esraa.dandash@city.ac.uk

HRG

Requirements documents for data cleaning tool

HRG

# Introduction

This Document is covering different aspects of what the HRG Hotel data cleaning tool will be doing. It covers the overall work plan for the project, a quick guide of the current system used for cleaning, the use cases of the new system under development and their specifications, in addition to the analysis class diagrams for the application.

|  |  |  |
| --- | --- | --- |
| WORK PLAN | | |
|  |  |  |  |  |
|  |  |  |  |  |
| |  |  |  |  | | --- | --- | --- | --- | | **Task Name** | **Start Date** | **End Date** | **Duration** | | Use Case Models | 17/02/2018 | 23/02/2018 | **6** | | Use Case Specifications | 23/02/2018 | 01/03/2018 | **6** | | Analysis Class diagrams | 23/02/2018 | 01/03/2018 | **6** | | Req. Spec. Submitted | 17/02/2018 | 04/03/2018 | **15** | | Design Class diagrams | 11/03/2018 | 16/03/2018 | **5** | | Modelling of GUI | 15/03/2018 | 19/03/2018 | **4** | | Modelling DB interaction | 15/03/2018 | 22/03/2018 | **7** | | Packages | 22/03/2018 | 25/03/2018 | **3** | | Sequence Diagrams | 22/03/2018 | 30/03/2018 | **8** | | State Diagrams | 25/03/2018 | 03/04/2018 | **9** | | GUI Design | 03/04/2018 | 09/04/2018 | **6** | | DB Design | 09/04/2018 | 14/04/2018 | **5** | | Defining Impl. Constraint | 10/04/2018 | 14/04/2018 | **4** | | Design Phase | 11/03/2018 | 15/04/2018 | **35** | | Implementation Phase | 15/04/2018 | 22/06/2018 | **68** | |  |  |  |  |
|  |  |  |  |  |

# Existing System

**Currently, Hotel cleaning is done in two different processes at HRG, while some colleagues would use method A, the Excel based method, others would use method B, which involves working with both SQL and Excel. Both methods are fairly similar to each other, however, method B gives quicker and more accurate results because it compares against a bigger database than the ones in Excel.**

## Method A

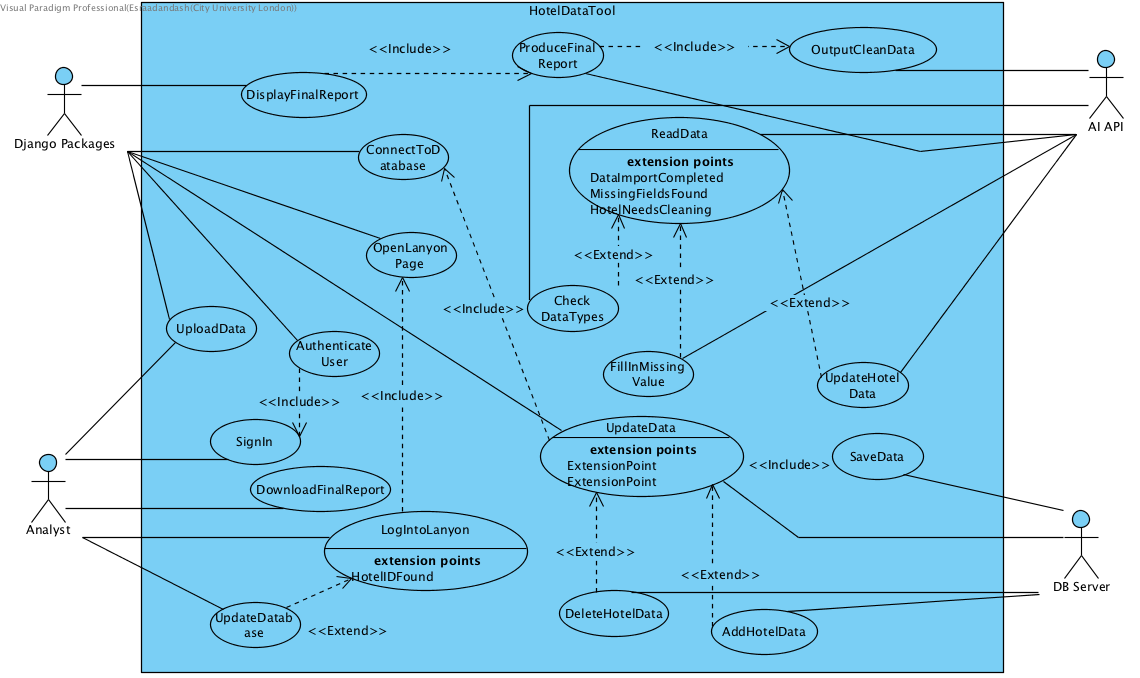
1. Insert all the new data into an Excel sheet.
2. Concatenate CountryName, CityName and HotelName.
3. Copy paste this new concatenated field into the cleaning table.
4. Vlookup the name against the existing database of hotels.
5. If there is no match found, manually search for this hotel’s data online, and log into Lanyon system online (LanyonID is the primary key for hotels, and Lanyon is not a database, it is merely a procurement tool), then search for this hotel’s LanyonID.
6. Add all data of the unmatched hotel into the cleaning database and the vlooukp will update automatically because the database it is running the formulas against is updated.
7. Repeat this for every unmatched hotel.
8. Add the entire dataset into the CLEAN Fields hotels in the new month’s dataset.
9. For hotels with no found records, leave them in the clean columns as they are because they normally are not big hotels and will have very little effect on the total consolidation.
10. The clean fields should include the clean HotelName, CityName, CountryName, HotelChain, HotelState for the US and Canada, LanyonID and HotelAdress.

