esraadandash@gmail.com

HRG

Implementation Report HRG DATA Cleaning TOOL

Table of Contents

[Compilation: 2](#_Toc517603802)

[project components 2](#_Toc517603803)

[The main Project Tree 2](#_Toc517603804)

[The Application –Hotel Cleaner Tree 2](#_Toc517603805)

[All Project Directories 3](#_Toc517603806)

[Run-time 4](#_Toc517603807)

[list of cots used 4](#_Toc517603808)

[traceability – design vs runtime components 5](#_Toc517603809)

[dependancies between different components 5](#_Toc517603810)

[Component Diagram 6](#_Toc517603811)

[Testing 6](#_Toc517603812)

[unit testing 6](#_Toc517603813)

[Tests for every Use Case 6](#_Toc517603814)

[system testing 8](#_Toc517603815)

[Operational Profile 9](#_Toc517603816)

[Automation of Testing 9](#_Toc517603817)

[Non-functional Testing 9](#_Toc517603818)

[Appendix A 10](#_Toc517603819)

[project directory 10](#_Toc517603820)

[installed apps section in settings.py 10](#_Toc517603821)

[urls.py 10](#_Toc517603822)

[quick project run with screenshots 11](#_Toc517603823)

# Compilation:

**My Web Application, Called Hotel Genius, is built in Python, using the Django framework, which has a very well organised files directory and follows a really good overall layout/structure that any Django Project has.**

**The implementation of project follows the most common Compilation of Python, allowing CPython to compile the source files into a Python-specific lower-level form, known as bytecode, then it saves the bytecode files to disk (to avoid recompiling them in the future). These cached .pyc files are then interpreted on a virtual machine at runtime, which speeds up the overall performance since they don’t need to be imported every time the program is run.**

## project components

The following Trees will be used to demonstrate different components of the project and the directory structure:

Any Django project –Hotel Genius in my case- consists of a specific set of .py files, in addition to at least one application –Hotel Cleaner in my case- and its enclosed files.

### The main Project Tree

├── \_\_init\_\_.py

├── \_\_pycache\_\_

│   ├── \_\_init\_\_.cpython-36.pyc

│   ├── settings.cpython-36.pyc (all .pyc are already compiled and cached)

│   └── urls.cpython-36.pyc

├── settings.py (where DB, Template and other configurations are done)

├── urls.py (another file that includes mapping to all views on my page)

└── wsgi.py

### The Application –Hotel Cleaner Tree

├── \_\_init\_\_.py

├── \_\_pycache\_\_

│   ├── \_\_init\_\_.cpython-36.pyc

│   ├── admin.cpython-36.pyc (all .pyc are already compiled and cached)

│   ├── models.cpython-36.pyc

│   └── views.cpython-36.pyc

├── admin.py (Where I register all my models)

├── apps.py (Where I add all my apps that I want to be configured)

├── migrations (This is any model of my created models that I ran a migrate command for)

│   ├── 0001\_initial.py

│   ├── \_\_init\_\_.py

│   └── \_\_pycache\_\_

│       └── \_\_init\_\_.cpython-36.pyc

├── models.py (Where I create all my models according to my designed schema)

├── tests.py ( Comes as a default, but I haven’t used it)

└── views.py (This is where all my application’s functionality sits, where all responses for my rendered views are created)

The Project includes more files, such as Templates, including all my Web pages as HTML files, Static files like Bootstrap and CSS for styling my pages, Populate-db.py which is a Script that populates one of Database tables, and manage.py, which is a standard .py file in any Django project that is the most important because it has all commands to start projects, start apps within the project, make any Database migrations, and most importantly, run server which runs the whole application (on the local server by default) and allows us to run all the code.

Another important component in my project is the venv directory, for my virtual environment, and lib directory, which includes the pip (package manager for my project), all the Django packages and different libraries that have been imported throughout my project, such as Django admin and Python3.

Additionally, my data is stored on a SQLite database, in db.sqlite3 file.

This Overall tree includes all the files under all directories within my Project. Please note than not all files in lib directory could be shown due to their depth in the tree and their considerable amounts.

