|  |
| --- |
| Game\_Knight  System  We now present a study of using ADD 3.0 for a brownfield system based on a game database system. This study examines the initial design of the game database. The study is composed of three iterations and is tied to real-world circumstances. Initially the study presents the business case, and then a summary of the requirements for the database system. A summary of the activities that are performed during the ADD iterations is done iteration by iteration and steps are included within those. |

.

* 1. **Business Case**

A movie database corporation wanted to extend their reach into the game verse to achieve this they decided to investigate their competitors and decide what is necessary when building a system that focuses on games. As with their movie database they found it necessary to include an API that adds tuples of game related data in the respective tables. A poor API will greatly affect the performance of the system as the corporation intends to connect its API to its database program. The featured tables in the database include:

* Game (game\_id, rawg\_game\_id, fk\_dev\_id, name, release\_date, game\_img, description, metacritice\_score)
* Developers (dev\_id, rawg\_dev\_id, dev\_name, dev\_img)
* Genre (genre\_id, rawg\_genre\_id, genre\_name)
* Platform (plat\_id, rawg\_platform\_id, name)
* Genre\_Associated (ga\_id, fk\_game\_id, fk\_genre\_id)
* Has\_Tags (ht\_id, rawg\_game\_id, game\_id, tag)
* Played\_On (played\_on\_id, fk\_plat\_id, fk\_game\_id, release\_date)

To achieve the goals of the corporation, in making a game database relevant to the public

functions must be developed. The model system was named Game\_Knight and the functions include:

* *Sort function*. The purpose of the Sort function is to narrow the field of games the end-user views based on the prompt they have selected provided by a drop select list containing genre, rating, tag, platform, and developer name. the system is altered based on the prompt selected.
* *Expand and Collapse function*. The purpose of the Expand and Collapse function is to allow the end-user to view more information about the game they have selected and close that information when they decide to, returning to the home screen. The function is accompanied by a hover highlight to signify expandability.
* *Recommend Game function*. This function allows the end-user to receive a recommended game either based on a relation to another game, relation to a genre, or a random game.
* *Search function*. The search function allows the end-user to search a specific game title and provide a form with games either matching the search or games that are closely related. This function also allows the end-user to use all other functions within the form.
* *Display platform associated*. The function allows the user to view the platform associated to the game the end-user is viewing. This function also displays an icon when the game is in its collapse state.
* *Game stats analysis*. The function displays game information based on the prompt the user selects for example, games with the highest Metacritic score which shows a form of the top ten games with the highest Metacritic score.
* *Fault analysis function*. The goal of fault analysis function is to show, examine, and log faults that occur in the system. The system detects whether a function responds the way it is intended for example, the expand and collapse function within the system intends to show less or more information whether the function operates the server is expected to log this information.



* 1. **System Requirements**

Requirement elicitation activities had been examined, and the below is a representation of the most relevant requirements collected.

* + 1. **Use Case Model**

The use case model figure 1.1 represents the use cases that are essential in the production of the Game\_Knight system. Other use cases that were deemed irrelevant are not shown in this study



Diagram

Description automatically generated



**FIGURE 1.1** Use case model for the Game\_Knight system

Each of these cases are elaborated in the table below:

|  |  |
| --- | --- |
| **Use Case** | **Description** |
| UC-1: Monitor User Interaction | The system monitors the end user’s interactions in the Game\_Knight website. When the end user hovers in the vicinity of an interactable service provided by the systems it will provide a visual cue. The users can expand and collapse film directories and any related information. |
| UC-2: Detect function operability | The server manages faults in the operability of functions in the Game\_Knight system. The server detects whether a function responds the way it is intended for example, a collapse function within the system intends to show less information whether the function operates the server is expected to log this information |
| UC-3: Display recommended Game | Stored data within the Game\_Knight database to recommend to the end user a film based on games the user has interest in, has given a rating or simple requests related to games. |
| UC-4: Manage database operability | The administrator manages operability in the Game\_Knight server to, whether it be to remove or add functions to the server from, the database system. |
| UC-5 Configure Navigation | The administrator alters the configuration parameters assigned to the navigation within the system. |
| UC-6: Collect performance data | System performance data is collected from the system. |
| UC-7: Create statistical analysis of game data | Creates a chart analysis of statistics provided by the database such as number of games, number of games per genre in the user’s library. |
| UC-8: Display information | The system displays stored information acquired from the database to the end user. Various parameters are displayed based on the interaction received. |

