., payment gateways, third-party APIs). Aim for seamless data exchange.
- ✚ **Source:** Integration with a payment gateway.
- ✚ **Stimulus:** User making a payment.
- ✚ **Environment:** Normal operation.
- ✚ **Artifact:** Payment integration module.
- ✚ **Response:** The system should successfully process payments.
- ✚ **Response measure:** Number of successful integrations.

9. Testability

- ✚ **Scenario:** Implement unit/integration tests to cover interactions between system components, aiming for at least 90% coverage of unit/integration scenarios.
- ✚ **Source:** Testing team or automated test suite.
- ✚ **Stimulus:** Running test cases (unit tests, integration tests, etc.).
- ✚ **Environment:** Development or testing phase.
- ✚ **Artifact:** Codebase, modules, and interfaces.
- ✚ **Response:** The system should exhibit the following characteristics:
 - **Ease of Test Creation:** Test cases can be designed and implemented efficiently.
 - **Isolation of Components:** Modules can be tested independently without complex dependencies.
 - **Visibility of Internal State:** Relevant internal state (variables, data structures) is accessible for testing.
 - **Mockability and Stubbing:** External dependencies (e.g., APIs, databases) can be replaced with mock objects or stubs.

- **Response measure:** High test coverage (percentage of code exercised by tests), low test maintenance effort.

10. Adaptability

- ✚ **Scenario:** The new changes will be adjusted properly within 1 days.
- ✚ **Source:** Regulatory changes.
- ✚ **Stimulus:** New requirements or business rules.
- ✚ **Environment:** Normal operation.
- ✚ **Artifact:** System components.
- ✚ **Response:** The system should accommodate changes promptly.
- ✚ **Response measure:** Change request response time (e.g., < 24 hours).

3.2. QA Utility Tree

“Utility” to express the overall “goodness” of the system

QA utility tree construction:

- Most important QA goals are high level nodes (typically performance, modifiability, security, and availability). Scenarios are the leaves
- Output: a characterization and prioritization of specific quality attribute requirements.
- High/Medium/Low importance for the success of the system
- High/Medium/Low difficulty to achieve (architect’s assessment)

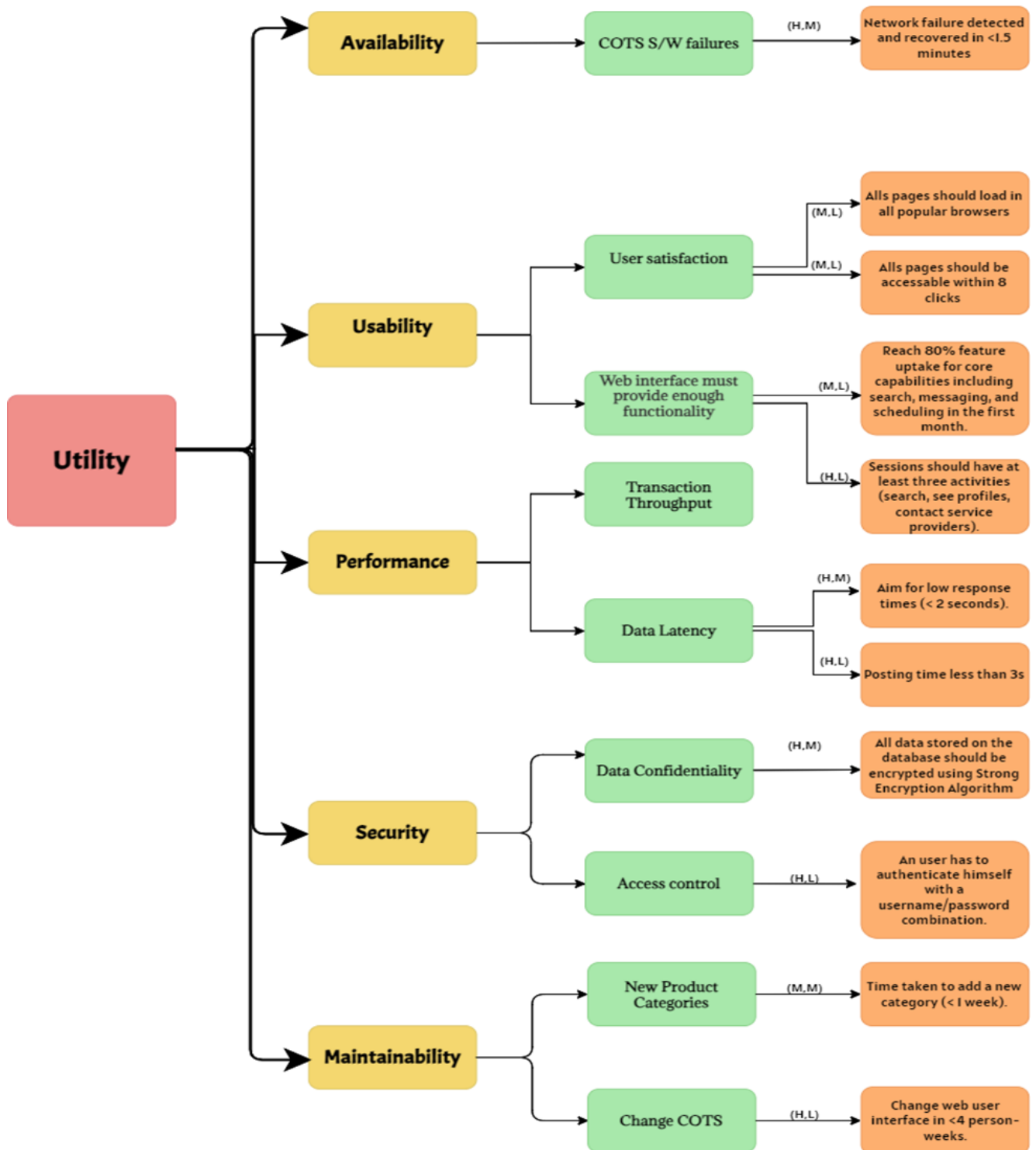


Figure 1 : QA utility tree

3.3 Design Tactics

3.3.1 Availability Tactics:

Availability in software is the measure of how reliably a system is accessible and operational when required, indicating minimal downtime.

Availability tactics in software involve strategies to ensure a system remains operational. These include redundancy, failover mechanisms, and load balancing, minimizing disruptions and enhancing the continuous availability of the software. Here some tactics-

1. Ping/Echo :

HandyConnect360 uses a ping/echo mechanism to check the availability of its online platform. It regularly sends signals to its servers, and if the servers respond, it confirms that the platform is reachable and operational.

2. Heartbeat :

HandyConnect360 employs a heartbeat system to ensure the continuous functionality of its service. Devices within the platform exchange periodic signals to confirm their operational status, helping to detect and address any issues promptly.

3.Timestamp :

HandyConnect360 logs timestamps for various events, such as job requests, handyman responses, and completed tasks. Timestamps help in tracking the sequence of events and providing users with a clear timeline of their service interactions.

4. Passive Redundancy :

HandyConnect360 incorporates passive redundancy in its server setup. Backup servers remain inactive during regular operations but become active if the primary servers fail, ensuring uninterrupted service for users.

5. Spare :

HandyConnect360 maintains spare capacity to handle an increased number of job requests or to replace a temporarily unavailable handyman, ensuring a smooth experience for users.

6. Rollback :

If a recent update or change causes issues in the HandyConnect360 platform, the system can initiate a rollback to revert to the previous stable version, minimizing disruptions for both users and handymen.

7. Shadow :

HandyConnect360 may have a shadow environment for testing new features or updates before deploying them to the live platform. This ensures that any potential issues are identified and addressed before affecting the user experience.

8. Non-Stop Forwarding :

HandyConnect360 employs non-stop forwarding capabilities to ensure uninterrupted service, even during maintenance or software updates. This enables users to continue hiring handymen without experiencing downtime.

9. Removal from Service :

HandyConnect360 may temporarily remove a handyman from service for maintenance or if they violate platform policies. This ensures that users are connected with reliable and compliant service providers.

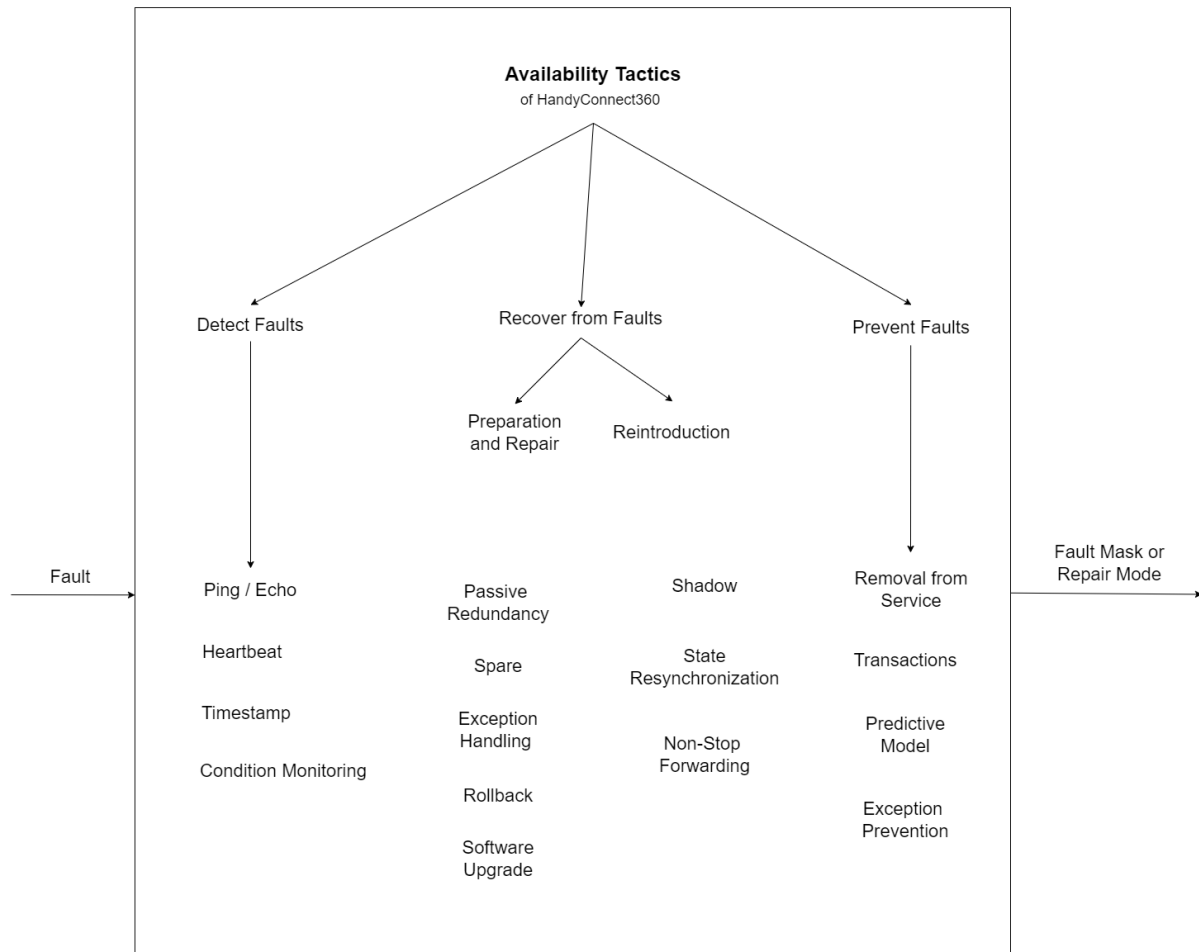


Figure 2 : Availability Tactics

3.3.2 Performance Tactics:

Performance in the context of software refers to the efficiency and speed with which a system or application executes tasks and delivers results. It encompasses factors such as response time, throughput, and resource utilization. Performance tactics are strategies or techniques employed to optimize and enhance the efficiency of a software system. These may include caching, load balancing, parallel processing, and other methods aimed at improving response times, reducing latency, and optimizing resource utilization. Here some tactics-

1. Manage Sampling Rate:

Adjusting the frequency of data collection in HandyConnect360 optimizes performance while still gathering essential information for analytics and system monitoring.

2. Limit Event Response:

HandyConnect360 restricts the system's reaction to specific events to prevent unnecessary actions, ensuring a balanced and controlled response to events such as user requests or system notifications.

3. Prioritize Events:

Assigning importance levels to different events in HandyConnect360 allows the system to focus on critical tasks first, optimizing responsiveness and ensuring timely handling of high-priority activities.

