Implementation Document

for

The Dorm Room Dealer

Version 1.1

Prepared by Team TechJedis

Group 15: **Group Name: TechJedis**

Akhil Sagwal	**200078**	**asagwal20@iitk.ac.in**
Akshat Gupta	**200085**	**akshatg20@iitk.ac.in**
Devesh Shukla	**200322**	**devesh20@iitk.ac.in**
Milan Anand Raj	**200584**	**manandraj20@iitk.ac.in**
Sarthak Motwani	**200887**	**sarthakm20@iitk.ac.in**
Rahul Rustagi	**200756**	**rrustagi20@iitk.ac.in**
Akshat	**200080**	**akshat20@iitk.ac.in**
Vaibhav Chirania	**211134**	**vaibhavc21@iitk.ac.in**

Course: **CS253**

Mentor TA: ***Sumit***

Date: **20th March, 2023**

CONTENTS II

REVISIONS II

1 IMPLEMENTATION DETAILS 1

2 CODEBASE 3

3 COMPLETENESS 5

APPENDIX A - GROUP LOG 7

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
1.1	Akshat Gupta Rahul Rustagi Akhil Sagwal Akshat Sarthak Motwani Milan Anand Raj Devesh Shukla Vaibhav Chirania	This version contains details of the programming language, framework, libraries, database system, build system, that we have used to implement our software, and also details on how to navigate the codebase.	20/03/23
1.2	Akshat Gupta Rahul Rustagi Akhil Sagwal Akshat Sarthak Motwani Milan Anand Raj Devesh Shukla Vaibhav Chirania	The document was revised and updated according to the latest version of our web application.	23/04/23

1 Implementation Details

Provide the details of programming languages, frameworks, libraries, database systems, build systems, etc. that you have used for implementing your software.
Provide a brief justification of choosing any tool by stating its benefits over the alternatives.

- Programming Languages
  - Python
  - HTML
  - CSS

- Backend Framework
  - Django

- Frontend
  - HTML
  - CSS
  - CSS Bootstrap
  - Django's Template Functionality
  - Inline JavaScript

- Database System
  - SQLite (db.sqlite3)

- Libraries
  - datetime
  - os
  - sys

Why we used Django to implement our software:

- Django web development Framework is Simple
  Django fulfills the main purposes by covering the basics and simplifying the development process so that you can focus on the more complex and unique features of your project.

- Django Runs on Python
  The Django framework is based on Python, which is a dynamic and high-level programming language.
  Python is also more interactive than most other languages and helps you focus on solving tasks rather than on syntax. Python also has extensive libraries, which make it easy to learn, implement, and adopt.

- Django is Highly-Secure and Up-To-Date
  Django is regularly updated with security patches even in case you are using an older version of the framework. Django has an LTS (Long-term Support) version.

- Django is Backward-Compatible
  Django offers complete backward-compatibility with reusable components such as interfaces, common features, and formats of previous versions. Additionally, Django has a well defined roadmap and descriptions.

Why did we use HTML and CSS for our software's frontend :

- Simplicity
  HTML and CSS are the bare minimum needed to form an interactive website like ours. The learning curve for these is not very steep at the beginning, and that allowed us to get a decent frontend for our website in less time.

- Speed
  HTML and CSS are lightweight and fast, making them an excellent choice for building web applications that need to load quickly. This feature enables websites made using HTML and CSS to load quickly on mobile devices.

- Compatibility
  HTML and CSS are supported by all modern web browsers. Using only these technologies ensures compatibility with a wide range of devices, including desktop computers, laptops, tablets, and mobile devices.

Why did we use Javascript for our software's frontend :

- Simplicity
  Inline JavaScript may be used for small, simple scripts that are only needed on a single page and do not warrant a separate external script file. For example, a simple form validation script that checks if a form has been filled out correctly may be included inline.

- Specific functionalities
  If a specific page requires unique functionality that is not used elsewhere in the application, inline JavaScript may be used to implement that functionality directly within the HTML document. This can help keep the code localized and easier to manage.

- Quick Fixes
  During rapid prototyping or for quick fixes on a live site, inline JavaScript can be a convenient way to make changes without having to modify external script files or perform a full deployment. However, it's important to ensure that proper development practices are followed once the changes are finalized.

2 Codebase

Provide the link to your github repository.
Mention briefly how to navigate the codebase.