* + 1. **Quality Attribute Scenarios**

Accompanying these cases, various quality attribute scenarios were documented. The six quality attributes are presented in the table below. For each tuple an id is presented alongside its ties with the use cases determined.

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Quality Attribute** | **Scenario** | **Associated Use Case** |
| QA-1 | Availability | The system database fails to display the information associated with the film. System will display an error message to the user. | UC-8 |
| QA-2 | Modifiability | A function is added to the database system. The new function is successfully added and works as intended. | UC-2 |
| QA-3 | Performance | The database system collects performance data while the user is operating the program. The performance data is collected and logged to monitor system fluidity. | UC-6 |
| QA-4 | Security | A user decides to alter his/her account information. The altered account information is stored within the system and updated for the user’s future system operation | All |
| QA-5 | Testability | Recommendations fail to display to the user because of insubstantial user input. Recommendations will display after the user is prompted for their input. | UC-3 |
| QA-6 | Usability | The user desire functions that make searching for a game more informative. The functions must operate as intended, and if not, an error is sent to the maintenance technician. | UC-4 |

* + 1. **Constraints**

Constraints on the Game\_Knight system and its implementation were collected and presented in the table below.

|  |  |
| --- | --- |
| **ID** | **Constraint** |
| CON-1 | A catalogue of games must already be downloaded to utilize the functions within the system |
| CON-2 | A pre-existing database must be used to fetch the info for the games |
| CON-3 | A stable internet connection must be available |
| CON-4 | Performance information must be collected and logged for a minimum of 30 days |

* + 1. **Architectural Concerns**

The architectural concerns of the system are listed in the table below for the brownfield system.

|  |  |
| --- | --- |
| **ID** | **Concern** |
| CRN-1 | The initial structure of the system will determine the usability of the system |
| CRN-2 | The background knowledge of the team is essential in determining whether the system can be built or not. SQL knowledge is necessary, and python is necessary in implementing the views for the statistical analysis function |
| CRN-3 | A proper framework for the architecture of the system must be selected |
| CRN-4 | Assign members to efficient places of development |

* 1. **The Design Process**

Now that system requirements were analysed the next step would be to move on to the design process. The design process is analysed using the ADD method. This decision will determine the architecture of the system.

* + 1. **ADD Step 1: Review Inputs**

The first step of the ADD method includes reviewing the inputs and analysing which requirements will be further developed into drivers. The table below summarizes the inputs with the category and details.

|  |  |
| --- | --- |
| **Category** | **Details** |
| Design purpose | This is a brownfield system, so the purpose is to update the design to a more efficient architecture using the ADD method to support the construction of the Game\_Knight system |
| Primary functional requirements | From the use cases presented in section 1.2.1 the primary functional requirements were determined as:   * UC-1: Monitor user interaction * UC-3: Display recommended film * UC-7: Create statistical analysis of game data |
| Quality attribute scenarios | Described in section 1.2.2 they are now examined by priority from Low-Medium-High |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Scenario ID** | **Importance to the Customer** | **Difficulty of Implementation According to the Architect** |
|  | QA-1 | High | Low |
|  | QA-2 | Medium | Medium |
|  | QA-3 | High | High |
|  | QA-4 | Low | High |
|  | QA-5 | Medium | Low |
|  | QA-6 | High | Low |
|  | From this list QA-1, QA-2, QA-3, and QA-6 are selected as drivers. | | |
| Constraints | All the constraints discussed in 1.2.3 are included | | |
| Architectural concerns | All the architectural concerns in 1.2.4 are included | | |
|  |  |  |  |

* + 1. **Iteration 1: Establishing an Overall System Structure**

In this section the results of the design process activities preformed in the steps of the ADD method first iteration are represented.