4. Removal from Service:

Temporarily taking a handyman offline for maintenance or policy violations in HandyConnect360

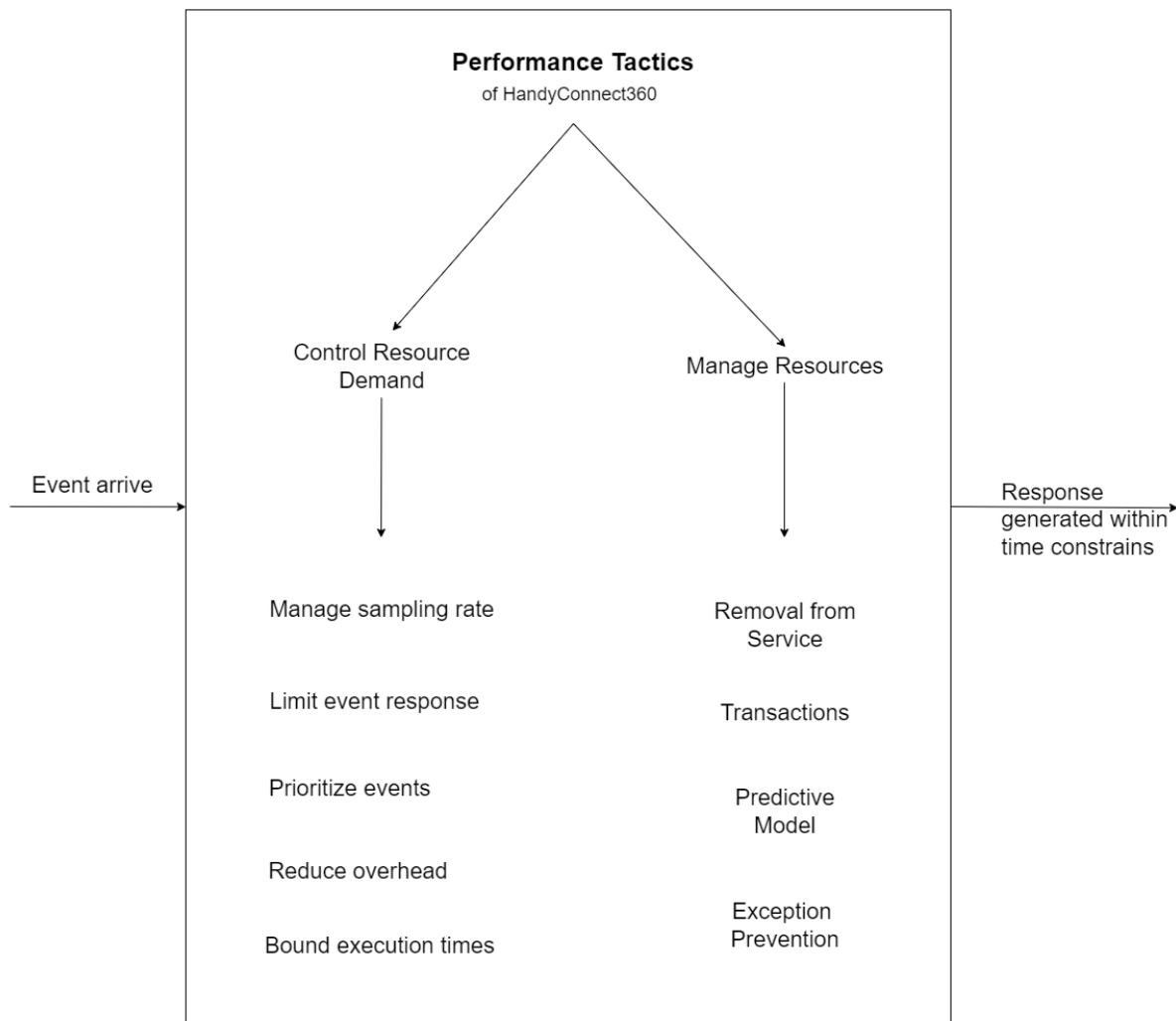


Figure 3:Performance Tactics

ensures that users are connected with reliable and compliant service providers.

5. Transaction:

HandyConnect360 manages user interactions and transactions efficiently, ensuring smooth and reliable exchanges between users and handymen while maintaining the integrity and consistency of the platform.

6. Predictive Model:

Utilizing predictive models in HandyConnect360 helps forecast user demands, aiding in resource planning, service allocation, and ensuring proactive measures are taken to meet user needs effectively.

7. Exception Prevention:

Implementing measures in HandyConnect360 to proactively identify and address potential issues before they escalate prevents exceptions or errors that could impact the user experience or system stability.

3.3.3 Usability Tactics:

Usability refers to the extent to which a software product or system is user-friendly, allowing users to interact with it easily, efficiently, and with satisfaction. Usability tactics are specific strategies or techniques implemented to enhance the user-friendliness of a system. These may include intuitive user interfaces, clear navigation, consistent design elements, and other measures aimed at improving the overall user experience and satisfaction. Here some tactics-

1. Support User Initiative :

HandyConnect is designed to support user initiative by providing an intuitive platform that allows users to easily browse, select, and connect with handymen. The interface is user-friendly, empowering individuals to initiate and manage their service requests effortlessly.

2. Support System Initiative :

In HandyConnect, the system takes initiative by offering intelligent suggestions, such as recommending highly-rated handymen based on user preferences and historical data. The platform proactively guides users through the selection process, enhancing efficiency and ensuring a seamless experience.

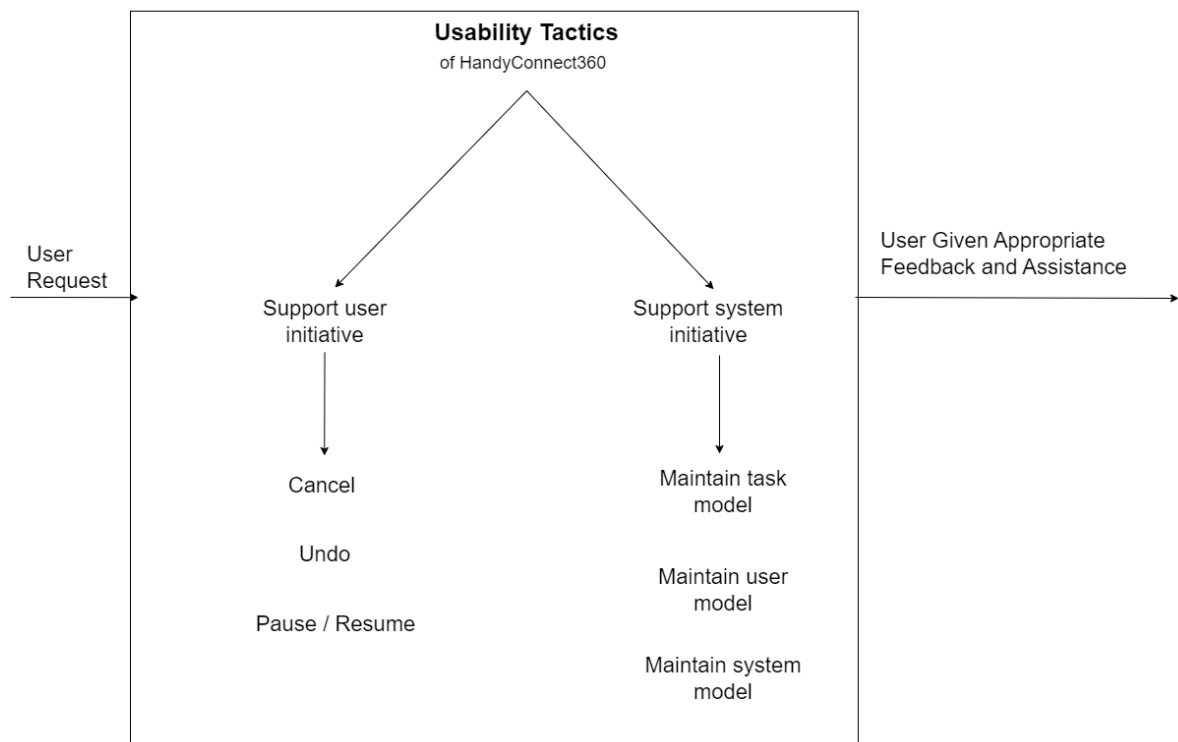


Figure 4:Usability Tactics

3.3.4 Interoperability tactics:

Interoperability refers to the ability of different software systems or components to seamlessly exchange and use information, ensuring effective communication and collaboration between diverse technologies.

Interoperability tactics involve implementing strategies to enable smooth interaction between different systems. These may include using standardized protocols, APIs (Application Programming Interfaces), and data formats to facilitate seamless data exchange and integration across various software applications and platforms.

1.Discover Service:

The capability in HandyConnect involves enabling users to effortlessly explore and find relevant handyman services. It includes features such as search functionalities, service categorization, and recommendations to help users discover the specific services they need.

2.Orchestrate:

Within HandyConnect, orchestration involves coordinating and managing the workflow of service requests and job assignments. The platform efficiently organizes tasks, communicates with handymen, and ensures a well-organized and seamless process from user request to job completion.

3.Tailor Interface:

Tailoring the interface in HandyConnect means customizing the user experience based on individual preferences and needs. The platform allows users to personalize their interactions, adjust settings, and receive tailored recommendations, enhancing overall usability and satisfaction.

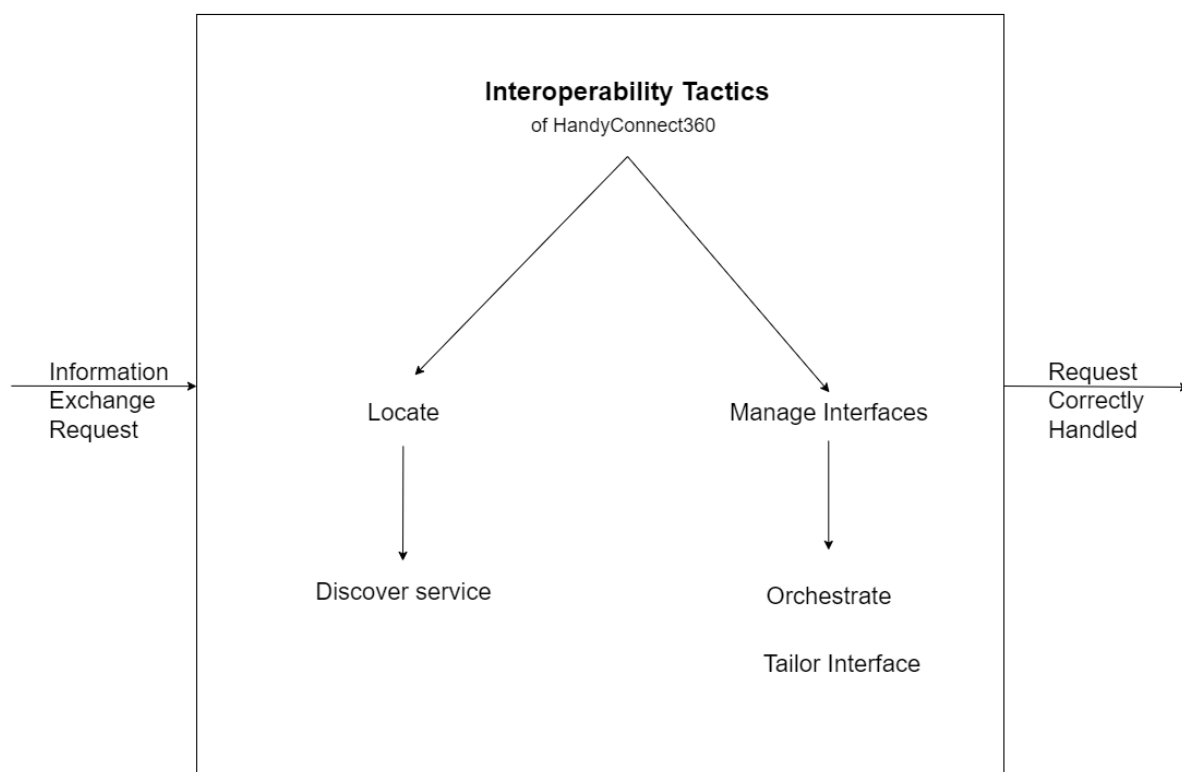


Figure 5:Interoperability Tactics

3.3.5 Security tactics:

Security in the context of software refers to the protection of data, systems, and users from unauthorized access, attacks, and potential threats, ensuring confidentiality, integrity, and availability.

Security tactics are strategies or measures implemented to safeguard a system. These may include encryption, access controls, authentication mechanisms, regular security audits, and other practices aimed at preventing, detecting, and responding to security breaches effectively. Some tactics are-

1.Detect Service Denial:

In the context of service platforms, detecting service denial involves actively monitoring for disruptions or intentional blockages in service availability, ensuring users can access and utilize the offered services effectively.

2.Detect Instruction:

Implementing mechanisms to accurately detect and interpret user instructions is crucial for seamless user interactions. This includes recognizing user commands, preferences, and service requests, ensuring precise understanding and appropriate system responses.

3.Identify Actors:

Recognizing and authenticating various entities interacting with the system, such as users and service providers, is essential for security. Robust identification processes enhance system integrity and contribute to accurate tracking of user activities.

4.Limit Access:

Security measures are employed to restrict system entry to authorized entities. Limiting access ensures that only authenticated users and service providers can access specific functionalities and information within the platform, safeguarding sensitive data.

5.Revoke Access:

The ability to promptly revoke access privileges is crucial for security management. If a user or service provider poses a security risk or violates platform policies, access privileges are revoked to prevent unauthorized usage and maintain overall system security.