### All Project Directories

├── DB\_NAME

├── HotelCleaner

│   ├── \_\_init\_\_.py

│   ├── \_\_pycache\_\_

│   │   ├── \_\_init\_\_.cpython-36.pyc

│   │   ├── admin.cpython-36.pyc

│   │   ├── models.cpython-36.pyc

│   │   └── views.cpython-36.pyc

│   ├── admin.py

│   ├── apps.py

│   ├── migrations

│   │   ├── 0001\_initial.py

│   │   ├── \_\_init\_\_.py

│   │   └── \_\_pycache\_\_

│   │       └── \_\_init\_\_.cpython-36.pyc

│   ├── models.py

│   ├── tests.py

│   └── views.py

├── HotelGenius

│   ├── \_\_init\_\_.py

│   ├── \_\_pycache\_\_

│   │   ├── \_\_init\_\_.cpython-36.pyc

│   │   ├── settings.cpython-36.pyc

│   │   └── urls.cpython-36.pyc

│   ├── settings.py

│   ├── urls.py

│   └── wsgi.py

├── db.sqlite3

├── manage.py

├── populate-db.py

├── static

│   ├── bootstrap-3.3.7-dist

│   │   ├── css

│   │   │   ├── bootstrap-theme.css

│   │   │   ├── bootstrap-theme.css.map

│   │   │   ├── bootstrap-theme.min.css

│   │   │   ├── bootstrap-theme.min.css.map

│   │   │   ├── bootstrap.css

│   │   │   ├── bootstrap.css.map

│   │   │   ├── bootstrap.min.css

│   │   │   └── bootstrap.min.css.map

│   │   ├── fonts

│   │   │   ├── glyphicons-halflings-regular.eot

│   │   │   ├── glyphicons-halflings-regular.svg

│   │   │   ├── glyphicons-halflings-regular.ttf

│   │   │   ├── glyphicons-halflings-regular.woff

│   │   │   └── glyphicons-halflings-regular.woff2

│   │   └── js

│   │       ├── bootstrap.js

│   │       ├── bootstrap.min.js

│   │       └── npm.js

│   ├── css

│   │   └── style.css

│   └── images

├── templates

│   └── HotelCleaner

│       ├── cleaning.html

│       ├── index.html

│       ├── report.html

│       └── upload.html

└── venv

    ├── bin

    │   ├── \_\_pycache\_\_

    │   │   └── django-admin.cpython-36.pyc

    │   ├── activate

    │   ├── activate.csh

    │   ├── activate.fish

    │   ├── django-admin

    │   ├── django-admin.py

    │   ├── easy\_install

    │   ├── easy\_install-3.6

    │   ├── pip

    │   ├── pip3

    │   ├── pip3.6

    │   ├── python

    │   ├── python3

    │   └── python3.6

    ├── include

    ├── lib

    │   └── python3.6

    │       └── site-packages

    │           ├── Django-2.0.6.dist-info

A screenshot of the project files is included in the Appendix A.

# Run-time

## list of cots used

* Django Framework to implement all the classes.
* Django Admin Interface, which helps modify instances of the models in the database using an easy-to-use GUI.
* Different Python libraries such as CSV to read csv files.

## traceability – design vs runtime components

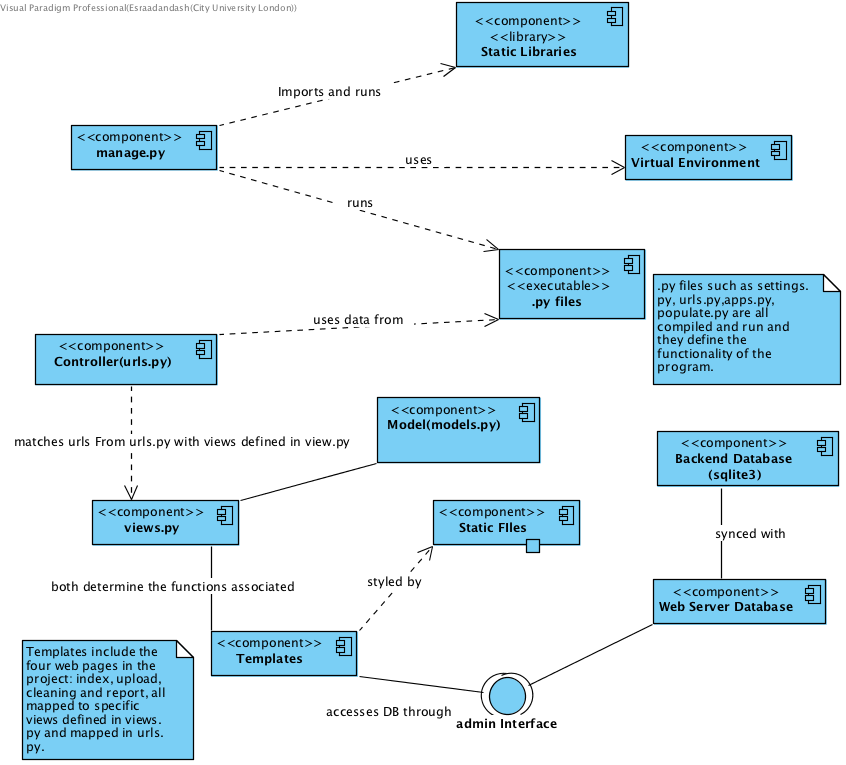
Please note that due to the nature of Django, which I did not know in depth while designing my packages, I will be mapping mainly classes to .py files, models or views. Django does not implement a packages system therefore I will be tracing the classes and listing their parent directory so that are very easy to locate within the project’s directory.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Design | | | Implementation | | |
| Class Name | Package Name | Comment | Class Name | Parent Directory | Language |
| GUI\_Login | GUIs | Web page | Index.html | Templates | HTML |
| GUI\_UploadData | GUIs | Web page | Upload.html | Templates | HTML |
| GUI\_CleaningInProgress | GUIs | Web page | Cleaning.html | Templates | HTML |
| GUI\_SessionReport | GUIs | Web page | Report.html | Templates | HTML |
| Hotel | CleaningDatabases | DB entity | Dirty Hotel in Models.py | HotelCleaner | Python |
| CleaningDB | CleaningDatabases | DB entity | Lanyon Hotel in Models.py | HotelCleaner | Python |
| DataCleaner | MainCleaningFunctionality | Main cleaning functionality | Views.py – upload(request) | HotelCleaner | Python |
| CleanedRecords | FinalReports | DB entity | Cleaned Hotel in Models.py | HotelCleaner | Python |
| UncleanedRecords | FinalReports | DB entity | Uncleaned Hotel in Models.py | HotelCleaner | Python |
| SessionReport | FinalReports | Session-specific information | Views.py – report(request | HotelCleaner | Python |

## dependancies between different components

* Any application needs to be installed in the settings.py file INTALLED APPS section to be able to migrate and use –please find the screenshot to demonstrate this is Appendix A.
* All web pages are gives a URL regex and a name to be throughout the project, in the URLs.py, please find the screenshot to demonstrate this in Appendix A.
* All requests from the 4 web pages are handled and rendered differently in views.py.
* Please note that this is a standalone project and does not require any external database services/hosts. Therefore, there is no deployment diagram.