* + - 1. Step 2: Establish Iteration Goal by Selecting Drivers

This is the primary iteration of the design of a brownfield system Game\_Knight, the iteration goal is tied to the first architectural concern CRN-1 *The initial structure of the system will determine the usability of the system* producing an initial overall system structure.

Along with the concern the architect must factor in all the drivers that influence the structure of the Game\_Knight system. Below is a list of drivers the architecture will be required to be conscious of:

* QA-1: Availability
* QA-2: Modifiability
* QA-3: Performance
* QA-6: Usability
* CON-1: A catalogue of games must already be downloaded to utilize the functions within the system
* CON-2: A pre-existing database must be used to fetch the info for the games
* CON-3: A stable internet connection must be available
* CON-4: Performance information must be collected and logged for a minimum of 30 days
* CRN-2: The background knowledge of the team is essential in determining whether the system can be built or not. SQL knowledge is necessary, python in necessary in implementing the views for the statistical analysis function

|  |
| --- |
| Z |

**FIGURE 1.2** Context diagram for the Game\_Knight system

* + - 1. Step 3: Choose One or More Elements of the System to Refine

This is a brownfield system, so this system has a previous build which heavily relies on the MYSQL Workbench database. To refine the system, it is necessary to refine the input stream of the Game\_Knight system, which will be done using the search and transfer API.

* + - 1. Step 4: Choose One or More Design Concepts That Satisfy the Selected Drivers

In the primary iteration the goal of establishing an initial structure of the Game\_Knight system design concepts are introduced. The table below briefs the selection of design decisions.

|  |  |
| --- | --- |
| **Design Decisions and Location** | **Rationale** |
| Logically structure the client view of the system using a **Web application** | This reference architecture is used toward the development of applications that is typically initiated by the user from a web browser like google that communicates with a server using HTTP protocol. Since the bulk of the system resides in the MYSQL workbench database application |

|  |  |  |
| --- | --- | --- |
|  | **Discarded alternatives:** | |
|  | **Alternative** | **Reason for Discarding** |
|  | Rich Internet applications | RIA typically run inside a browser which is great for the system and may be developed using code that is executed by the browser like Asynchronous JavaScript or XML (AJAX). Though this reference architecture is not needed as a much simpler approach |
|  | Mobile applications | This reference architecture is like a web application in terms of linking the system to a device, but it is deployed to handheld devices. This reference architecture was not used due to the fact it was not the greatest option when considering tying the database server. |
|  | Rich Client applications | This reference architecture is used typically when want an application to be highly interactive and responsible and desire to leverage the user’s machine resources like a graphics card. This architecture wasn’t used because there is no need to be highly interactive and responsible and desire to leverage the user’s machine resources |
| Build the user functions using **python** and **Django**    Deploy the web application using html and **API** | The systems main functionality needs to be imbedded into the system using python functions will be built using the classes that are simplified from the database from the Django  Access to the application is obtained via a git hosted web browser, which launches the system  This technology makes it, so the end-user does not have to download any applications. | |
|  |  | |

**1.3.2.4** Step 5: Instantiate Architectural Elements, Allocate Responsibilities, and Define Interfaces

The instantiation design decisions are explained in the table below:

|  |  |
| --- | --- |
| **Design Decisions and Location** | **Rationale** |
| Web application to disconnect local sources of data | With a strong and stable connection, it is viable to not store any data locally. The data layer is where server communication is handled. Communication between the many client components are directed via local method calls without any other support. |
| Create a module dedicated to accessing the views pre-built in the database server. | The module will allow permission to view and close different iterations of the main page of the web application. Which has ties to QA-3 (performance), QA-6 (usability). This will further the development of the system’s views as many different variants of existing pages in the site will be constructable. |

What was examined from these instantiation results are elaborated on the next step of the iteration, though it is too vague in terms of functionality as this is the primary iteration. These results will be used in further iterations to make definitions more concise.