6.Maintain Audit Trail:

Maintaining a comprehensive audit trail involves recording key activities and events within the system. This tactic ensures accountability, facilitates forensic analysis in case of security incidents, and aids in monitoring and enhancing overall system security.

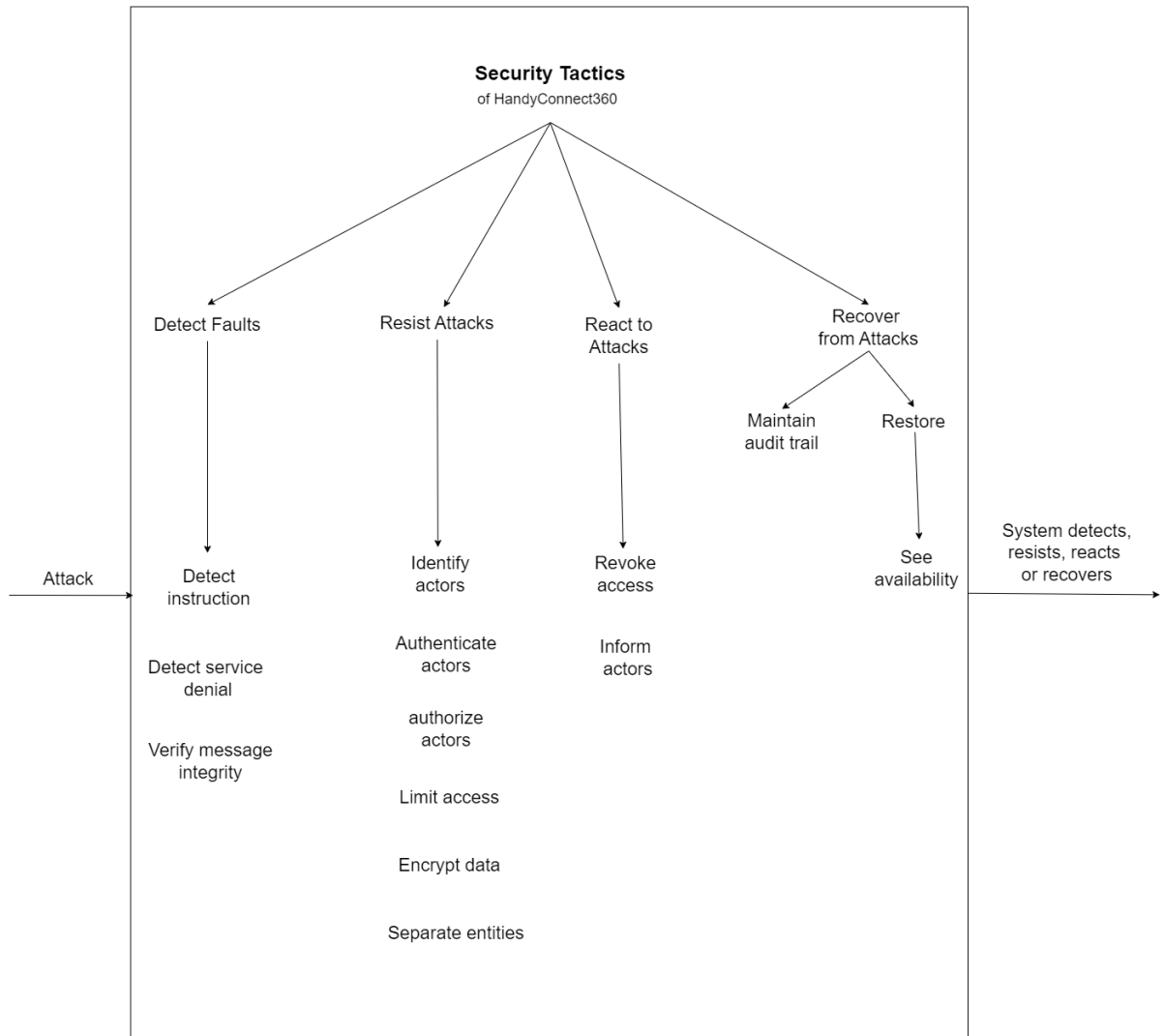


Figure 6:Security Tactics

3.3.6 Maintainability Tactics

Maintainability in software refers to the ease with which a system can be maintained, updated, and modified. A maintainable system allows for efficient troubleshooting, bug fixing, and implementation of new features without causing significant disruptions.

Maintainability tactics are strategies employed to enhance the ease of maintaining a software system. These may include modular design, clear documentation, version control, and practices such as code refactoring. The goal is to minimize the effort required for ongoing maintenance, making it more straightforward for developers to manage and improve the system over time.

1.Modular Design:

Implementing a modular design in the system involves organizing HandyConnect's components into discrete and independent modules. This enhances maintainability by allowing developers to focus on specific functionalities without affecting the entire system, facilitating easier updates and modifications.

2.Clear Architecture:

Maintaining a clear and well-defined architecture in HandyConnect ensures that the system's structure is easily understandable. A clear architecture aids developers in navigating the codebase, facilitating efficient troubleshooting, bug fixing, and the implementation of new features.

3.Track Change of Architecture:

HandyConnect monitors and tracks changes to its architecture over time. This involves documenting modifications and updates made to the system's structural design, enabling developers to understand the **evolution of the platform and make informed decisions during maintenance.**

4.Loose Coupling:

HandyConnect employs loose coupling between its modules and components, reducing dependencies. This design tactic enhances maintainability by allowing modifications to one part of the system without affecting others. Loose coupling makes it easier to update and enhance specific features without causing widespread disruptions.

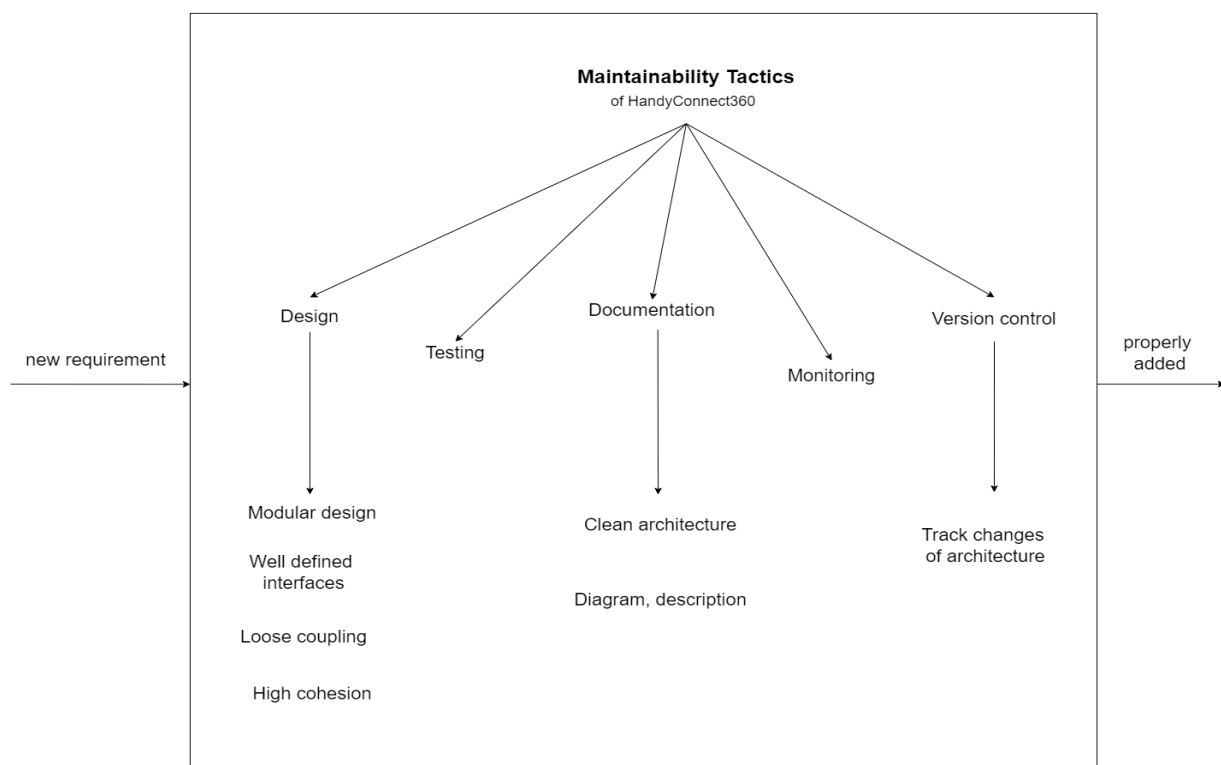


Figure 7:Maintainability Tactics

3.3.7 Scalability tactics:

Scalability in software refers to a system's ability to handle an increasing workload or growing user base without sacrificing performance. A scalable system can efficiently accommodate higher demands and resource requirements as the load expands.

Scalability tactics are strategies implemented to ensure a system can effectively scale to meet growing demands. These may include horizontal scaling (adding more instances or nodes), vertical scaling (increasing resources on existing nodes), load balancing, and optimizing database performance. The goal

is to maintain or improve system performance as the workload increases, providing a seamless user experience. Some tactics are-

1.Serverless:

Implementing a serverless architecture in the system involves relying on cloud services for computing resources instead of managing traditional servers. For HandyConnect, this means leveraging cloud functions and services to handle specific functionalities, promoting efficient resource utilization and scalability without the need for constant server management.

2.Distributed System:

HandyConnect utilizes a distributed system architecture, where tasks and data are spread across multiple interconnected servers. This promotes enhanced performance, fault tolerance, and scalability, allowing the platform to efficiently handle a large number of concurrent users and requests.

3.Load Balancing:

To optimize resource usage and ensure even distribution of incoming requests, HandyConnect employs load balancing techniques. This tactic involves distributing incoming traffic across multiple servers, preventing overloads on specific nodes and enhancing overall system performance and responsiveness.

4.Scaling:

HandyConnect incorporates scaling mechanisms to adapt to varying workloads. This can involve horizontal scaling by adding more servers or vertical scaling by increasing resources on existing servers. Scaling ensures the platform can handle increased demand, providing a responsive and reliable service.

5.Caching:

Utilizing caching strategies in HandyConnect involves storing frequently accessed data in a temporary storage layer. This enhances performance by reducing the need to fetch data from the database repeatedly. Caching contributes to faster response times, especially for commonly requested information, and supports the overall scalability of the system.

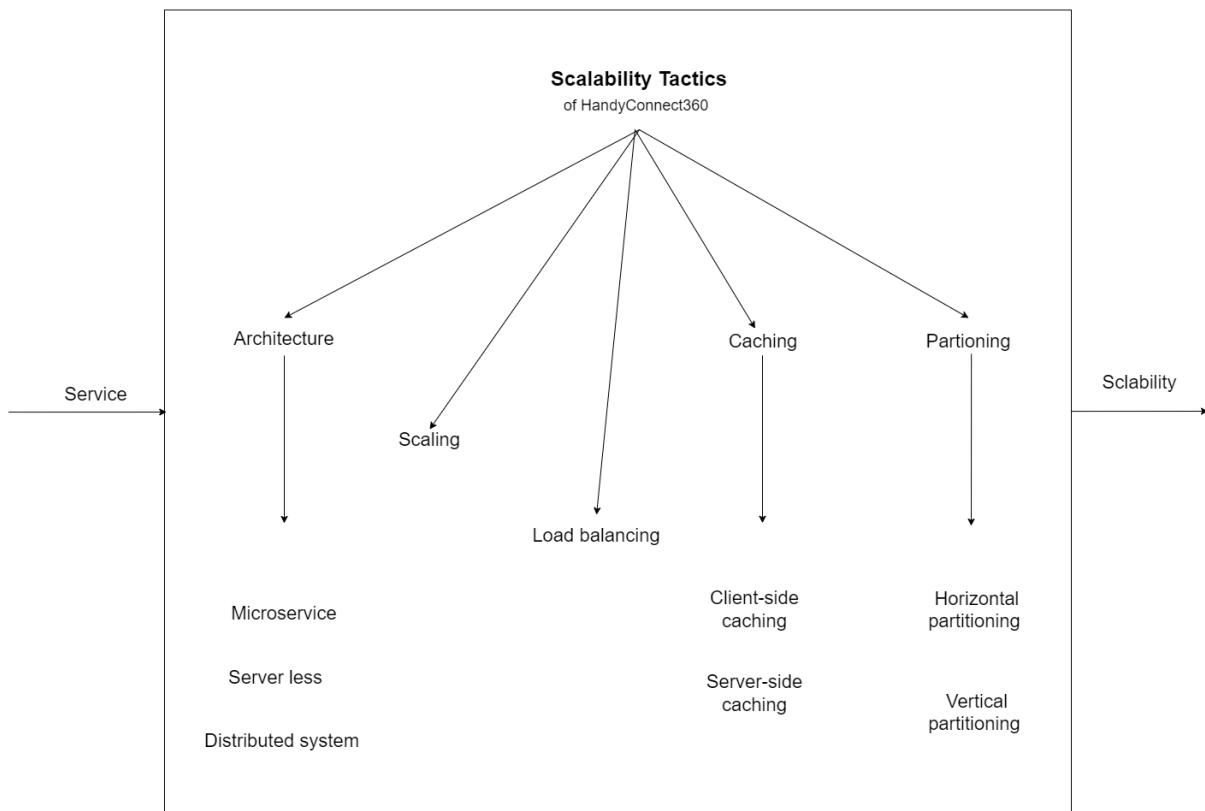


Figure 8: Scalability Tactics

3.3.8 Adaptability Tactics

Adaptability in software refers to a system's capability to evolve and adjust to changing requirements, technologies, or environments. An adaptable system can efficiently incorporate new features, technologies, or business rules without significant disruptions.