### Component Diagram



# Testing

## unit testing

All tests for specific use cases are demonstrated below. Please note that only the essential use cases are included in those unit tests, other secondary use cases are automatically tested in the system testing.

### Tests for every Use Case

|  |  |
| --- | --- |
| Use Case ID: 1 | Use Case Name: SignIn |
| Test Number: 1 | |
| Objective: User should be able to enter their name and client so that the client cannot start the cleaning without enter these details. | |
| Set Up: Create 2 Text Boxes, labels and a submission button, then program the response in views.py to obtain the details entered and save them to the session automatically. | |
| Expected Results: Page displays an error | |
| Test: 1. Leave any of the records blanks.  2. Press Submit. | |
| Test Record: An error is thrown successfully. | |
| Date: 10/06/2018 | Tester: Esraa Dandash |
| Result: An error is thrown successfully. | |
| Date: 11/06/2018 | Tester: Esraa Dandash |
| Result: An error is thrown successfully. | |

|  |  |
| --- | --- |
| Use Case ID: 3 | Use Case Name: UploadData |
| Test Number: 1 | |
| Objective: Upload a csv file to the application’s upload box in the upload page. | |
| Set Up: Hit the choose button and browse to find the csv file. | |
| Expected Results: File uploaded on the page and its name is showing. | |
| Test: 1. Press choose.  2. Browse your laptop to find the .csv file for dirty hotel data.  3. Press ok | |
| Test Record: Run successfully. | |
| Date: 11/06/2018 | Tester: Esraa Dandash |
| Result: File name shown correctly. | |
| Date: 12/06/2018 | Tester: Esraa Dandash |
| Result: File name shown correctly. | |

|  |  |
| --- | --- |
| Use Case ID: 4 | Use Case Name: ReadData |
| Test Number: 1 | |
| Objective: Read a csv file that was uploaded to the application’s Dirty Hotel table in the Database. | |
| Set Up: Upload file through the previous use case. | |
| Expected Results: The same hotel is the uploaded CSV files can be found on the admin interface Dirty Hotel table. | |
| Test: 1. Press the Clean on the Upload page.  2. Open the admin interface by adding /admin in the urn bar. | |
| Test Record: All hotels showing in the admin’s Dirty Hotel table. | |
| Date: 14/06/2018 | Tester: Esraa Dandash |
| Result: Failed, Code debugging and reformatting some loop features solved the issue. | |
| Date: 14/06/2018 | Tester: Esraa Dandash |
| Result: Success | |

|  |  |
| --- | --- |
| Use Case ID: 7 | Use Case Name: UpdateHotelData |
| Test Number: 1 | |
| Objective: Updating dirty hotel data | |
| Set Up: Delete any cleaned or uncleaned data from previous sessions. | |
| Expected Results: Matched hotels have the LanyonID added to them and present in the cleaned hotel table. | |
| Test: 1. Upload file.  2. Press Clean Data.  3. Open the admin interface | |
| Test Record: All lanyonIDs for cleaned hotels showing in the admin’s Cleaned Hotel table. | |
| Date: 14/06/2018 | Tester: Esraa Dandash |
| Result: Success | |
| Date: 14/06/2018 | Tester: Esraa Dandash |
| Result: Success | |

|  |  |
| --- | --- |
| Use Case ID: 8 | Use Case Name: OutputCleanData |
| Test Number: 1 | |
| Objective: Check the cleaned file produced when the user downloads clean data. | |
| Set Up: Have the cleaned hotel table open in Django admin to check against. | |
| Expected Results: All Hotel data on the database should be on the file as well. | |
| Test: 1. Upload dirty data file.  2. Press clean on the upload page.  3. Press Get Report on the cleaning page.  4. Press Get Cleaned Data on the report page.  5. Open the downloaded file and compare against the cleaned hotel table on the admin interface, and sense check the data in it. | |
| Test Record: All data showing is complete and matched. | |
| Date: 15/06/2018 | Tester: Esraa Dandash |
| Result: Success | |
| Date: 16/06/2018 | Tester: Esraa Dandash |
| Result: Success | |

|  |  |  |
| --- | --- | --- |
| Use Case ID: 10 | Use Case Name: DisplayFinalReport | |
| Test Number: 1 | | |
| Objective: The user should be able to see some quick analysis on the data quality. | | |
| Set Up: All cleaning should be finished before being able to view the report. | | |
| Expected Results: A quick analysis of the Data quality. | | |
| Test: 1. Press Get Report on the cleaning web page.  2. Check that all figures are shown correctly. | | |
| Test Record: All figures showing on the final page. | | |
| Date: 17/06/2018 | | Tester: Esraa Dandash |
| Result: Success | | |
| Date: 21/06/2018 | | Tester: Esraa Dandash |
| Result: Success | | |