* + - 1. Step 6: Sketch Views and Record Design Decisions

The Figure 1.3 shows the module view of the reference architecture web application that is used in the client and server side.

Graphical user interface, diagram

Description automatically generated

**FIGURE 1.3**

For Figure 1.3 a table is presented below naming each element and providing a brief description of the responsibility the element holds. As this is the primary iteration these descriptions are rough and unfinished.

|  |  |
| --- | --- |
| **Element** | **Responsibility** |
| Presentation client side (CS) | This layer is used to show modules involved in the client side of the system |
| Business logic (CS) | This layer contains modules associated with the business functions in the system |
| Data (CS) | This layer discusses modules linked to the communication of the system |
| Cross cutting (CS) | This layer has modules involved in multiple layers of the client side of the system |
| Process module | This module is responsible for control flow in the system |
| Business modules (CS) | Show business functions from server side |
| Business entities (CS) | Make up the domain model |
| Communication module (CS) | Represents the communication on client side of the system |
| Services server side (SS) | Represents the services included on the server side of the system |
| Business logic (SS) | This layer contains modules associated with the business functions in the system |
| Data (SS) | This layer discusses modules linked to the communication of the system |
| Cross cutting (SS) | This layer has modules involved in multiple layers of the server side of the system |
| Service interface (SS) |  |
| Business modules (SS) | Implement functions |
| Business entities (SS) | Make up the domain model |
| DB Access module | This module is responsible for linking the database to the system |

The responsibilities of the elements concerned with the process of using the system are below:

|  |  |
| --- | --- |
| **Element** | **Responsibility** |
| User device | The user’s device that allows them to view the web application on a web browser |
| Web server | The server that host the application for the client-side view |
| Database server | The server in which the data for the tables are stored along with views of the table |
| API Application | The application that digs information of game data for the Game\_Knight system |

The initial deployment diagram is featured below:

Diagram

Description automatically generated

**FIGURE 1.4** Initial deployment diagram

* + - 1. Step 7: Perform Analysis of Current Design and Review Iteration Goal and Achievement of Design Purpose

A summary of the design process is explained in the table below

|  |  |  |  |
| --- | --- | --- | --- |
| **Not Addressed** | **Partially Addressed** | **Completely Addressed** | **Design Decisions Made During the Iteration** |
|  | UC-1 |  | The web application reference architecture make it simple to view the clients view of the systems. Flags are set to further elaborate on the systems performance. |
|  | UC-3 |  | The selected architecture defines the modules that will support this case. |
|  | UC-7 |  | The selected architecture defines the modules that will support this case. |
|  | QA-1 |  | The Availability of the system would be 24/7 as it is a website |
|  | QA-2 |  | The modifiability of the system should be flexible as many new functions will be added when the database is extended though as it is too early to make any final decisions. |
|  | QA-3 |  | The performance of the system is crucial functions must return a value whether it completes the operation or it does not. |
|  | QA-4 |  | The security of the website is not a priority as the user is saving their information locally |
| QA-5 |  |  | No Relevant decisions made |
|  | QA-6 |  | Data logging using the save tactic will aid in the usability attribute of the system. |
|  |  | CON-1 | A catalogue of games must already be downloaded to utilize the functions within the system this is taken care of as it is a brownfield system |
| CON-2 |  |  | No Relevant decisions made |
| CON-3 |  |  | No Relevant decisions made |
|  | CON-4 |  | Use of the save tactic to manage data logging will aid in this constraint as it is required in the greater implementation. |
|  | CRN-1 |  | Changes to the web application were made based on the concern |
|  |  | CRN-2 | All modules related to the use cases have been examined and identified and a matrix for the development distributed was created. |
|  |  | CRN-3 | No Relevant decisions made as this is in the user’s hands. |
| CRN-4 |  |  | No Relevant decisions made |

* + 1. **Iteration 2: Identifying Structures to Support Primary Functionality**

This section presents the results of the activities that are performed in each of the

steps of ADD in the second iteration of the design process. This iteration relates to the recommend game component for the game database system. This iteration expands on the more basic definitions made in the iteration 1 to make them more specific. This will aid in the drive of the implementation.