Adaptability tactics are strategies implemented to enhance a system's flexibility and responsiveness to change. These may include using modular and loosely coupled architectures, implementing feature toggles for controlled rollouts, and employing version control. The goal is to make the system agile and easily adaptable, allowing it to evolve to meet evolving needs efficiently. Some tactics are-

1. Monitor Internal System:

Implementing a system to monitor internal processes and performance in HandyConnect involves continuous tracking and analysis of the platform's health. This ensures early detection of potential issues, enabling proactive maintenance and optimization to keep the system running smoothly.

2. Track External Market:

HandyConnect tracks the external market to stay informed about industry trends, user preferences, and emerging competitors. This tactic allows the platform to adapt and incorporate features that align with market demands, enhancing its competitiveness and relevance.

3. Maintain Modular Architecture:

Ensuring a modular architecture in HandyConnect involves organizing the system into independent and interchangeable modules. This design tactic facilitates easier updates, replacements, and enhancements without causing widespread disruptions, contributing to the platform's adaptability and maintainability.

4. Monitor Emerging Technology:

HandyConnect proactively monitors emerging technologies relevant to its domain. This involves staying informed about new tools, frameworks, and innovations that can enhance the platform's features or performance. By integrating emerging technologies strategically, HandyConnect can stay at the forefront of service delivery and user experience.

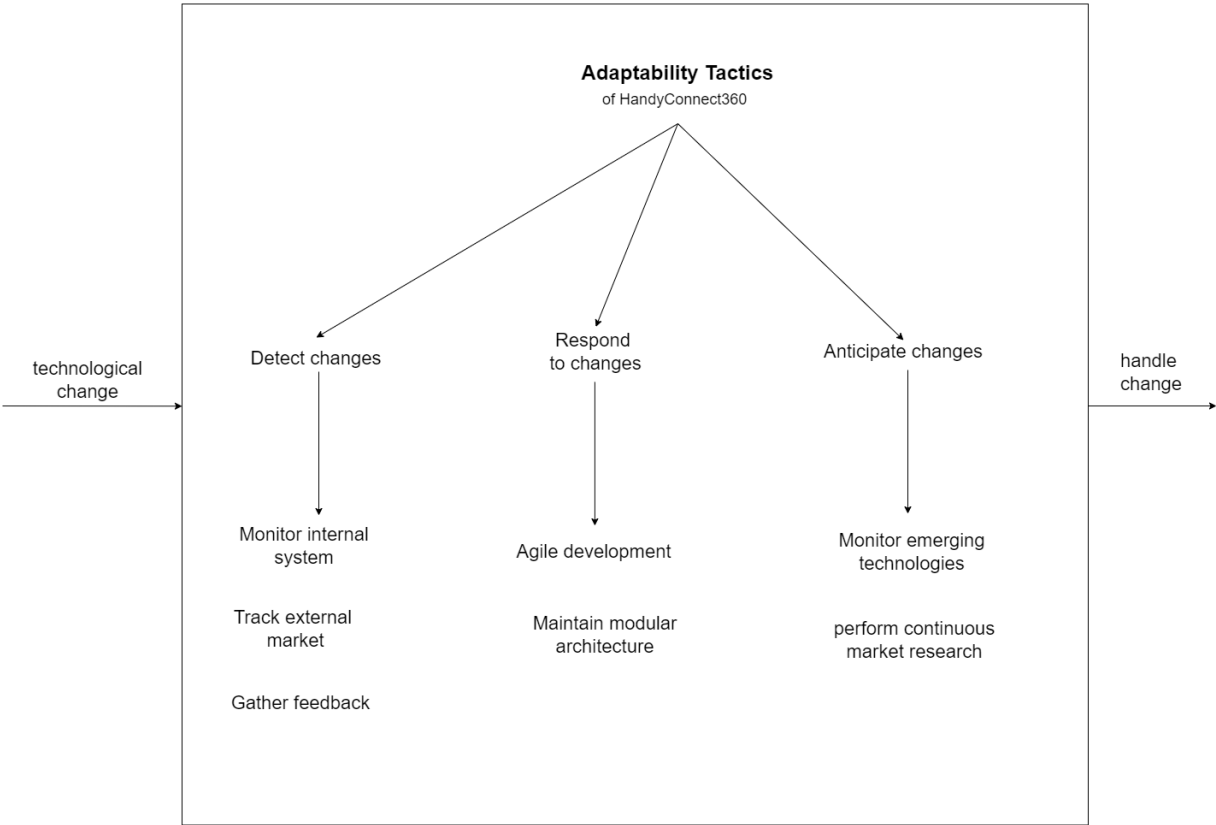


Figure 9: Adaptability Tactics

3.3.8 Testability Tactics

Testability in software refers to the ease with which a system or component can be tested. A highly testable system allows for efficient and comprehensive testing, ensuring the identification of defects, bugs, or inconsistencies during the development and maintenance phases.

Testability tactics are strategies employed to enhance a system's testability. These may include implementing modular and loosely coupled architectures, providing clear and comprehensive documentation, using automated testing tools, and incorporating logging mechanisms. The goal is to streamline the testing process, making it thorough, repeatable, and efficient, ultimately improving the overall quality of the software.

1, Specialized Interface:

In HandyConnect, employing a specialized interface involves tailoring user interactions for specific functionalities, ensuring a user-friendly and efficient experience. This tactic allows users to easily navigate and engage with features related to hiring handymen, optimizing the platform's usability.

2.Sandbox:

Utilizing a sandbox environment in HandyConnect creates a controlled and isolated space for testing and experimenting with new features or updates. This helps ensure that changes can be thoroughly evaluated without impacting the live platform, contributing to a more stable and reliable user experience.

3.Limit Structural Complexity:

HandyConnect implements measures to limit structural complexity, ensuring that the underlying system architecture remains straightforward and easy to understand. This tactic facilitates efficient maintenance and troubleshooting, contributing to the overall stability and reliability of the platform.

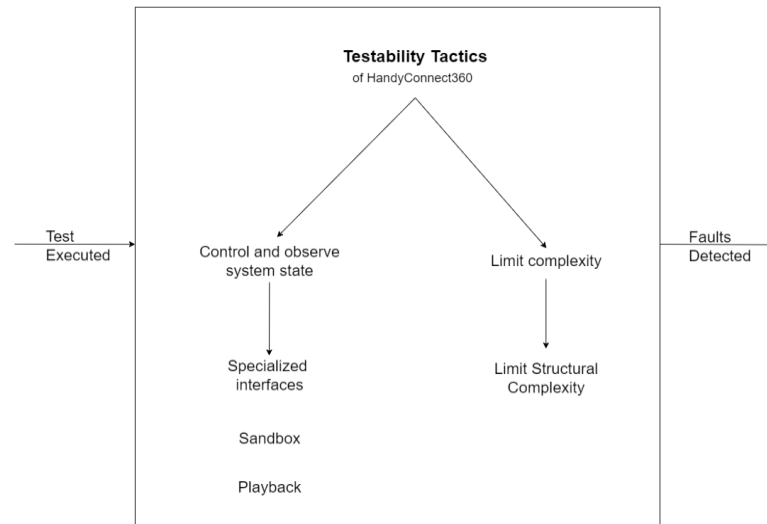


Figure 10:TestabiliTactics

4.

Architectural Representation

4.1 Context diagrams

Context diagrams focus on how external entities interact with your system.

In our HandyConnect360, the context diagram contains how our system interacts with its external entities like customer, service provider, payment gateway, database and other entities. The context diagram is-