|  |  |
| --- | --- |
| Use Case ID: 16 | Use Case Name: AddHotelData |
| Test Number: 1 | |
| Objective: If a new LanyonHotel is created, the user shall be able to add it to the database the application compares the data against. | |
| Set Up: Have your login created to the Django admin interface. | |
| Expected Results: The new hotel is added to the LanyonHotel table. | |
| Test: 1. Open the admin by typing /admin after the local host url in the URL bar.  2. After Logging in, open the LanyonHotel table.  3. Click add and enter the hotel fields.  4. Press save. | |
| Test Record: Hotel showing on the list. | |
| Date: 20/06/2018 | Tester: Esraa Dandash |
| Result: Success | |
| Date: 21/06/2018 | Tester: Esraa Dandash |
| Result: Success | |

|  |  |
| --- | --- |
| Use Case ID: 17 | Use Case Name: DeleteHotelData |
| Test Number: 1 | |
| Objective: If a Hotel is no longer open and it needs to be deleted from Lanyon, we should be able to delete it. | |
| Set Up: Have your login created to the Django admin interface. | |
| Expected Results: The hotel does not exist on the LanyonHotel Table anymore. | |
| Test: 1. Open the admin by typing /admin after the local host url in the URL bar.  2. After Logging in, open the LanyonHotel table.  3. Open the Hotel field.  4. Press delete, then OK. | |
| Test Record: Hotel disappearing from the list. | |
| Date: 18/06/2018 | Tester: Esraa Dandash |
| Result: Success | |
| Date: 19/06/2018 | Tester: Esraa Dandash |
| Result: Success | |

|  |  |
| --- | --- |
| Use Case ID: 18 | Use Case Name: SaveData |
| Test Number: 1 | |
| Objective: The application should be able to save the data after any session or during it. | |
| Set Up: The database should be up and running on the server (migrated). | |
| Expected Results: The Lanyon Hotel, dirty, cleaned and uncleaned hotel tables should all be correctly populated.And the correct number of them should be displayed on the admin interface. | |
| Test: 1. Run populate-db to populate Lanyon Hotel table with the latest Lanyon list.  2. On the Django Interface, check the number of imported hotels. | |
| Test Record: All 4 tables are well populated after fixing a bug in the CSV reader. | |
| Date: 17/06/2018 | Tester: Esraa Dandash |
| Result: Success | |
| Date: 21/06/2018 | Tester: Esraa Dandash |
| Result: Success | |

## system testing

### Operational Profile

|  |  |  |
| --- | --- | --- |
| Use Case Name | Use Case ID | Probability of This Use Case |
| SignIn | 1 | 0.1 |
| UploadData | 3 | 0.02 |
| ReadData | 4 | 0.1 |
| CheckDataTypes | 5 | 0.02 |
| FillInMissingValues | 6 | 0.05 |
| UpdateHotelData | 7 | 0.05 |
| OutputCleanData | 8 | 0.1 |
| ProduceFinalReport | 9 | 0.05 |
| DisplayFinalReport | 10 | 0.15 |
| DownloadFinalReport | 11 | 0.01 |
| ConnectToDatabase | 14 | 0.01 |
| UpdateDatabase | 15 | 0.1 |
| AddHotelData | 16 | 0.06 |
| DeleteHotelData | 17 | 0.03 |
| SaveData | 18 | 0.15 |

### Automation of Testing

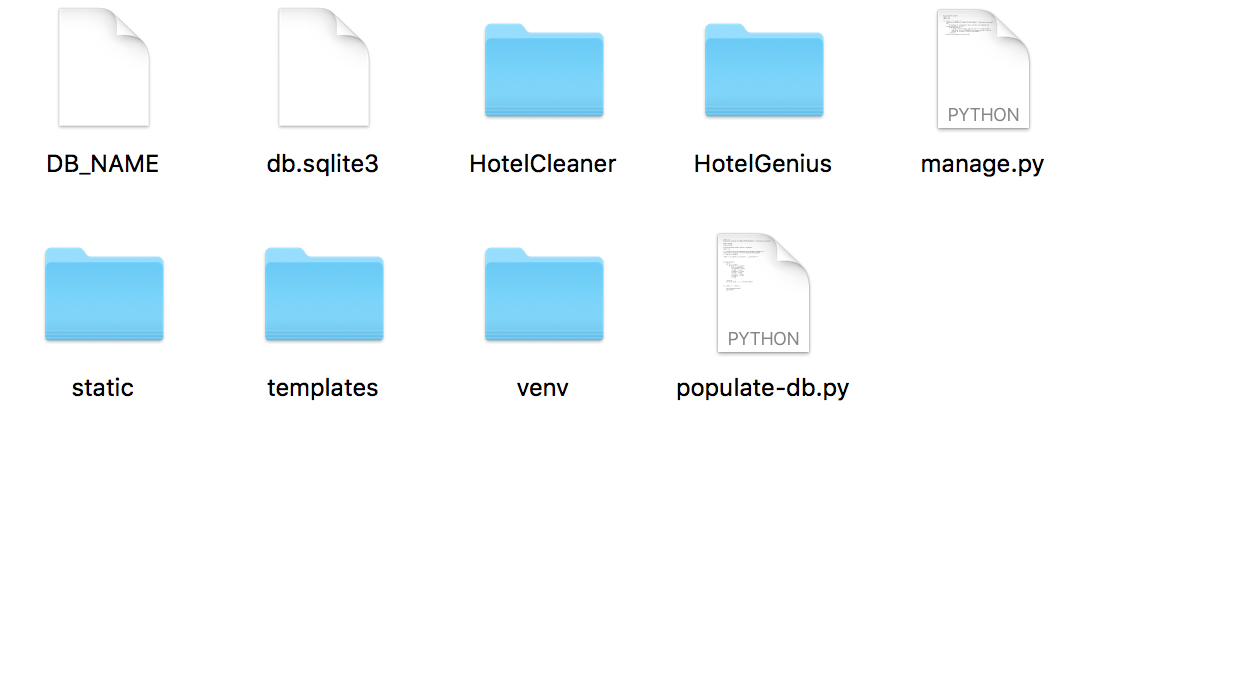
* Creation of a system that regularly connect the Lanyon procurement tool, where we get the download to upload it back to our Hotel Lanyon database, so that the update is done automatically without having to do all the download and uploads manually, then it tests if the correct set of Lanyon Hotel is on our database.
* A system that tracks the connection with the database in real time.
* A system that keeps comparing any specific data to ensure it has been added, updated, deleted or saved.
* A system that tracks user login inputs and maintains a secure login in.