## Method B

1. Open the cleaning master file in Excel, copy the hotel Name, City, State and Country into the corresponding columns of blue color in the sheet.
2. This Excel template is running some macros and scripts on the backend.
3. Hit the “Get Lanyon Details button”, thin runs the entire uncleaned data against an SQL database.
4. Unmatched fields will automatically be extracted from the data and imported into a table that the company later on sends to an outsourced company in India, they clean these hotel information for us, then we add the data into the SQL database.
5. The new cleaned fields will be updated and the green columns with Hotel Name, City, Country, Chain, LanyonID and Address are the final data that can be copied and pasted into the new month’s data.
6. For the unmatched fields, there is an extra step that some of the more experienced analyst do, which is to manually go through the data on the search for irregularities in names. For instance, they will remove Unicode characters, sometimes reverse the order of name (i.e. London Hilton to Hilton London), and essentially using their own observational skills and experience to minimize the number of unmatched records.
7. The unmatched final set is sent to an Indian outsourced team.

**These processes are not time efficient and have an element of error, due to the fact that both match the exact name, and have no intelligence when dealing with characters or the order of names. According to my colleagues, the first one can clean about 75% of the data, while the second one can go up to 85% in best cases. Another drawback is that there is a big amount of manual work done on the data, which is not only time-inefficient, but also costly for the company since the team’s time resource could be better used and more focused on reporting. Also, the analyst would never get an analysis of the cleaning process, which is vital since we optimally would need to go back to our sources and request better data that would reflect directly on our reporting and consulting services.**

# Use case diagram



# Use Case specifications

|  |
| --- |
| **Use Case: SignIn** |
| **ID: 1** |
| **Brief Description: The analyst logs into the system.** |
| **Primary Actors: Analyst** |
| **Secondary Actors: None** |
| **Preconditions: None** |
| **Main Flow:**   1. **The analyst enters the username/email** 2. **The analyst enters the password** |
| **Postconditions: None.** |
| **Alternative Flow: None.** |

|  |
| --- |
| **Use Case: AuthenticateUser** |
| **ID: 2** |
| **Brief Description: The system authenticates the user and allows access to the application or not.** |
| **Primary Actors: Django packages.** |
| **Secondary Actors: None.** |
| **Preconditions: The user entered the username/email and password.** |
| **Main Flow:**   1. **Include (signIn).** 2. **Compare user credentials against user DB.** 3. **If matched, allow access, otherwise, follow alternative flow.** |
| **Postconditions: User can use the system and upload the data.** |
| **Alternative Flow: Print a rejection message and re-open login window.** |