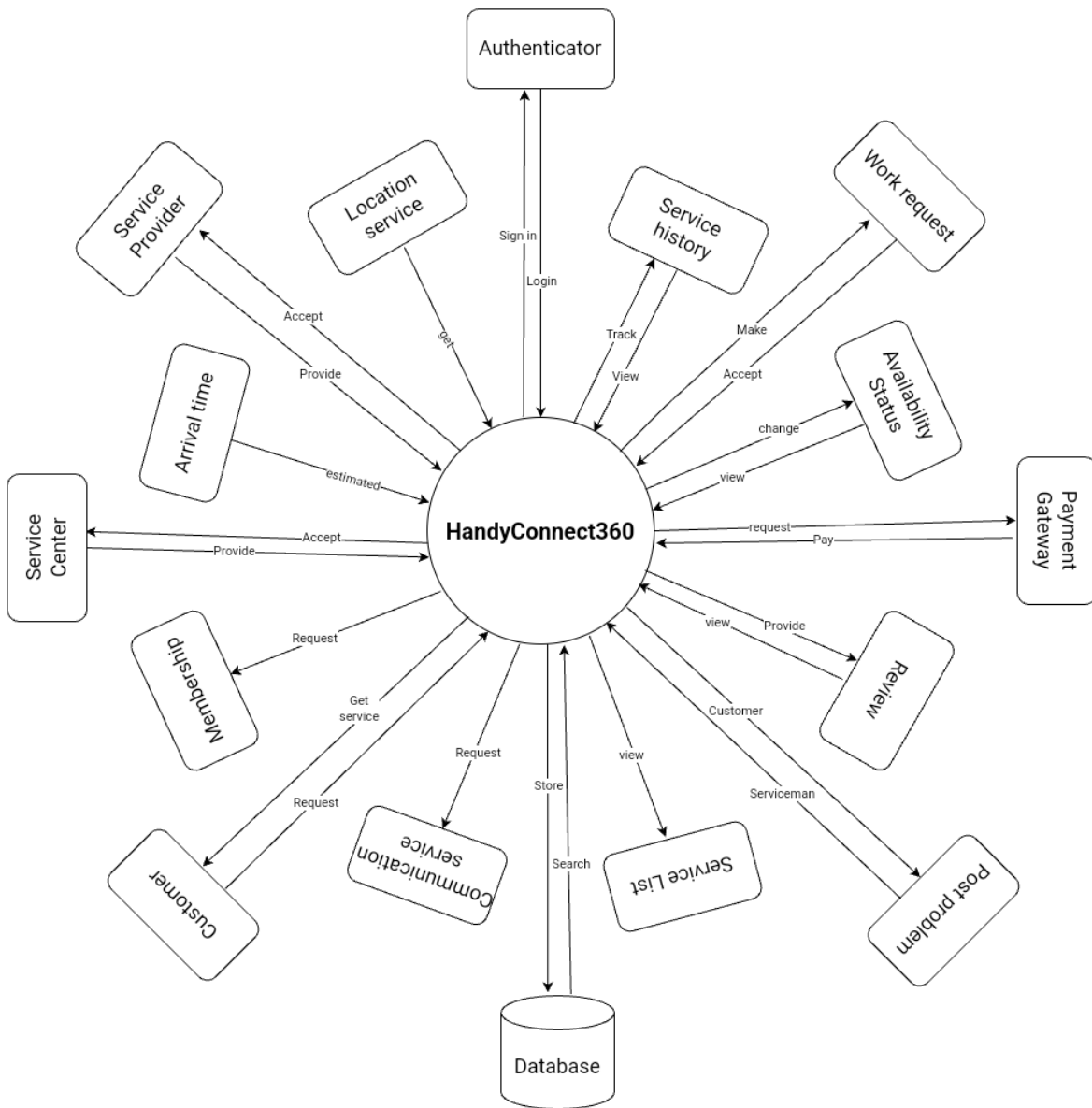


Figure 11:Context diagrams

4.2 Use-case Diagram

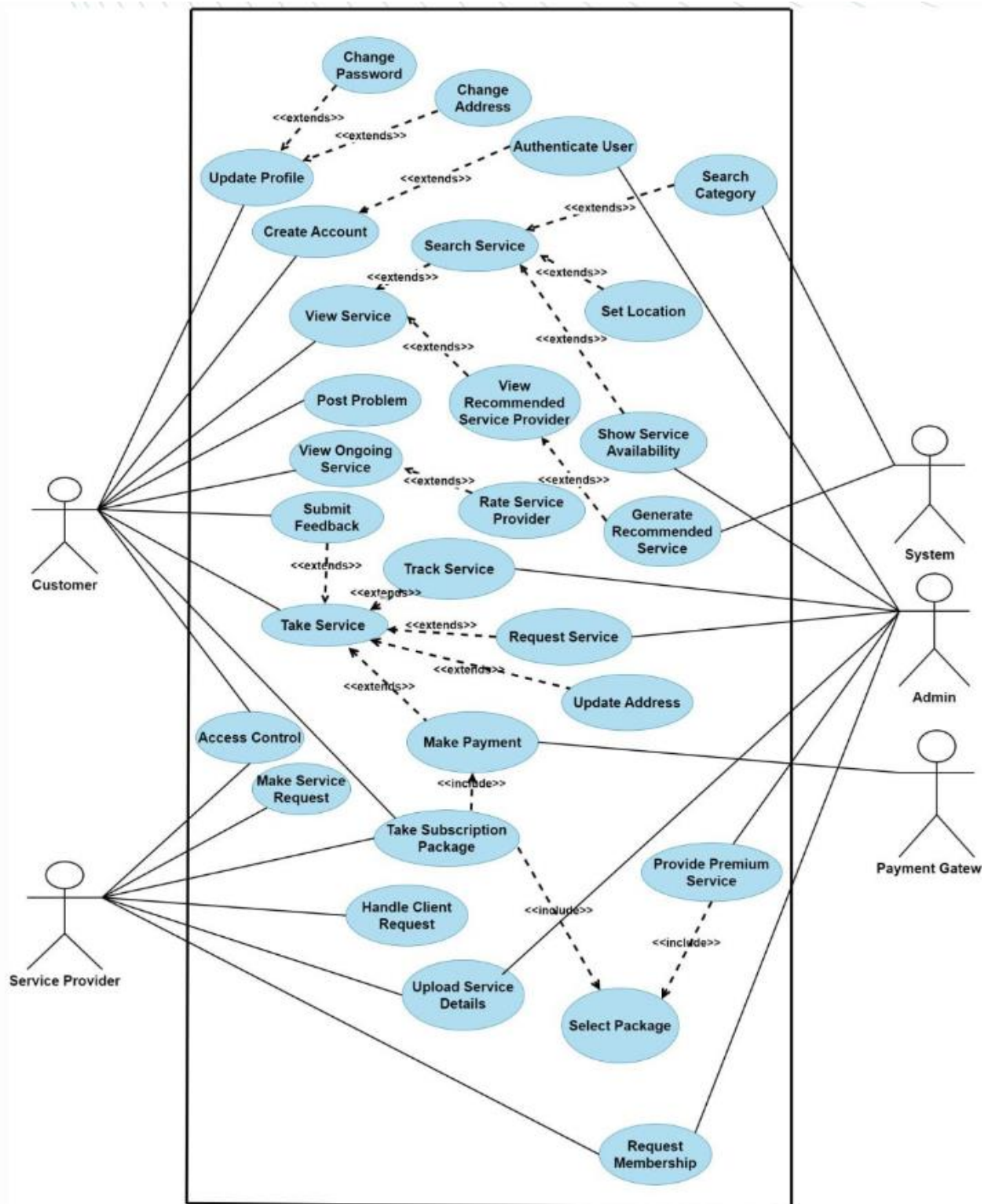


Figure 12:: Use-case diagram

4.2.1 Use Case Description

Search Service

Use Case	Search service	
Goal <a longer statement of the goal in context if needed>	The customer wants to search for a service.	
Preconditions <what we expect is already the state of the world>	Customer must log into the service	
Success End Condition <the state of the world upon successful completion>	The customer successfully selects a service.	
Failed End Condition <the state of the world if goal abandoned>	The customer is unable to find the desired service.	
Primary Actors:	Customer	
Secondary Actors:	N/A	
Trigger <the action upon the system that starts use case>	The "Search box" is clicked.	
Main Success Flows <the steps of the scenario from trigger to goal delivery and any clean up after>	Step	Action
	1	The customer clicks the search box.
	2	The customer provides a search keyword.
	3	Customer's desired category is shown.
Alternative Flows <a: condition causing branching> <a1: action or name of sub use case>	Step	Branching Action
	3a	The customer finds different result.
	3a1	The customer refines the search with a proper keyword.
Quality Requirements	Step	Requirement
	1	Search results will be within 5 seconds.

Table1:Search Service

Post Problem







Use Case	Post Problem	
Goal	Enable the customer to report and describe a problem or issue.	
Preconditions	The customer must be logged into the service platform.	
Success End Condition	The customer successfully submits the problem, and it is recorded for resolution.	
Failed End Condition	The customer encounters issues while submitting the problem.	
Primary Actors	Customer	
Secondary Actors	N/A	
Trigger	The customer encounters a problem and decides to report it.	
Main Success Flows	Step	Action
	1	The customer logs into the service platform.
	2	The customer navigates to the "Post Problem" section.
	3	The customer describes the problem in detail.
	4	The customer submits the problem report.
Alternative Flows	Step	Branching Action
	3a	The customer encounters difficulty in describing the problem.
	3a1	The customer seeks additional guidance or provides simpler details.
	4a	The problem submission fails.
	4a1	The customer retries the submission.
Quality Requirements	Step	Requirement
	4	Successful submission of the problem report will be 3 second

Table2: Post Problem

4.3 Logical View

The Logical View of the HandyConnect360 system provides an abstraction of the system's functionality from a software architecture perspective. It focuses on the organization of software components, their interactions, and the logical relationships between them, without delving into specific implementation details like programming languages or technologies. Here's a detailed breakdown of the Logical View:

4.3.1 Entities

-  **User:** Represents the platform's user base.
-  **Service Provider:** Represents service providers registered on the platform.
-  **Service Category:** Represents various categories of home maintenance services.
-  **Service Request:** Represents a user's request for a specific service.
-  **Review:** Represents a user's feedback on a service provider.
-  **Payment:** Represents a completed transaction for a service.

4.3.2 Relationships

User:

- Has one user profile.
- Can create multiple service requests.
- Can write reviews for service providers.
- Can make payments for services.

Service Provider:

- Has one service provider profile.
- Can be associated with multiple service categories.
- Can fulfill service requests.
- Can receive reviews from users.
- Can receive payments for services.

Service Category:

- Can have multiple associated service providers.
- Can be associated with multiple service requests.

Service Request:

- Belongs to one user.
- Belongs to one service category.
- Can be assigned to one service provider.
- Can have multiple reviews.
- Can have one payment.

Review:

- Is written by one user.
- Is about one service provider.
- Belongs to one service request.

Payment:

- Is made by one user.

- Is for one service request.
- Is received by one service provider.

4.3.3 Attributes

- **User:** Username, password, email address, phone number, address, profile picture, service preferences.
- **Service Provider:** Name, contact information, profile description, service categories offered, ratings, reviews.
- **Service Category:** Category name, description, associated service providers.
- **Service Request:** Description of service needed, location, date and time, assigned service provider, status, reviews, payment details.
- **Review:** Content, rating, date and time.
- **Payment:** Amount, date and time, payment method, payment status.

4.3.4 Constraints

- ✚ A user can only have one active service request at a time.
- ✚ A service provider can only fulfill one service request at a time.
- ✚ A service request can only be assigned to one service provider.
- ✚ A review can only be written for a service request that the user has made.
- ✚ A payment can only be made for a completed service request .

Logical View Diagram

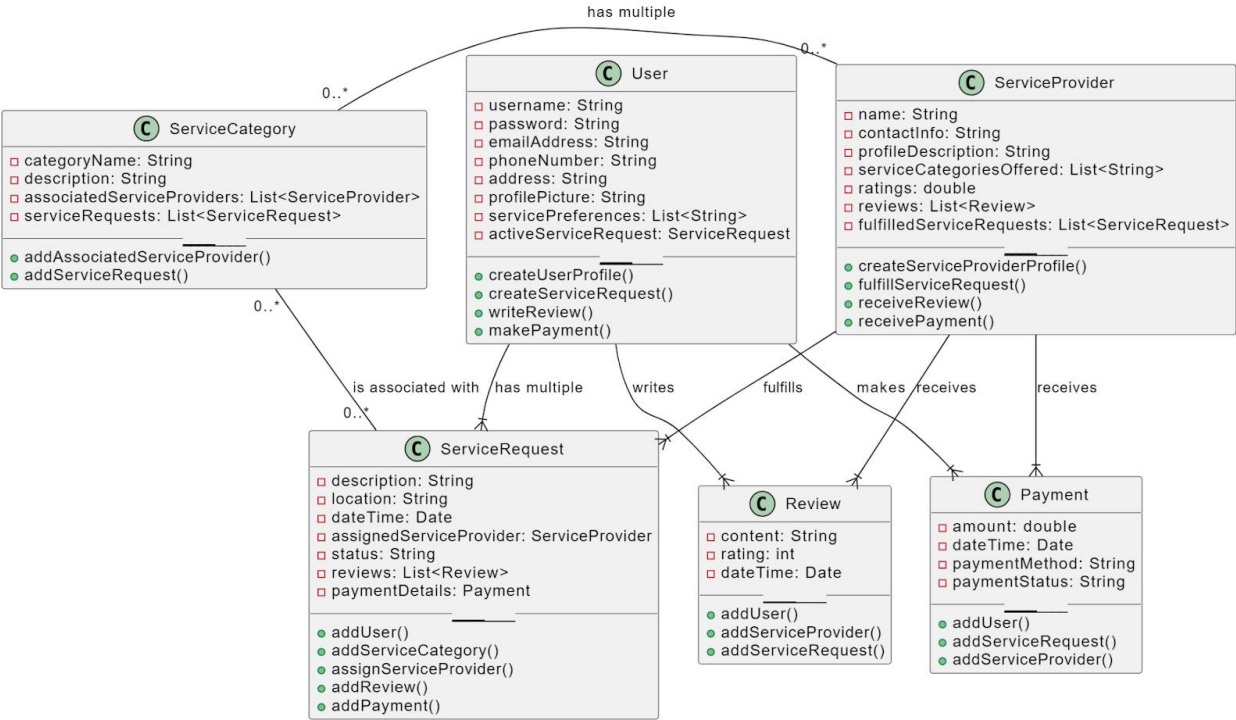


Figure 13: Logical View of HandyConnect360

4.4 Process View

The Process View of Handy Connect 360 provides a detailed overview of the system's operational processes, including how users interact with the platform to hire handymen, manage service requests, and handle transactions. Here's a breakdown of the key processes within the system :

4.4.1 Search Serviceman

Search Algorithm: Matches customer criteria with available handymen profiles based on skills, experience, ratings, and location.

Results Display: Presents a filtered list of handymen with relevant information like name, profile picture, average rating, distance, and service offerings.

4.4.2 Find Info of Serviceman

Profile Information: Displays bio, experience, skills, service rates, certifications, availability, calendar, customer reviews, and photos of past work.

Contact Options: Allows customers to contact the handyman through messaging, phone call, and others.

4.4.3 Send Hire Request

Proposal Submission: Customers can send a personalized message detailing their specific needs and desired service date.

Price Negotiation: Optional feature for customers to propose a price or negotiate with the handyman.

Confirmation Request: Handymen receive a notification and can accept or decline the request.

4.4.4 Accept Hire Request

Review Agreement: Handyman reviews the customer's proposal and service details.

Confirmation & Scheduling: Upon acceptance, the system automatically sends confirmation emails to both parties.

Calendar Integration: Appointment integrates with both customer and handyman's calendars.

4.4.5 Cancellation Process

Either customer or handyman can initiate cancellation through the platform.

Reason Selection: System prompts for selection of cancellation reasons (e.g., change in schedule, unforeseen circumstances).

Confirmation & Notification: Both parties receive notification and cancellation reason.

4.4.6 Communication Channels

Push Notifications: Alert both parties about new requests, confirmations, cancellations, and messages.

In-App Chat: Enables real-time communication between customer and handyman for clarification and coordination.

4.4.7 Database

Customer Information: Stores user details like name, contact information, location, and service history.

Handyman Information: Houses details like skills, experience, certifications, ratings, service rates, and availability.

Service Requests: Tracks all hire requests, including details like service type, date/time, price, and status.

Reviews & Ratings: Records customer feedback on handyman performance and service quality.

profile picture: This is a process view architecture of handy connect 360 system. It is a web-app for hiring handy-man. Now describe briefly this architectural diagram.

Uploaded image preview: The process view architecture of Handy Connect 360 is a web-app for hiring handymen. The system has six processes:

Search serviceman: This process allows customers to search for handymen based on their criteria.

Find info of serviceman: This process provides customers with more information about a particular handyman, such as their experience, ratings, and reviews.

Send hire request: This process allows customers to send a hire request to a handyman.

Accept hire request: This process allows handymen to accept or decline hire requests.

Cancellation process: This process allows customers to cancel hire requests and handymen to cancel accepted hire requests.

Transaction service: This process handles all financial transactions between customers and handymen.

The system also has a database and other communication channels. The database stores all of the data for the system, such as customer information, handyman information, and hire requests. The communication channels allow the system to send notifications to customers and handymen.

The following is a brief overview of the process view:

- ❖ A customer searches for a handyman using the search process.
- ❖ The customer finds a handyman that they are interested in and views their information using the find info of the serviceman process.
- ❖ The customer sends a hire request to the handyman using the send hire request process.
- ❖ The handyman receives the hire request and decides to accept or decline it using the accept hire request process.
- ❖ If the handyman accepts the hire request, the customer and handyman communicate with each other to schedule the service.
- ❖ After the service is complete, the customer makes a payment to the handyman using the transaction service.
- ❖ The customer can then leave a review for the handyman.