The expansion is part of the ADD method as initially the team cannot design everything at once vague definitions must be used to be built upon later to allow maneuverability in the system architecture. This iteration focuses on addressing risks the system can face during the building process and sharpening the structure of the system. The goal of iteration one has been met, so the goal of the second iteration must commence which will is to represent and explain the units of implementation.

* + - 1. Step 2: Establish Iteration Goal by Selecting Drivers

The goal of the second iteration is *identifying structures to support primary functionality*. By addressing these concerns, it aids in the understanding of how the Game\_Knight system’s functionality is allowed. This also explains how CRN-2-which is, the background knowledge of the team is essential in determining whether the system can be built or not. SQL knowledge is necessary, and python is also necessary in implementing the views for the statistical analysis function. It also allocates members to the respective position in the development team.

In the second iteration the architecture must also consider the following system’s use cases:

* UC-1: Monitor user interaction
* UC-3: Display recommended game
* UC-7: Create statistical analysis of game data
  + - 1. Step 3: Choose one or more Elements of the system to refine

The specific elements selected to be refined is the game application function which displays and manages game recommendations. This includes the backed aspect which provides the recommendation generation.

* + - 1. Step 4: Choose one or more design concepts that satisfy the selected drivers

Within the second iteration, design concepts in the architectural design patterns are selected. The following table below builds on the design decisions.

|  |  |
| --- | --- |
| Design Decisions and Location | Rationale and Assumptions |
| Use **Observer Design Pattern** to publish game recommendations and updates to users | There will be several cases in which multiple users should be pushed the same game recommendations based on their previous history. To better manage these cases, it can be advantageous to use an observer design pattern to better manage and more efficiently push updates to user accounts. |
| Introduce the **save (tactic**) in the system to monitor user interaction | The save tactic is introduced to be able to receive, process and save several events to log user interaction.  Implementing a save tactic is essential in the logging of user data, and allows more advanced statistical analysis to proceed |
| Use of systems **hover pattern** through the website to investigate the function operability and return it as a Boolean value | The system monitors the end user’s interactions in the Game\_Knight website. When the end user hovers in the vicinity of an interactable service provided by the systems it will provide a visual cue. The users can expand and collapse game directories and any related information. |
| Create a **Domain model** for the system and identify objects | Create an initial domain model to identify the major entities in the system along with the relationship between entities. Each object must be identified in the system for later use |

* + - 1. Step 5: Instantiate Architectural Elements, Allocate Responsibilities, and Define Interfaces

The instantiation of the architectural elements, responsibilities and definitions are explained in the table below.

|  |  |
| --- | --- |
| Design Decisions and Location | Rationale and Assumptions |
| Web application to disconnect local sources of data | With a strong and stable connection, it is viable to not store any data locally. The data layer is where server communication is handled. Communication between the many client components are directed via local method calls without any other support. |
| System use cases are mapped to domain objects | By analyzing the system’s use cases an initial identification of the domain objects can be made. |

* + - 1. Step 6: Sketch Views and Record Design Decisions
* Figure 1.5 initial domain model for the system.
* Figure 1.6 the domain objects that are instantiated for the use case

model in Section 1.2.1.

* Figure 1.7 shows a model of a module view with modules that are derived

from the business objects and linked with the primary use cases.

The description for the elements provided in figure 1.7 are explained in the table below it.