- Github link : DormRoomDealer

- Understanding the CodeBase directory structure:

- The workspace is “DORMROOMDEALER-MAIN”
- The project is “DRD”

- The ***media*** directory contains images of the items that are uploaded by the sellers as a description of their product.

- The ***static*** directory contains the css files used for styling the corresponding .html files. The ***img*** directory inside it contains images used in .html files i.e. as website stock images.

- Different apps in the project are
  - *accounts*
  - *items*

- The ***accounts*** application contains :
  - *urls.py* contains the list of possible paths that are concerned with the login / logout / register of a user.
  - *views.py* contains the function definitions to handle POST / GET requests generated by a user like login / register / send_mail / logout
    - *ilogout* and *logout* are different functions implemented where *ilogout* corresponds to the logout function available when accessed through items page and *logout* is accessed when at home.
  - *models.py* contains the backend model used in accounts app i.e. a field of username and contact is being stored in the database through this app.

- The ***items*** application contains :
  - *urls.py* contains the list of possible paths that are concerned with the operations like additem, biditem, validate, item/biditem.
  - *views.py* contains the function definitions to handle POST / GET requests generated by a user like additem, biditem, validate.
    - biditem and item/biditem are different URLs allowed to access. They call the same function ‘biditem’. They are different buttons and separate.
  - *models.py* contains the backend model used in items app i.e. a form collecting details like image of item / Name of item / Base Price of item / etc. is being stored in the database through this app.

- The ***assets*** directory contains css, fonts, img file used for rendering the frontend files

- The **DRD** directory contains:
  - *settings.py* is a central location to configure various settings and parameters for the project. It contains an INSTALLED APPS setting to specify the apps installed in the

django project. The TEMPLATES setting in settings.py is used to configure the template engines used by our Django project. It also contains the email and app-specific password of the host user.
    - *urls.py* includes urls from two other URLconf modules, accounts.urls and items.urls.

- The ***templates*** directory contains the .html files used for rendering when received a GET request:
    - *home.html* is the html file for the home webpage, which displays the live products.
    - *additem.html* is the html file for the add item webpage.
    - *biditem.html* is the html file for the webpage of bidding a particular item.
    - *dashboard.html* is the html file for the dashboard webpage, which shows the products uploaded by user (live, past and future) and products bidded by user.
    - *future.html* is the html file for the web page displaying future auctions.
    - *login.html* and register.html are for login and register web pages respectively.
    - *notification.html* and *notification2.html* are for displaying appropriate notifications to the user.
    - *myprofile.html* contains the frontend rendered when edit profile button is clicked
    - *register.html* contains code rendered when the user wants to register a new account
    - *login.html* contains template rendered when login is prompted

    - It also contains necessary html files for the webpages of password change.

- ***manage.py*** file is used to run the development server.

- ***tests.py*** file contains unit tests to check the correctness of the implemented functions.

- ***db.sqlite3*** is the database for storing the data of the accounts details and items.

- ***README.md*** is the read-me file that has necessary details regarding how to run the application.

How to run our application locally:

1. Clone the Github repository using the following command:
   git clone https://github.com/CS253-The-Dorm-Room-Dealer/dormroomdealer.git

2. Run the following commands in the same sequence as described below:

   a) **python manage.py migrate** (to pull and integrate new changes in the database model)

   b) **python manage.py runserver** (runs the application and hosting the server on localhost)

   The website starts running at localhost:8000/. (default port)

3 Completeness

Provide the details of the part of the SRS that have been completed in the implementation.

Provide the future development plan by listing down the features that will be added in the (may be hypothetical) future versions.

Currently Implemented :

- Add an Item functionality: Putting up a product for auction
- Login / Signup functionality
- Bid Item functionality
- Dashboard with Bid history and My profile
- My Profile : containing profile pic / contact number / email id
- Bid History : contains information about previous bids / current bids / future bids / items sold
- Mail functionality: Mail sent to buyer and seller when the seller accepts the bid of a user. Also, sellers are notified via email whenever someone bids for their product.
- Start Date and End Date of a Auction : Enables the possibility of planning a future auction
- Accept Bid functionality : Allows the seller to dynamically accept a bid he / she likes
- Reset Password through email link
- Filter auction products through tags: Users can use the search bar on the home screen to filter products based on tags.
- Bidders cannot bid less than the base price (minimum price).
- Bidders cannot bid above a certain price (maximum price).
- Seller cannot bid on his/her own product.
- End auction dynamically before the end date/time