Process View Diagram

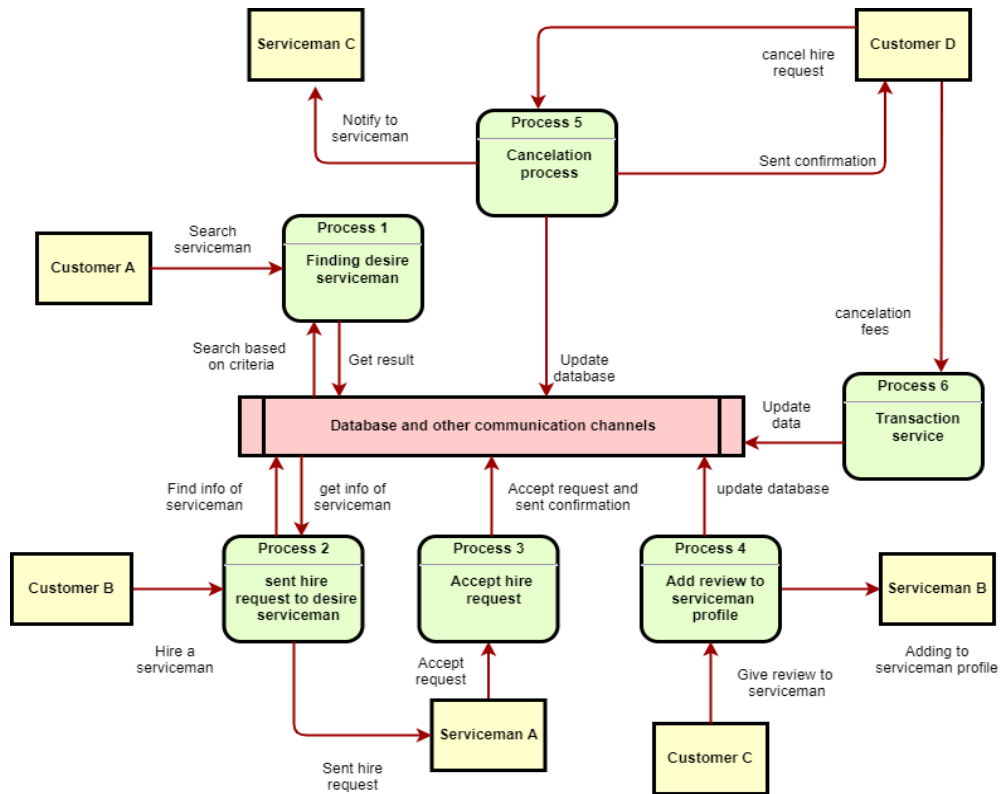


Figure 14:Process View of HandyConnect360

Sequence Diagram

The interaction flow between users and the application when searching for, evaluating, and rating handyman services:

User/Client Interaction:

- ❖ **Description:** The interaction begins with a user or client initiating the process within the HandyConnect360 Application.
- ❖ **Actor:** User/Client
- ❖ **Action:** Initiates the application and performs actions within the system.

Search Service:

- ❖ **Description:** Users search for handyman services within the application, allowing them to find relevant service providers.

- ❖ **Actor:** User/Client
- ❖ **Action:** Initiates a search for handyman services based on specified criteria.

Service Provider/Handyman Details:

- ❖ **Description:** After conducting a search, the application displays information about available handymen. Users can review details such as ratings, services offered, and contact information.
- ❖ **Actor:** HandyConnect360 Application
- ❖ **Action:** Retrieves and presents details of service providers or handymen matching the user's search criteria.

Rate Service Provider/Handyman Service:

- ❖ **Description:** Users have the option to rate their experience with a specific service provider or handyman. This step is crucial for providing feedback and helping others make informed choices.
- ❖ **Actor:** User/Client
- ❖ **Action:** Selects and rates the service received from a specific service provider or handyman.

Rate Service:

- ❖ **Description:** Users input their rating for the service received. Whether it's a positive or negative experience, this feedback contributes to the overall quality of the platform.
- ❖ **Actor:** User/Client
- ❖ **Action:** Inputs rating for the service received from the selected service provider or handyman.

Rating Confirmation:

- ❖ **Description:** After successfully rating a service provider, users receive confirmation. This ensures that their feedback has been recorded.
- ❖ **Actor:** HandyConnect360 Application
- ❖ **Action:** Confirms the successful recording of the user's rating for the service provider or handyman.

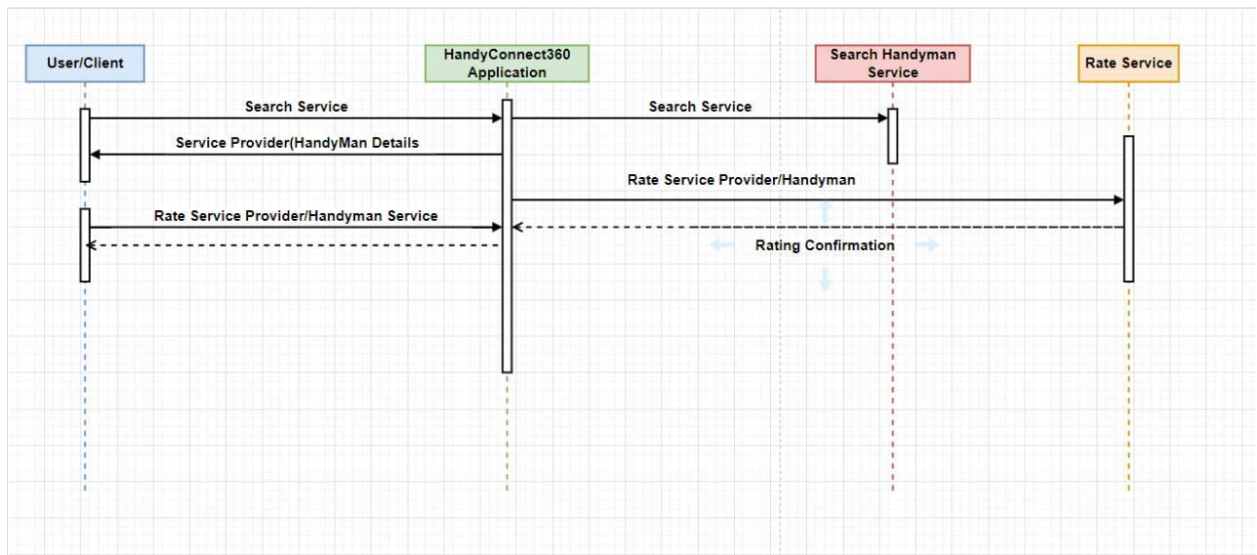


Figure 15:Sequence Diagram of HandyConnect360

4.5 Deployment View

The Deployment View of HandyConnect360 outlines how the various components of the system are deployed and how they interact with each other to deliver the functionality of the application. Here's a detailed explanation of the deployment view of HandyConnect360 :

4.5.1 Client Devices:

- ❖ **Description:** Two primary types of clients are depicted: Customers & Service provider(handyman). Their devices serve as the interface for users to access the HandyConnect360 application.
- ❖ **Operating Systems:** Supported operating systems include Windows 10 and Linux for personal computers, while mobile devices can operate on Android or iOS platforms.
- ❖ **Browsers:** A variety of web browsers, such as Microsoft Edge, Edge, Safari, and Chrome for personal computers, and Safari for mobile devices, are utilized to access the application.
- ❖ **Data Management:** Client devices employ mechanisms such as cookies and local storage for managing user-specific data.

4.5.2 Web Servers:

- ❖ **Description:** Web servers, represented are responsible for handling incoming web requests from client devices.
- ❖ **Components:** These servers host web applications comprising HTML5, JavaScript files, and the HandyConnect360 logo.
- ❖ **Operating Systems:** Web servers support diverse operating systems to ensure compatibility with different browser environments.
- ❖ **Functionality:** They facilitate the delivery of web pages, process user requests, and interface with backend systems for data retrieval and processing.

4.5.3 C-Web Servers:

- ❖ **Description:** Multiple instances of web applications denoted as CApplication1, CApplication2, and CApplication3.
- ❖ **Configurations:** Each application possesses distinct configurations, including file structures, configuration files (e.g., XML format, db.ini), and caching servers (<<CProcessor1>>, <<CProcessor2>>, <<CProcessor3>>) aimed at enhancing data processing efficiency.
- ❖ **Functionality:** These servers host various components of the HandyConnect360 application, encompassing user interfaces, business logic, and data processing modules.

4.5.4 Database Servers:

- ❖ **Description:** Database servers refer to specific databases with their schemas, stored procedures, and backup procedures.
- ❖ **Functionality:** These servers store and manage data pertaining to users, service providers, service requests, reviews, and other application entities.
- ❖ **Interactions:** They interact with web servers and C-Web servers to facilitate data retrieval and storage, thereby supporting the application's functionalities.

4.5.5 Connections & Interactions:

- ❖ **Description:** Arrows delineate the flow of information between client devices, web servers, C-Web servers, and database servers.
- ❖ **Data Flow:** User requests from client devices are processed by web servers, which communicate with C-Web servers for business logic execution and database servers for data retrieval and storage.
- ❖ **System Collaboration:** The collaborative efforts of all components ensure seamless user experience and efficient data processing, reflecting the robust architecture of the HandyConnect360 application.

Deployment View Diagram

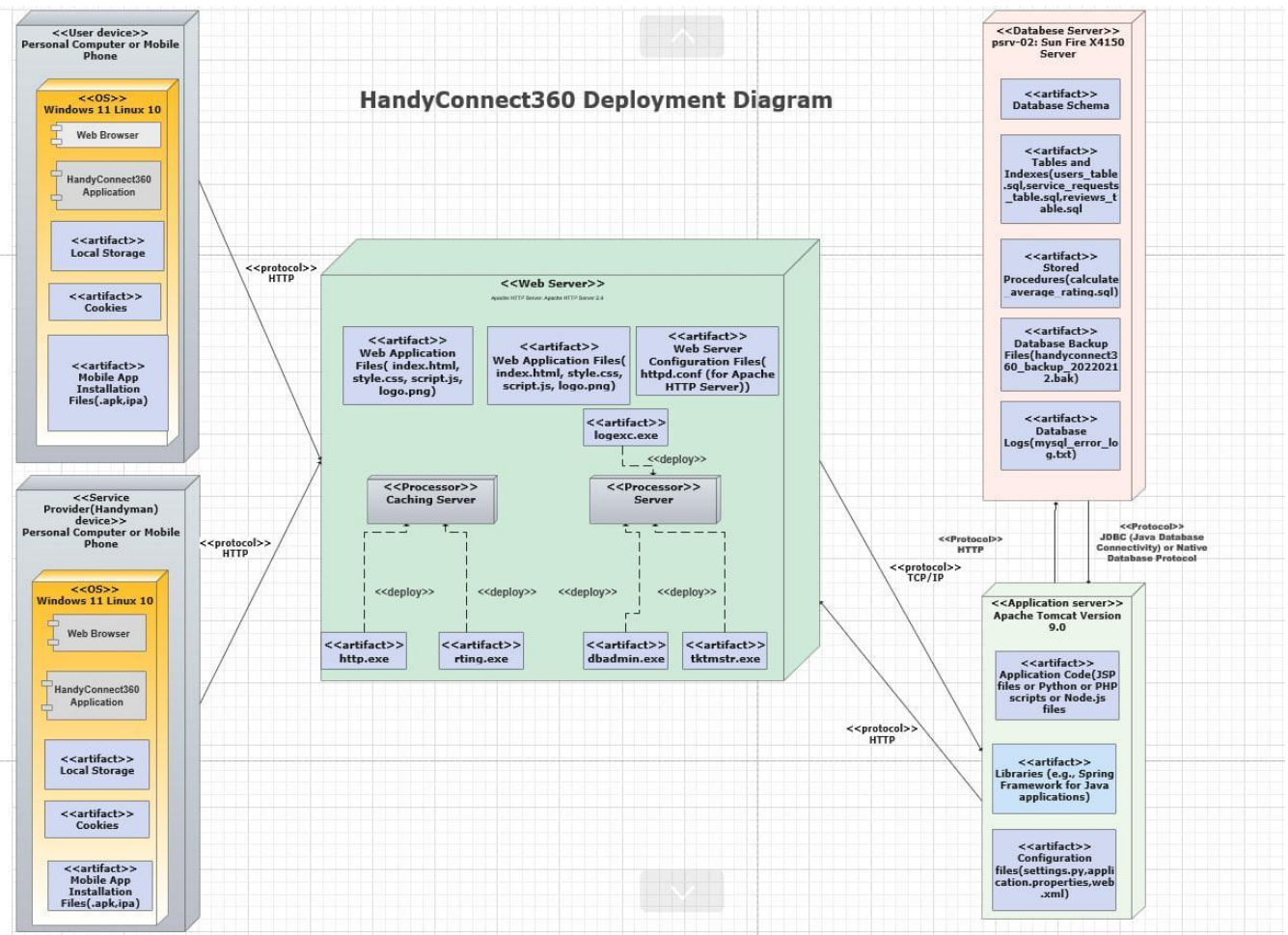


Figure 16:Deployment View of HandyConnect360

4.6 Implementation view

Implementation view or development view shows how development artifacts are organized in the file system.