Diagram

Description automatically generated

**FIGURE 1.5** Initial domain model

**Diagram

Description automatically generated**

**FIGURE 1.6** The domain objects linked to the use case model

Diagram

Description automatically generated

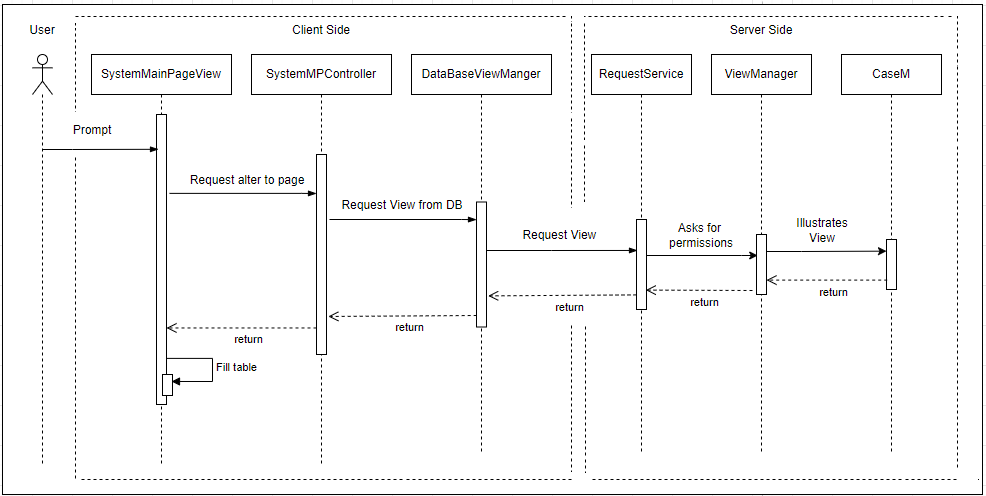
**FIGURE 1.7** Modules that support the essential use cases

|  |  |
| --- | --- |
| **Element** | **Responsibility** |
| SystemMainPageView | Displays the main page of the Game\_Knight system represent a view of the system that inherits the compilation of the systems functions and services |
| SystemMPController | Responsible for controlling the interactions with the main page as well as controlling the flow of information accessible to the end user |
| DataBaseTManager | Database tool manager responsible for managing the functions available to the system provided by the tuples in the database |
| DataBaseVManager | Responsible for managing the views in the system available from the database |
| RequestService | Provides a façade that receives prompts from the end user |
| ViewController | The responsibility of this element is to control the views of the database it can provide to the systems page based on if a user selects the function |
| Domain Ent | Contains the server-side domain model entities |
| DBServerCaseC | Controls the systems different cases |
| DBServerDataC | Controls the flow of data from the database linked to the system allows tuples to be shown from the various tables |
| ViewManager | Manages the views in the system |
| DBServerManager | Responsible for the server-side maintenance operatable without admin interaction |
| CaseM | Case mapper illustrates the various cases provided from the systems’ functions |
| Controller | The controller for the linked server |

The sequence diagram below for UC-3 was created to define interfaces in the system. Similar diagrams were created for the UC-1, and UC-7 but were not needed as they repeated interfaces, or their interfaces were deemed valid not to be represented.

**UC-3: Display Recommended Game**

Figure 1.8 shows the initial diagram for UC-3 (Display Recommended Game). It shows how the user views the systems web applications page after a user request a certain genre or searched a game. The SystemMainPageView on the server requests the DataBaseVManager on the server to retrieve tuples that fit the prompt of the user. The CaseM then displays the view on the user’s page.