### Non-functional Testing

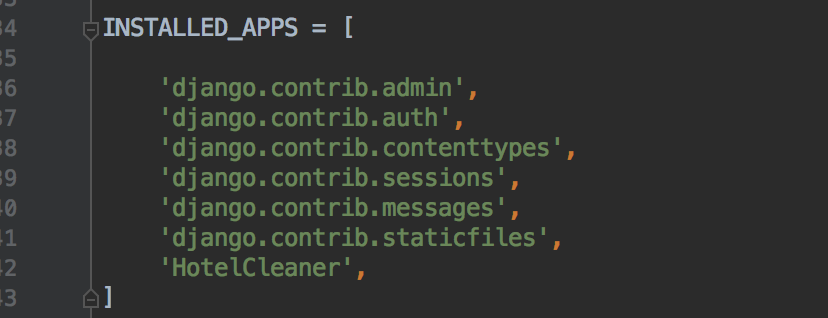
* Security:Penetration testing by simulating an attack from a malicious hacker. This testing involves analysis of a particular system functionality to check for potential vulnerabilities to a hacking attempt.
* Scalability: Increasing the size of the uploaded .csv file and counting how long it will take as the file size increases.
* Reliability: Counting the number of times any use case Test is not passed, divided by the total number of running the program tests.
* Availability: Run on multiple computer if varying computing powers, or in different locations.
* Usability: Ask different team members to use the application and navigate through it, and take notes of their feedback.
* Reusability: Evaluate how much can the team further add to the code to extend the system’s capabilities and use cases.