|  |
| --- |
| **Use Case: UploadData** |
| **ID: 3** |
| **Brief Description: The analyst uploads the new hotel data into the system.** |
| **Primary Actors: Analyst** |
| **Secondary Actors: Django packages** |
| **Preconditions: User is logged onto the system.** |
| **Main Flow:**   1. **Press upload data.** 2. **Browse for you data file, choose the file you need to clean in Excel (xlsx) format.** 3. **Press Choose.** 4. **Wait for upload to finish.** |
| **Postconditions:**  **A message indicating the completion of upload is displayed.** |
| **Alternative Flow: None.** |

|  |
| --- |
| **Use Case: ReadData** |
| **ID: 4** |
| **Brief Description: The AI package will import the data from Excel.** |
| **Primary Actors: AI API** |
| **Secondary Actors: None** |
| **Preconditions: Data is uploaded by the analyst.** |
| **Main Flow:**   1. **For every row, import all columns.** 2. **Stop running after all rows are read.** |
| **Postconditions: None.** |
| **Alternative Flow: None.** |

|  |
| --- |
| **Use Case: CheckDataTypes** |
| **ID: 5** |
| **Brief Description: Checking data types of the imported data.** |
| **Primary Actors: AI API** |
| **Secondary Actors: None.** |
| **Preconditions: Data import is complete.** |
| **Main Flow:**   1. **Check data in every field, if the type is different from what it should be, change it to the correct type.** |
| **Postconditions: None.** |
| **Alternative Flow: None.** |

|  |
| --- |
| **Use Case: FillInMissingValues** |
| **ID: 6** |
| **Brief Description:**  **To ensure there are no nulls or zeros where there shouldn’t be.** |
| **Primary Actors: AI API** |
| **Secondary Actors: None.** |
| **Preconditions: Data import complete.** |
| **Main Flow:**   1. **For every currency field, format them correctly as a number.** 2. **For particular fields, populate nulls according to the standards in HRG such as UNKNOWN or UNLINKED.** |
| **Postconditions: None** |
| **Alternative Flow: None** |

|  |
| --- |
| **Use Case: UpdateHotelData** |
| **ID: 7** |
| **Brief Description: Cleaning the data.** |
| **Primary Actors: AI API.** |
| **Secondary Actors: None.** |
| **Preconditions: Data Import complete.** |
| **Main Flow:**   1. **Follow the set of cleaning rules for every record.** 2. **Remove the hotel from the uncleaned set.** 3. **Add the cleaned record to the new clean set.** 4. **Repeat for every record in the uncleaned set.** |
| **Postconditions: Data cleaning completed message displayed.** |
| **Alternative Flow: If there is not issue with the data, remove the hotel directly from uncleaned set and place in the clean set.** |

|  |
| --- |
| **Use Case: OutputCleanData** |
| **ID: 8** |
| **Brief Description: The system exports the new set of clean data.** |
| **Primary Actors: AI API** |
| **Secondary Actors: None** |
| **Preconditions: Data cleaning complete.** |
| **Main Flow:**   1. **Connect to Django app.** 2. **Export data to app.** |
| **Postconditions: None** |
| **Alternative Flow: None** |

|  |
| --- |
| **Use Case: ProduceFinalReport** |
| **ID: 9** |
| **Brief Description: Produce analysis on number of matches from hotels and percentage cleaning achieved.** |
| **Primary Actors: AI API** |
| **Secondary Actors: None** |
| **Preconditions: Data Cleaning Complete** |
| **Main Flow:**   1. **Includes (OutputCleanData)** 2. **Identify number of hotel in clean data vs uncleaned data.** 3. **Run a count on both data sets and produce a quick overview of the count and concentration of errors in the hotel data.** |
| **Postconditions: None** |
| **Alternative Flow: None** |

|  |
| --- |
| **Use Case: DisplayFinalReport** |
| **ID: 10** |
| **Brief Description: The application should display the results to the user** |
| **Primary Actors: Django API** |
| **Secondary Actors: AI API** |
| **Preconditions: Latest AI API session is connected to App.** |
| **Main Flow:**   1. **Include (ProduceFinalReport)** 2. **Get the report produced by the AI API.** 3. **Display this report to the user.** |
| **Postconditions: None** |
| **Alternative Flow: If nothing is displayed, Refresh the connection to AI API.** |