In handyconnect360, The Implementation view reveals the intricate organization of development artifacts within the file system. It encompasses the structure of source code files, modules, libraries, and other essential components, facilitating effective collaboration, version control, and ongoing maintenance throughout the software development life cycle. This view provides developers with a comprehensive understanding of the codebase layout, promoting clarity and a systematic approach to building and evolving the software. The view is -

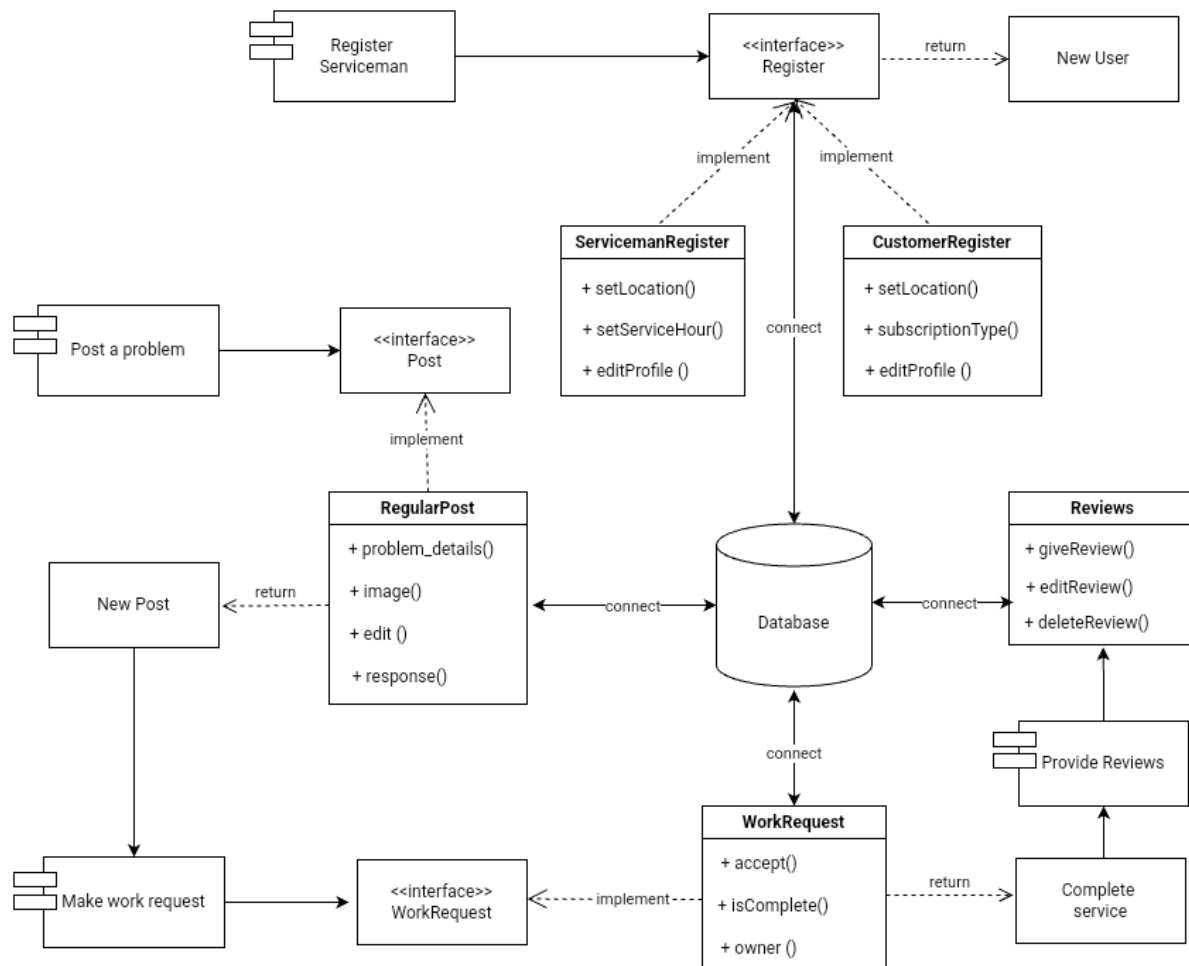


Figure 17:Implementation View of HandyConnect360

5. Architectural Goal and Constraints

HandyConnect360 aims to create a robust platform for connecting users with reliable service providers efficiently. To achieve this, several architectural goals and constraints have been identified:

🚦 Scalability:

Goal: Design a system architecture capable of scaling to accommodate a large user base and increasing service requests over time without sacrificing performance.

Constraint: Utilize distributed web servers and databases to ensure scalability, with the ability to handle up to 1 million concurrent users while maintaining responsiveness.

Security and Authentication:

Goal: Implement robust security measures to safeguard user data, prevent unauthorized access, and protect against common security threats.

Constraint: Employ strong encryption techniques, secure authentication mechanisms, and thorough security audits to ensure data integrity and user privacy.

Reliability and Availability:

Goal: Ensure high system availability and reliability to minimize downtime and service disruptions.

Constraint: Implement redundancy, failover mechanisms, and distributed architecture to enhance fault tolerance and ensure uninterrupted service availability.

Performance:

Goal: Optimize system performance to deliver fast response times for user requests and search functionality.

Constraint: Ensure that the system responds to user requests within 1 second and provides search results within 2 seconds to enhance user satisfaction and usability.

User Experience (UX):

Goal: Design an intuitive and responsive user interface that enhances user engagement and satisfaction.

Constraint: Prioritize user-centric design principles, usability testing, and feedback mechanisms to continually improve the user experience and drive user adoption.

Service Integration:

Goal: Seamlessly integrate with external services such as payment gateways and location services to enhance platform functionality.

Constraint: Implement robust APIs, communication protocols, and data exchange mechanisms to facilitate seamless integration with external service providers.

Data Management:

Goal: Ensure efficient storage, retrieval, and management of data to support platform functionality and performance.

Constraint: Implement data partitioning, indexing, caching strategies, and database optimization techniques to optimize data storage and retrieval processes.

Development Tools and Team Structure:

Goal: Foster a collaborative development environment and streamline the software development process.

Constraint: Utilize modern development tools, version control systems, and agile methodologies to facilitate effective collaboration, code management, and iterative development cycles.

Legacy Code and System Integration:

Goal: Integrate legacy systems and codebase seamlessly into the new architecture while maintaining system integrity and functionality.

Constraint: Conduct thorough legacy code analysis, refactor where necessary, and implement robust integration mechanisms to ensure compatibility and minimize disruption to existing services.

Schedule and Time Constraints:

Goal: Deliver the project within the specified timeline and budget constraints.

Constraint: Develop a detailed project plan, prioritize tasks, and allocate resources effectively to meet project milestones and deadlines.

Appendices

A. Design Processes

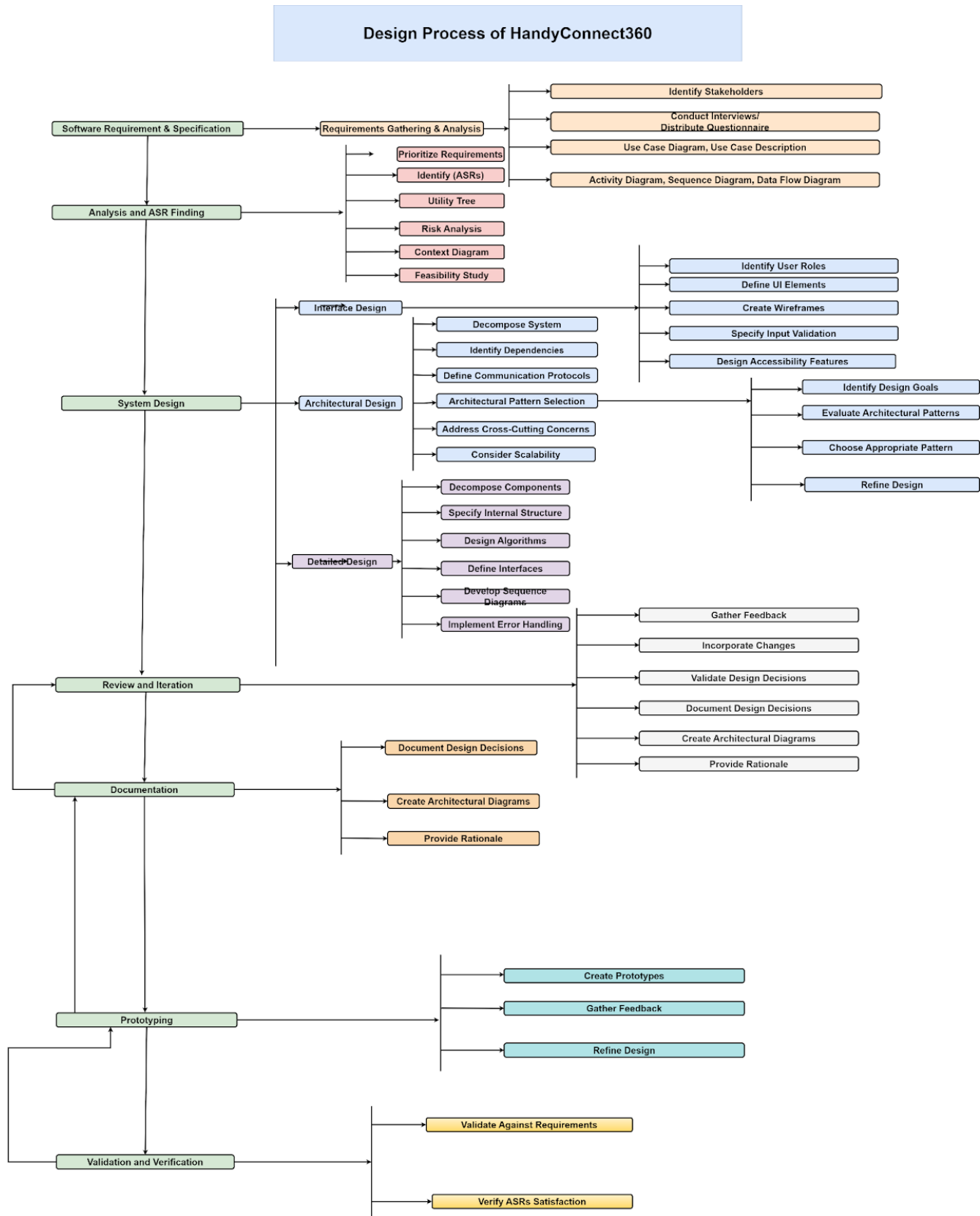


Figure 18:Design Process of HandyConnect360

The design process for HandyConnect360 involves multiple levels or phases. Let's break down each phase and outline the steps involved in designing the software architecture for HandyConnect360:

1. Requirements Gathering:

- **Identify Stakeholders:** Identify all stakeholders involved in the HandyConnect360 project, including users (customers), service providers (handymen), and system administrators.
- **Conduct Interviews:** Schedule and conduct interviews with stakeholders to understand their needs, pain points, and expectations from the platform.
- **Distribute Questionnaires:** Prepare and distribute questionnaires to gather quantitative data and feedback from a larger audience of potential users.
- **Create Use Cases:** Develop use cases to capture various scenarios and interactions that users and service providers will have with the platform.
- **Use Case Diagrams:** Create use case diagrams to visualize the interactions between actors (users, service providers) and the system.
- **Activity Diagrams:** Develop activity diagrams to represent the flow of activities and actions within the system from the perspective of different users

2. Analysis and ASR Finding:

- **Collect Requirements:** Gather all requirements identified from interviews, questionnaires, and use cases.
- **Prioritize Requirements:** Prioritize requirements based on their importance, urgency, and impact on the system and stakeholders.
- **Identify Architecturally Significant Requirements (ASRs):** Analyze requirements to identify those that have a significant impact on the system architecture and design.
- **Quality Attributes Analysis:** Identify quality attributes such as scalability, reliability, security, performance, and usability that are critical for the success of HandyConnect360.
- **Utility Tree:** Create a utility tree to hierarchically organize and prioritize requirements based on their utility to stakeholders.
- **Context Diagram:** Develop a context diagram to illustrate the external entities interacting with the system and the boundaries of the system.
- **Risk Analysis:** Perform risk analysis to identify potential risks and uncertainties associated with the project and its requirements.
- **Feasibility Study:** Conduct a feasibility study to assess the technical and economic feasibility of implementing various requirements and features.

3. System Design:

3.1 Interface Design:

Interface design focuses on specifying the interaction between the system and its environment. This phase deals with the external behavior of the system and how users interact with it.

Steps:

- **Identify Agents:** Identify the users (customers) and service providers (handyman) who will interact with the system.
- **Event Description:** Precisely describe the events or messages from users and service providers that the system must respond to.
- **Data Specification:** Specify the data formats and structures for input and output exchanged between the system and its users.
- **Timing Relationships:** Define the ordering and timing relationships between incoming events or messages and outgoing responses.