# Appendix A

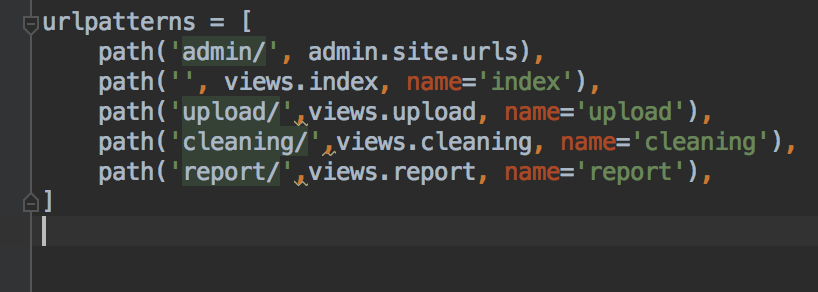
## project directory



## installed apps section in settings.py



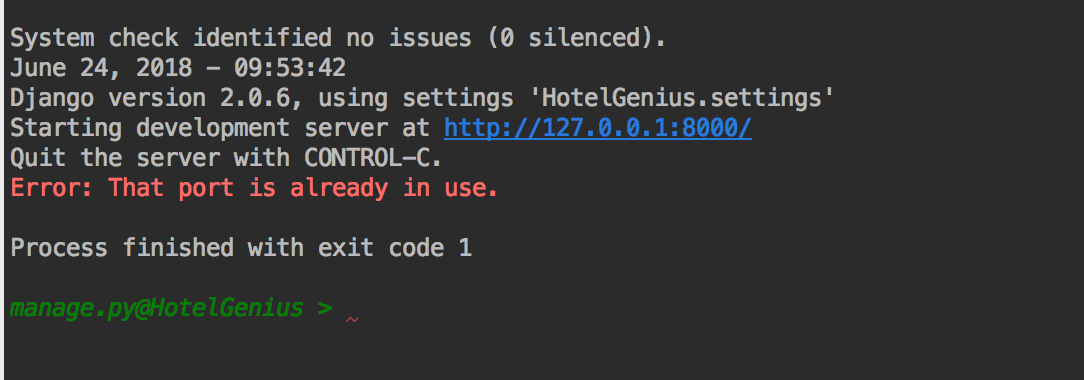
## urls.py



## quick project run with screenshots

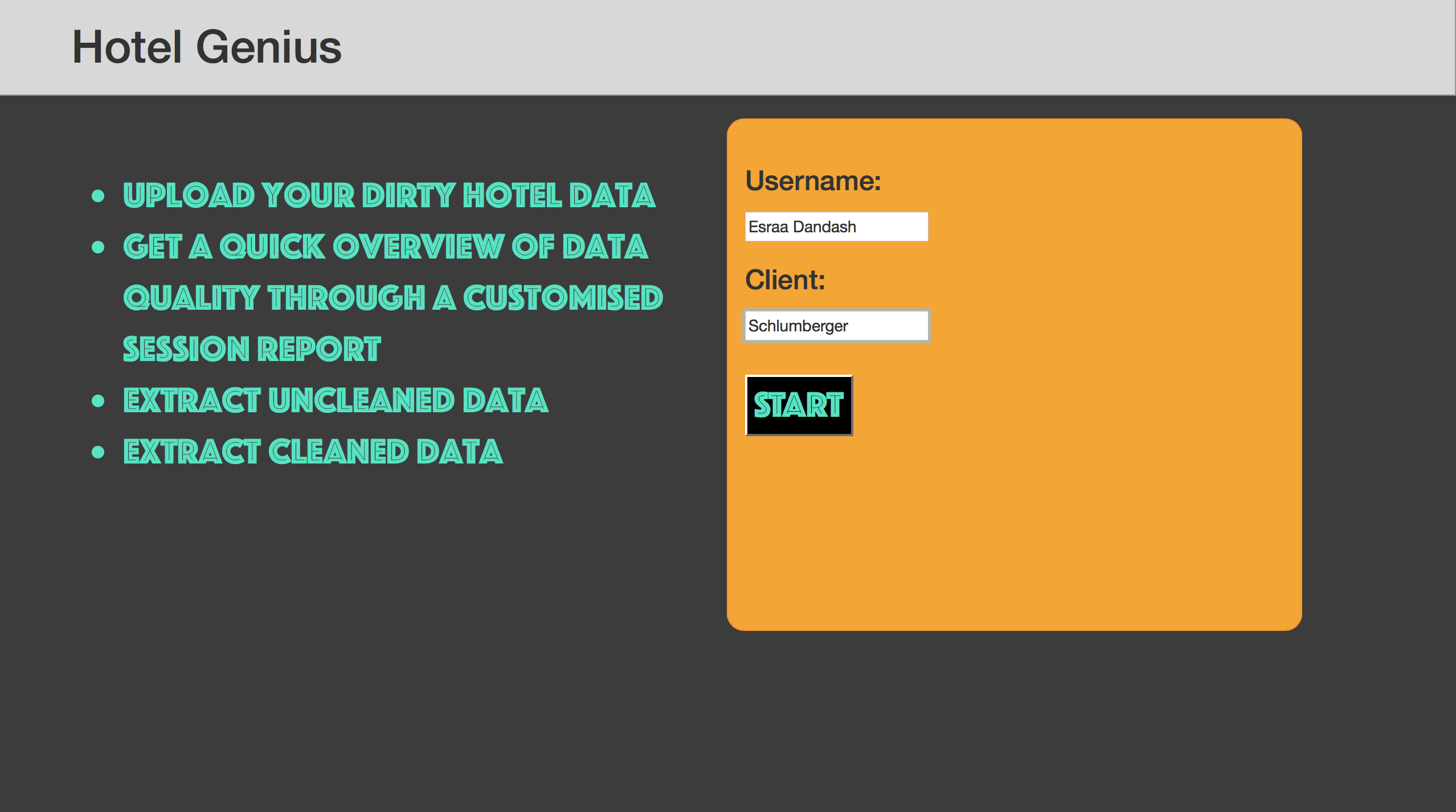
From the terminal, run python manage.py runserver.

Then it will show this success message, click on the local server to open the home page.