|  |
| --- |
| **Use Case: DownloadFinalReport** |
| **ID: 11** |
| **Brief Description: The analyst shall get an offline copy of the final cleaning report.** |
| **Primary Actors: Analyst** |
| **Secondary Actors: Django Packages** |
| **Preconditions: Report is displayed** |
| **Main Flow:**   1. **Press download report button.** 2. **Choose the destination folder.** 3. **Press OK.** 4. **Check it is downloaded as requested.** |
| **Postconditions: None** |
| **Alternative Flow: None** |

|  |
| --- |
| **Use Case: OpenLanyon** |
| **ID: 12** |
| **Brief Description: In a new window, open the Lanyon website for the analyst to manually search for a hotel.** |
| **Primary Actors: Analyst** |
| **Secondary Actors: Django API** |
| **Preconditions: None** |
| **Main Flow:**   1. **Press Use Lanyon button** 2. **Take the analyst to Lanyon Website** |
| **Postconditions: None** |
| **Alternative Flow: None** |

|  |
| --- |
| **Use Case: LogIntoLanyon** |
| **ID: 13** |
| **Brief Description: The analyst needs to log into lanyon** |
| **Primary Actors: Analyst** |
| **Secondary Actors: None** |
| **Preconditions: None** |
| **Main Flow:**   1. **Include (OpenLanyon)** 2. **Enter username and Password.** |
| **Postconditions: User is logged into Lanyon and can search for the hotel** |
| **Alternative Flow: If user credentials are wrong, allow the user to re-enter their details.** |

|  |
| --- |
| **Use Case: ConnectToDatabase** |
| **ID: 14** |
| **Brief Description: Establishing a live connection to the existing HRG HotelCleaning database on SQL servers.** |
| **Primary Actors: Django Packages.** |
| **Secondary Actors: None** |
| **Preconditions: None** |
| **Main Flow:**   1. **Establish a live connection to SQL Server.** |
| **Postconditions: Connection between the application and DB is established.** |
| **Alternative Flow: None.** |

|  |
| --- |
| **Use Case: UpdateDatabase** |
| **ID: 15** |
| **Brief Description: Hotels cleaned should be added to client's database if the analyst wishes to, and to the cleaning hotel.** |
| **Primary Actors: Analyst** |
| **Secondary Actors: None** |
| **Preconditions: New data imported** |
| **Main Flow:**   1. **Press connect to Client’s DB button.** 2. **SQL management studio will open.** 3. **Browse to your client’s database and import data.** |
| **Postconditions: Client’s database updated.** |
| **Alternative Flow: None** |

|  |
| --- |
| **Use Case: AddHotelData** |
| **ID: 16** |
| **Brief Description:**  **Hotels that have never been in the current DB should be added.** |
| **Primary Actors: DB Server** |
| **Secondary Actors: None** |
| **Preconditions: A hotel is identified as new.** |
| **Main Flow:**   1. **Obtain hotel state from Django.** 2. **Add Hotel data on server.** |
| **Postconditions: None** |
| **Alternative Flow: None** |