**FIGURE 1.8** Sequence Diagram for UC-3 Display Recommended Game

|  |  |
| --- | --- |
| **Method Name** | **Description** |
| **Element:** SystemMainPageView | |
| Boolean functcall ()  Hover getHoverElement () | Provides a function that test whether one of the systems services have been prompted by the user  Gets the service the user hovered then clicked |
| **Element:** SystemMPController | |
| Response allow ()  Block () | Allows the page to change  Blocks change |
| **Element:** DataBaseViewManager | |
| Show\_View () | This method responsibility is to manage views within the database server |
| **Element:** RequestService | |
| Response  sendPrompt (Request x) | This method request various functions from the Game\_Knight system. |
| **Element:** ViewController | |
| Region requestView () | Request the view. This method returns the view the end user wants to display on the main page of the system allows the expansion and the collapse of the view |
| **Element:** CaseM | |
| Region retrieve (VARCHAR (45) tuple) | Displays the tuples associated with the view the user has requested |

* + - 1. Step 7: Perform Analysis of Current Design and Review Iteration Goal and Achievement of Design Process

In the table below the design decisions made during the second iteration are address as being, not addressed, partially addressed, or completely addressed. Design decisions completely addressed in the previous iteration are not referred in the table below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Not Addressed** | **Partially Addressed** | **Completely Addressed** | **Design Decisions Made During the Iteration** |
|  |  | UC-1 | Modules across the layers and preliminary interfaces to support this use case have been identified. |
|  |  | UC-3 | Modules across the layers and preliminary interfaces to support this use case have been identified. |
|  |  | UC-7 | Modules across the layers and preliminary interfaces to support this use case have been identified. |
|  | QA-1 |  | The Availability of the system would be 24/7 as it is a website |
|  | QA-2 |  | The modifiability of the system should be flexible as many new functions will be added when the database is extended |
|  | QA-3 |  | The performance of the system is crucial functions must return a value whether it completes the operation or it does not |
|  | QA-4 |  | The security of the website is not a priority as the user is saving their information locally |
| QA-5 |  |  | No Relevant decisions made |
|  | QA-6 |  | Data logging using the save tactic will aid in the usability attribute of the system. |
| CON-2 |  |  | No Relevant decisions made |
| CON-3 |  |  | No Relevant decisions made |
|  | CON-4 |  | Use of the save tactic to manage data logging will aid in this constraint as it is required in the greater implementation. |
|  | CRN-1 |  |  |
| CRN-4 |  |  | No Relevant decisions made |

* + 1. **Iteration 3: Addressing Quality Attribute Scenario Driver (QA-6)**

This section provides the results preformed in each step of the ADD method in the third iteration of the design process. Expanding the decisions made in the primary and secondary iteration the third iteration focus on the quality attribute of usability.

* + - 1. Step 2: Establish Iteration Goal by Selecting Drivers

For this iteration the architect focuses on the quality scenario 6 QA-6: The user desire functions that make searching for a game more informative. The functions must operate as intended, and if not, an error is sent to the maintenance technician.

* + - 1. Step 3: Choose One or More Elements of the System to Refine

For the usability scenario, the elements that were decided by the architect to refine were the:

* Web application system
* Database server
* API used for scraping game information
  + - 1. Step 4: Choose One or More Design Concepts That Satisfy the Selected Drivers

Design concepts in this iteration are listed below:

|  |  |
| --- | --- |
| **Design Decisions and Location** | **Rationale and Assumptions** |
| Implement the python script to scrap information from the database to create views | By using python and linking it to the MYSQL workbench database the system can create views based on the tuples provided in the tables the script can also be used to reference multiple tables from the database to create a multi table view with different columns from other tables. |
| Using Django web framework tables from the database can be converted into classes | These classes will aid in the construction of different variants of the primary web page in the web application. |
| Introduce elements from the message queue to allow trap technology | Traps received from the web servers are placed in a message queue and then retrieved in the application. The traps are processed and delivered to ensure that the functions of the system built of python and Django are properly functioning |

* + - 1. Step 5: Instantiate Architectural Elements, Allocate Responsibilities, and Define Interfaces

The instantiation decisions are explained in the table below:

|  |  |
| --- | --- |
| **Design Decisions and Location** | **Rationale** |
| Deploy a message queue | Deploying a message queue will ensure that no traps are lost in the case of failure in compiling the system’s various pages this is essential as in the case of failure caused by the traps not being saved, it can cause information not to be presented in the destination it is intended to be within |
| Use python and Django to compile the known views featured in the database server | The implementation of Django and python allows the system to be able to combine tables with simpler code than within the database server by referencing foreign keys within the tables. |
| Using the API scrap information and transfer it into the database to automatically fill tuples | The are two APIs featured in the Game\_Knight system find and add information automatically into the MYSQL workbench database. This information can be further developed into pages featured in the web application. |

The results of the design decisions are further inspected in the next step of the iteration process.