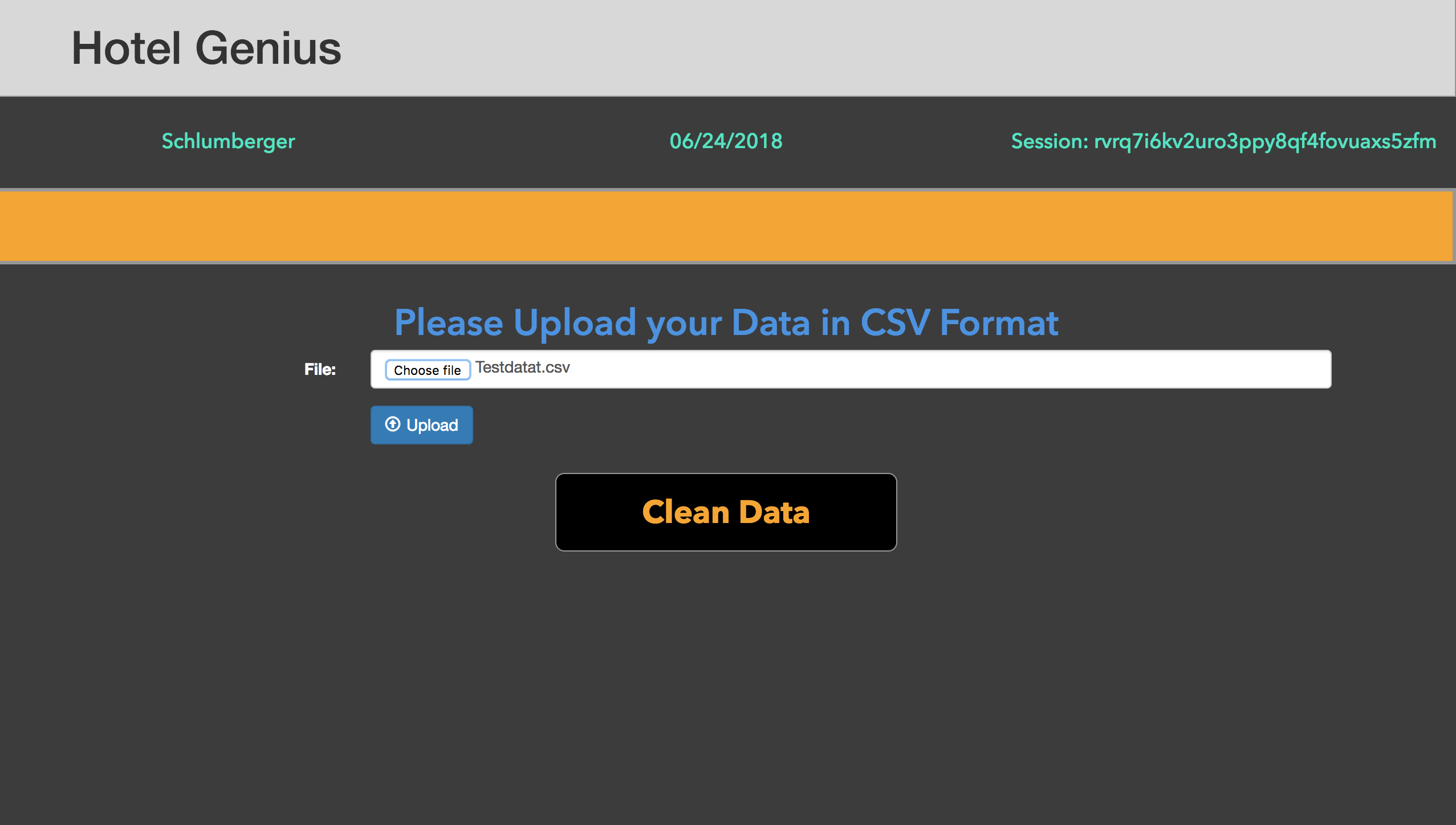


1. The open index page should have a form to fill in the username and client name

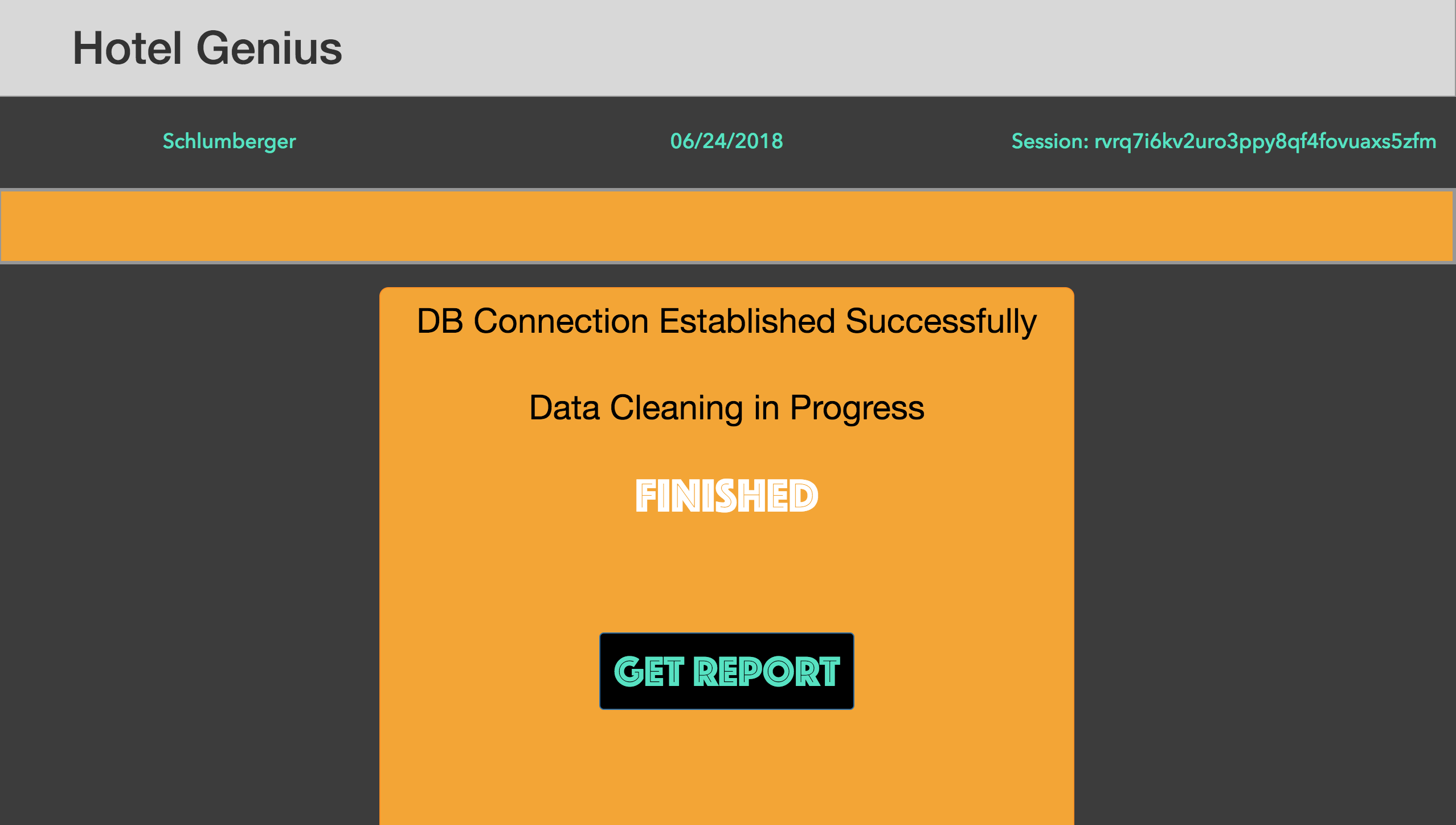
–please note this application is built for a full screen size on google chrome, other browsers experience difficulties. Press start after filling in the details.