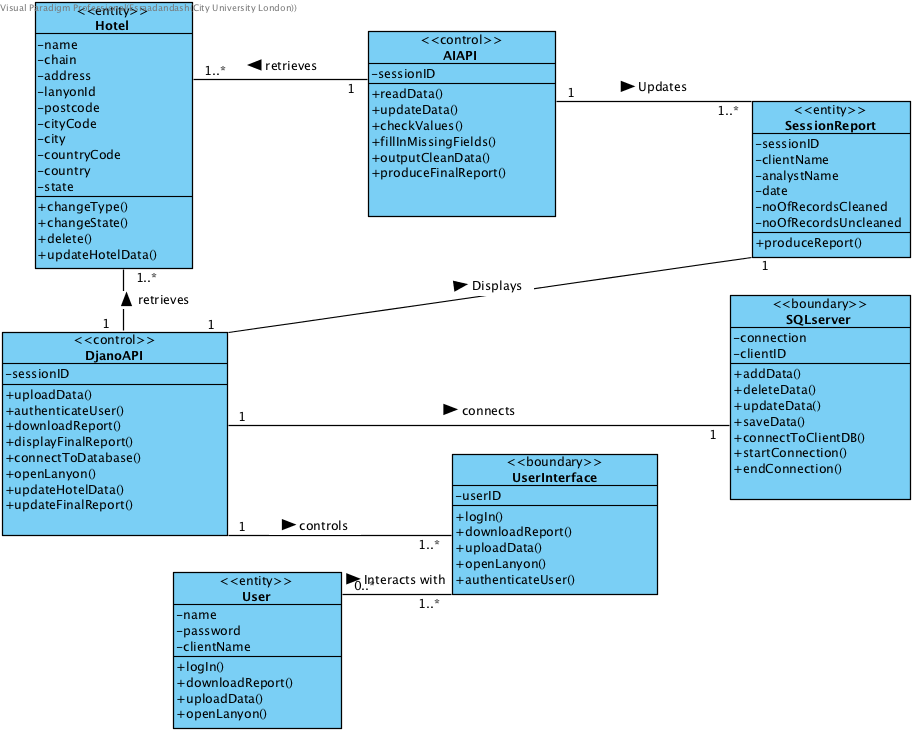
|  |
| --- |
| **Use Case: DeleteHotelData** |
| **ID: 17** |
| **Brief Description: Hotels closed or updated should be deleted from the existing DB** |
| **Primary Actors: DB server** |
| **Secondary Actors: None** |
| **Preconditions: A hotel is identified as closed or changed.** |
| **Main Flow:**   1. **Obtain hotel state from Django.** 2. **Change Hotel data on server.** |
| **Postconditions: None** |
| **Alternative Flow: None** |

|  |
| --- |
| **Use Case: SaveData** |
| **ID: 18** |
| **Brief Description:**  **All changes need to be saved last.** |
| **Primary Actors: DB Server** |
| **Secondary Actors: None** |
| **Preconditions: All Hotel identification and deletion or addition or update are finished.** |
| **Main Flow:**   1. **Save data as it is now.** 2. **Close management studio.** 3. **Print a session finished message.** |
| **Postconditions: None** |
| **Alternative Flow: None** |

# Use cases order and prioritisation

|  |  |  |
| --- | --- | --- |
| Use Case Name | Rank | Reason |
| SignIn | 1 | There is a login system implemented. |
| AuthenticateUser | 2 | This app will need to provide restricted access so that not anyone can manipulate the data. |
| UploadData | 3 | The analyst has to upload the dirty data file. |
| ReadData | 4 | Importing the data into the AI system. |
| CheckDataTypes | 5 | Some initial checks done first to ensure some fields such as dates and floats are in the correct formats otherwise data maybe truncated or corrupted during cleaning. |
| FillInMissingValues | 6 | To ensure there are no nulls or zeros where there shouldn’t be. |
| UpdateHotelData | 7 | This is the main cleaning set of procedures. |
| OutputCleanData | 8 | To feed back into the database. |
| ProduceFinalReport | 9 | Build full analysis on how fit the data was for cleaning and how many rows had a successful cleaning and other indicators such as the most repeated hotel. |
| DisplayFinalReport | 10 | Provide this analysis to the user. |
| DownloadFinalReport | 11 | The analyst should be able to keep a copy of the analysis to be able to negotiate changes with the client using it. |
| OpenLanyon | 12 | In case there are unmatched hotels, the analyst should be able to search for it on Lanyon website. |
| LogIntoLanyon | 13 | Lanyon is a limited access procurement tool and only few analysts can log in. |
| ConnectToDatabase | 14 | Establishing a live connection to the existing HRG HotelCleaning database on SQL servers. |
| UpdateDatabase | 15 | Hotels cleaned should be added to client's database if the analyst wishes to, and to the cleaning hotel. |
| AddHotelData | 16 | Hotels that have never been in the current DB should be added. |
| DeleteHotelData | 17 | Hotels whose data has been altered, or have been shut down should be deleted and added again if needed. |
| SaveData | 18 | All changes need to be saved last. |

# Class Diagrams



# Conclusion

In conclusion, the new system will provide a more efficient way of cleaning hotel data, ensuring better quality checks and more robust data analysis. It relies on a group of web frameworks and machine leaning frameworks to identify errors and inconsistencies within the data, correct them, and report of the data quality from the beginning. It has 18 use case so far, with the plan to further develop these whenever needed in the next phases.