Functionalities that were part of SRS, but have been implemented differently:

- The display of a product shall contain a clickable link containing the contact information of the seller. Each bidding history shall contain a link to access the contact details of the seller. *Modification*: *Seller and bidder contact information are shared via mail when the product has been sold.*

- The system shall allow the seller to either accept or reject the bid. *Modification: Seller has two options -*

  a) Wait for the auction to end as per schedule, this will lead to the overall highest bidder winning, or

  b) End the auction prematurely, this will lead to the current highest bidder winning the auction. Since there is no direct 'accepting' or 'rejecting' of individual bids, all the functionalities regarding notifications related to 'accepting' or 'rejecting' bids are now defunct.

Functionalities that were part of SRS, but not yet implemented (will be part of the future development plan):

- Allow the user to edit his personal information. *(done)*
- Segmentation into multiple pages if too many products listed. Each page must not contain more than 10 items. (*done)*
- Sort products using price. *(done)*
- Allow the bidder to zoom in on the picture of the product. *(seems unnecessary)*
- Ask the bidder for confirmation before accepting the bid. *(done)*
- The system shall allow the user to republish her product in the portal through the selling history page.

Future Development Plan:

- ☐ Allow the bidder to cancel their bid before the auction ends.
- ☐ We will make our app more responsive by fixing the drop-down navigation bar in the mobile design.
- ☐ Add a “Rating” based metric to classify seller based on his successful bids in the past
- ☐ Send mails through a “task-scheduler” which helps in improving latency.
- ☐ Allow sellers to edit and republish their unsold products.
- ☐ Provide a functionality (possibly in the form of a button) for the seller and user to verify payment.
- ☑ ~~Past / Sold bids to be removed beyond 2 weeks to maintain storage in the database.~~
- ☑ ~~Roll Number field addition to style the app towards campus junta.~~
- ☑ ~~Re-designing the home page to include the future auctions on the home page only.~~
- ☑ ~~Re-designing the dashboard, such that a user’s live products appear on top.~~

Appendix A - Group Log

<Please include here all the minutes from your group meetings, your group activities, and any other relevant information that will assist in determining the effort put forth to implement your software>

Team Member	Work Done
Akshat	• Worked in the front-end. Implemented the main html templates and the corresponding css files.
Sarthak Motwani	• Worked in the backend for implementing views functions in both the accounts and views directory. • Worked in coordination with frontend developers to make backend functions consistent with frontend. • Worked in the Implementation doc.
Rahul Rustagi	• Initialised the basic structure of the code. • Worked with frontend and backend developers to create models for the database. • Worked in the Implementation doc.
Vaibhav Chirania	• Worked in the backend for implementing the limit of the bids and worked parallelly with other developers in making the function consistent. • Worked in the implementation doc
Milan Anand Raj	• Worked in the backend for implementing views functions in both the accounts and views directory. • Worked in coordination with frontend developers to make backend functions consistent with frontend. • Worked in the Implementation doc.
Akshat Gupta	• Worked in the backend for implementing views functions in both the accounts and views directory. • Worked in coordination with frontend developers to make backend functions consistent with frontend. • Worked in the Implementation doc.
Akhil Sagwal	• Worked in the front-end. Implemented the main html templates and the corresponding css files. • Worked in the Implementation doc.

Devesh Shukla	● Initialised the basic structure of the project. ● Worked in the backend for implementing the email functionality ● Worked in implementation doc

Note: Despite the above work segregation as mentioned, all of us have contributed in almost all the parts of the implementation whenever it was necessary.

DATE	Members Present	Topic of Discussion
13/02/2023	All members were present + TA	● Discussed appropriate frameworks that could be used for the front-end and back-end development.
25/02/2023	All members were present	● Distributed work on frontend and backend
05/03/2023	All members were present	● Basic frontend of the software was completed before this meeting. Met online to discuss further improvements in the frontend and also discussed some implementation issues related to backend development.
08/03/2023	All members were present	● Discussed how to implement the feature of sending emails to the users and also, some improvements that had to be made to the backend code to handle any unexpected inputs that the user might enter, which earlier were leading to website crash.
13/03/2023	All members were present	● A basic functioning version of the website was ready. We discussed some other features that we could incorporate in this version, and few changes related to the application's frontend were also talked about.

20/03/2023	All members were present + TA	● Presented the implemented project to the TA. ● Noted the necessary suggestions made by the TA to be implemented in the further versions of the project.