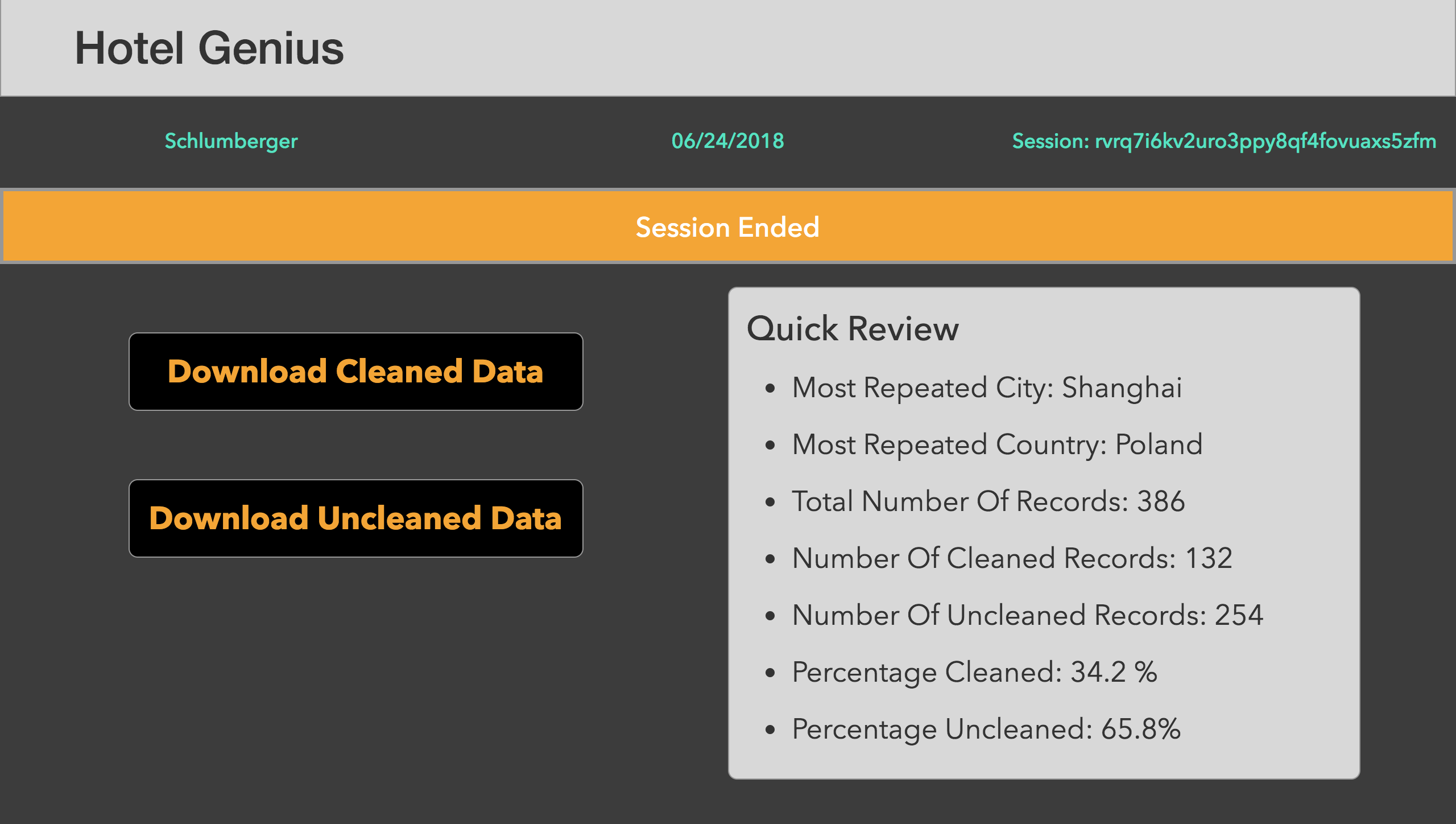
1. The upload page opens, please choose the csv file, then press clean.



1. Wait till the cleaning is finished, then the new page shall appear, press get report to get final report.



1. In the final page, you can view some statistical analysis on the data, as well as the ability to hit buttons to obtain the whole cleaned dataset or the uncleaned dataset.



1. Pressing any of the download buttons will automatically download a csv file.
2. Finally, the database admin interface by adding in “/admin” to the url, e.g. <http://127.0.0.1:8000/admin/>
3. In the login form, a special super user has been created for the purpose of this report, with username: ilirgashi and password: baml1234
4. After logging in, the user can have all access to all database objects to manually change or delete or add new instances using this admin interface.