* + - 1. Step 6: Sketch Views and Record Design Decisions

Figure 1.9 shows a refined deployment diagram that includes decisions listed in the instantiation above

|  |
| --- |
| Diagram  Description automatically generated |

**FIGURE 1.9** Finalised Deployment Diagram

The tables below explains the responsibilities for the elements added to the updated version of the deployment diagram

|  |  |
| --- | --- |
| Element | Responsibility |
| API server | The API server was left out of the first iteration’s deployment diagram as it was an incomplete the API server is important to the system as it adds information scrapped online to the tables within the database it is used to create views of tables with sampled information. |
| TrapReceiver | The TrapReceiver gets traps from the system and converts it into events that can be used by the technician to improve the system. |

The UML sequence diagram shown in figure 1.10 represents how the TrapReceiver introduced in this iteration operates within the system and creates events to be further understood to how the system functions. This sequence diagram is associated with UC-2 (detect function operability), which is associated with the quality attributes QA-3 (performance) and QA-5 (testability).

The purpose of the sequence diagram is to represent the communication that occurs between the TrapReceiver and other lifelines.

A picture containing text, indoor

Description automatically generated

**FIGURE 1.10** TrapReceiver interaction diagram

* + - 1. Step 7: Perform Analysis of Current Design and Review Iteration Goal and Achievement of Design Purpose

In the third iteration, important design decisions have been created to address QA-6 usability, which also impacts QA-2 Modifiability, QA-3 Performance, the table below expands on the different drivers and the decisions involved in this iteration. Design decisions completely addressed in the previous iteration are not referred in the table below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Not Addressed** | **Partially Addressed** | **Completely Addressed** | **Design Decisions Made During the Iteration** |
|  | QA-1 |  | The Availability of the system was decided to be at all hours depending on if the other servers are functioning the way they are intended to |
|  | QA-2 |  | The modifiability of the system is dependant on the API and Django application systems if the information is present in the Django classes the script can make tuples. |
|  | QA-3 |  | The performance of the system is ensured if the TrapReceiver functions and 100% of the traps are processed, even in the situation of failure. |
|  |  | QA-4 | As the security of the website is not necessary it will no longer be addressed |
|  |  | QA-5 | The TrapReceiver ensures the testability of the system |
|  | QA-6 |  | The usability quality attribute is improved with the improvements of Modifiability and Performance quality attributes |
| CON-2 |  |  | No Relevant decisions made |
| CON-3 |  |  | No Relevant decisions made |
|  | CON-4 |  | Use of the save tactic to manage data logging will aid in this constraint as it is required in the greater implementation. |
|  | CRN-1 |  |  |
|  |  | CRN-4 | No Relevant decisions made |

|  |
| --- |
| * 1. **Summary**   In this report, the Game\_Knight system was progressed using the ADD method to design the pre-existing application a brownfield system. The ADD method was done with three iterations with the spotlight on different properties of the system: a concern, addressing elements tied to the functionality of the system, and addressing a key quality attribute.  In the Game\_Knight system the web application reference architecture was used for the structure of the system which allowed a simple implementation of the external servers and services involved in the web application. The web application structure was very well known amongst the team involved in the innovation of the system. The iterations presented illustrate how architectural concerns appear as design decisions in progression of the build.  The Game\_Knight system ADD process represents how architectural concerns, principal use cases and various quality attribute scenarios can alter the architectural design. As the system continues to improve many more iterations will be required to fine tune the results. Addressing other properties is the team’s next priority. |