3.2 Architectural Design:

Architectural design involves defining the major components of the system, their responsibilities, interfaces, and interactions. This phase establishes the overall structure of the system.

Steps:

- **System Decomposition:** Decompose the system into major components based on functionality.
- **Responsibility Allocation:** Allocate functional responsibilities to each component.
- **Interface Design:** Define the interfaces between components for communication and interaction.
- **Performance and Reliability Considerations:** Address scalability, performance, and reliability requirements.
- **Component Interaction:** Specify how components communicate and interact with each other.

3.3 Detailed Design:

Detailed design involves specifying the internal elements of system components, including algorithms, data structures, and implementation details.

Steps:

- **Component Decomposition:** Further decompose major components into smaller program units or modules.
- **Responsibility Refinement:** Refine functional responsibilities and behaviors of each unit.
- **User Interface Design:** Design user interfaces for both web and mobile platforms.
- **Data and Control Interaction:** Specify how data and control flow between program units.
- **Algorithm and Data Structure Design:** Define algorithms and data structures for efficient processing.
- **Error Handling and Exception Handling:** Address error handling and exception handling mechanisms.

4. Architectural Pattern Selection

Choosing appropriate architectural patterns based on the requirements and design goals of HandyConnect360. Common patterns include Client-Server, Microservices, and Model-View-Controller (MVC).

5. Review and Iteration:

Throughout the design process, continuous review and iteration are essential. Stakeholders provide feedback, which is incorporated into the design. Design decisions are evaluated against ASRs to ensure they meet the required quality attributes.

6. Documentation:

Documenting the design decisions, architectural diagrams, and rationale behind them. This documentation serves as a reference for developers, testers, and future maintainers of the system.

7. Prototyping:

Create prototypes to validate design decisions and gather feedback from stakeholders. Prototyping helps in refining the user interface and functionality before full-scale development.

8. Validation and Verification:

Validating the design against requirements to ensure that it meets the expectations of users and stakeholders. Verification involves checking whether the design satisfies the specified ASRs and quality attributes.

B. ArchitecturePatterns

A software architectural pattern is a reusable solution to a commonly occurring problem in software architecture. These patterns help architects and developers design and structure software systems effectively by providing proven solutions to recurring design challenges. Architectural patterns define the organization of system components, the relationships between them, and the rules governing their interaction.

MVC (Model-View-Controller) Pattern

It can be applied to various components and features within the HandyConnect360 project:

Model Layer:

User Model: Represents user data including profiles, authentication details, and preferences. It interacts with the database to perform CRUD (Create, Read, Update, Delete) operations on user records.

Service Model: Manages service listings, categories, and details. It handles service-related operations such as adding new services, updating service information, and retrieving service details.

Schedule Model: Handles scheduling information such as appointments, availability of service providers, and booking details. It manages scheduling conflicts, availability status, and notifications.

Rating Model: Stores and retrieves user ratings and reviews for service providers and their services. It calculates average ratings, handles rating submissions, and displays rating information.

View Layer:

User Interface (UI): The UI components include web pages, mobile app screens, and user interfaces for various functionalities such as account creation, service browsing, scheduling, and communication.

HTML/CSS/JavaScript: Frontend technologies are used to design and implement the user interface, ensuring a visually appealing and interactive experience for users.

Templates and Views: Templates and view components render data from the model layer onto the user interface, presenting information in a structured and visually appealing manner. Views capture user interactions and input to be processed by controllers.

Controller Layer:

User Controller: Handles user authentication, registration, login/logout functionalities. It interacts with the user model to validate user credentials, create new user accounts, and manage user sessions.

Service Controller: Manages service-related operations such as searching for services, filtering services by category, and displaying service listings. It interacts with the service model to retrieve service information and handle service-related requests.

Schedule Controller: Facilitates scheduling functionalities such as booking appointments, managing schedules, and handling scheduling conflicts. It interacts with the schedule model to retrieve availability information, book appointments, and notify users of schedule updates.

Rating Controller: Manages user ratings and reviews for service providers and their services. It handles rating submissions, calculates average ratings, and displays rating information to users.

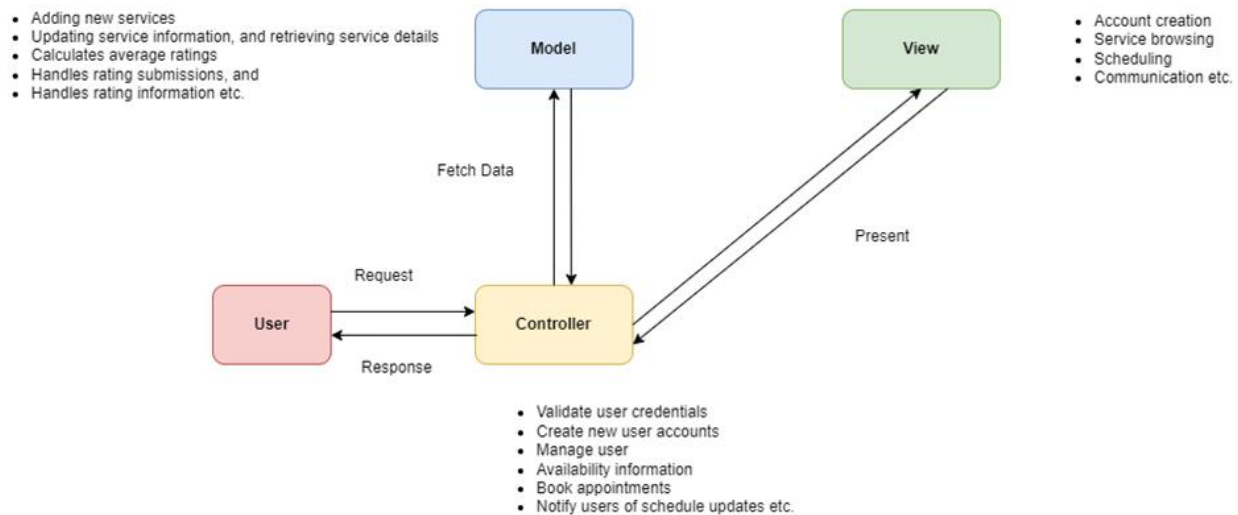


Figure 19:: MVC pattern of HandyConnect360

Benefits of using the MVC pattern in the HandyConnect360 project:

Separation of Concerns: The MVC pattern separates data, presentation, and control logic, making the application easier to maintain and extend.

Modularity: Each component (model, view, controller) can be developed and tested independently, promoting code reusability and scalability.

Testability: The separation of concerns allows for easier unit testing of individual components, ensuring the reliability and robustness of the application.

Enhanced User Experience: The MVC pattern facilitates the development of a user-friendly interface and ensures a consistent user experience across different functionalities and devices.

Different patterns can be used in different components as well. Here are some examples:

Client-Server Architecture

(Account Creation, Setting Location, Posting a Problem, Sent Work Request, Cancel Work Request):

In the context of HandyConnect360, the Client-Server Architecture enables users (clients) to interact with a central server. For account creation, users submit account information to the server, which validates and stores it securely. Similarly, users set their location by communicating with the server, which manages and stores location data. Posting a problem and sending work requests also involve users interacting with the server to submit information and initiate tasks. Canceling work requests follows a similar process, with users sending requests to the server to cancel previously initiated tasks. The server handles these requests, processes the data, and updates the system accordingly.

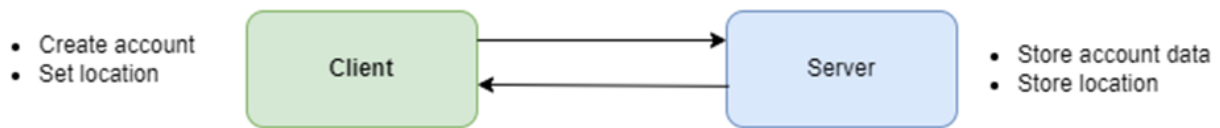


Figure 20:: Client-Server Architecture Pattern of HandyConnect360

Broker Pattern

(Customer Search a Service or Select Category)

The Broker pattern is suitable for service discovery and selection. In HandyConnect360, the broker component acts as an intermediary between users (clients) searching for services and service providers. Users submit their service requests or select categories to the broker, which then forwards these requests to appropriate service providers based on predefined criteria or categories. The broker facilitates efficient communication and matchmaking between users and service providers, helping users find relevant services quickly and effectively.

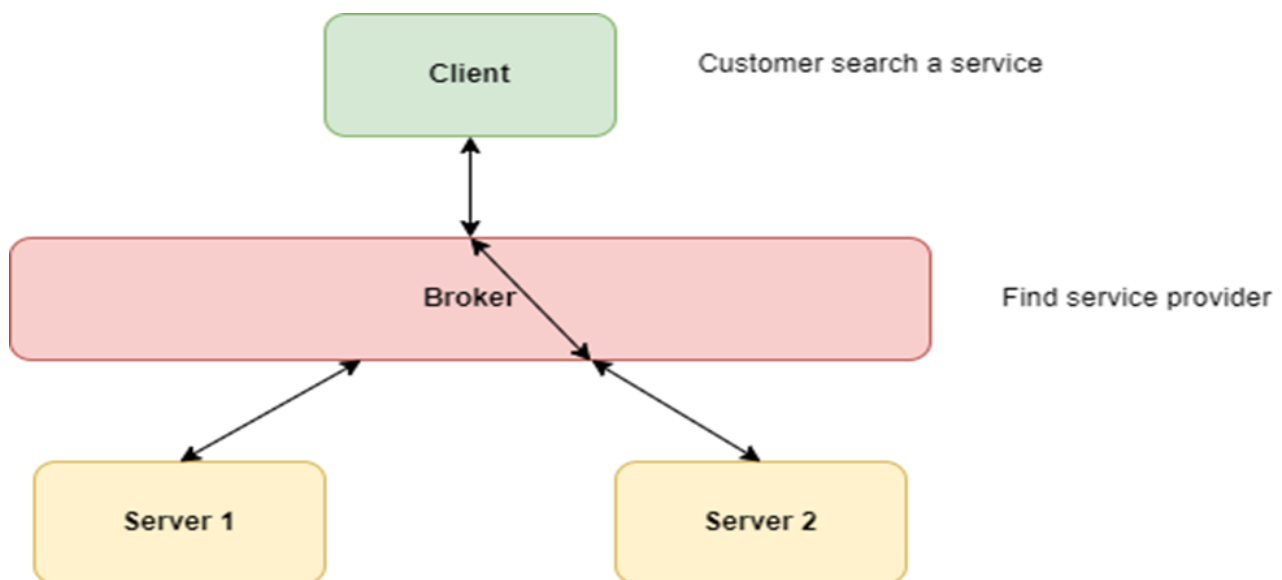


Figure 21:Broker Pattern of HandyConnect360

Pipe-Filter Pattern

(Emergency Post, Schedule Post):

The Pipe-Filter pattern is suitable for prioritising emergency posts and scheduling posts in HandyConnect360. In this pattern, posts or tasks flow through a series of filters, each performing specific actions or applying criteria. For emergency posts, the system can employ filters that prioritise urgent requests and ensure they are handled promptly. Similarly, scheduled posts can flow through filters that determine scheduling criteria, such as time or availability, and process the posts accordingly. The Pipe-Filter pattern provides flexibility and extensibility for managing different types of posts and tasks effectively.

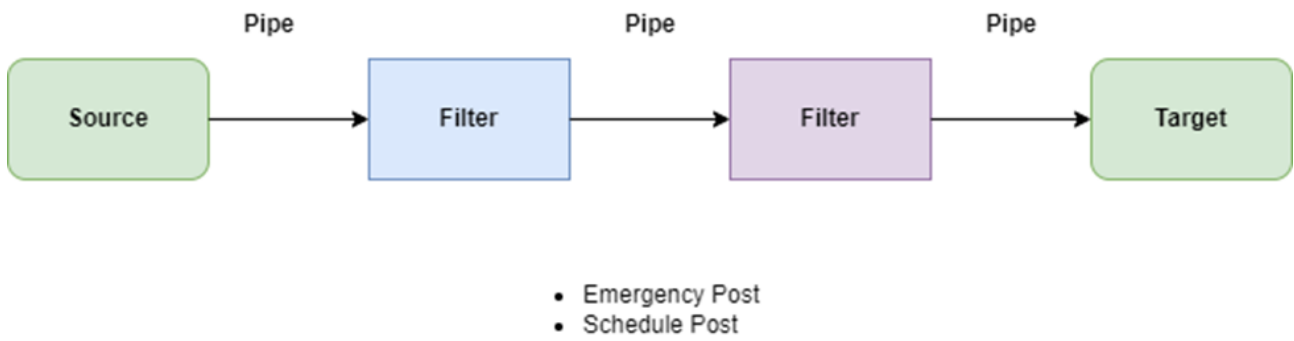


Figure 22:Pipe-Filter Pattern of HandyConnect360