

Cairo University Faculty of Engineering

Department of Computer Engineering

**Webby**

**A logo with a circle and a circle in the middle

Description automatically generated**

A Graduation Project Contribution Report Submitted to

Faculty of Engineering, Cairo University

in Partial Fulfillment of the requirements of the degree of

Bachelor of Science in Computer Engineering.

### Presented by

Shredan Abdullah Kamal

### Supervised by

Dr. Khale Suradi

July 2024

All rights reserved. This report may not be reproduced in whole or in part, by photocopying or other means, without the permission of the authors/department.

# Abstract

The Webby project is Webby revolutionizes website development for users with limited IT knowledge by leveraging machine learning and natural language processing (NLP). This project mitigates the technical challenges of website creation through an intuitive graphical user interface (GUI) that allows users to describe their requirements in natural language. These descriptions are then translated into fully functional websites with integrated databases.

Our backend team concentrated on extracting essential database components such as entities, attributes, primary keys, relationships, and cardinalities from user input. This information was used to generate SQL queries, create databases, and develop backend CRUD operations using Django. The development process involved extensive research, dataset sourcing, the implementation of both rule-based and machine learning methods, and the creation of association logic to link attributes to entities. A BiLSTM model was developed for entity extraction purposes.

The frontend development was equally comprehensive, focusing on creating a user-friendly and visually appealing interface. The GUI was designed to be intuitive, enabling users to easily input their requirements and view the real-time translation of their descriptions into website components. The frontend team utilized modern web technologies and frameworks to ensure a seamless user experience. Features included dynamic form fields, drag-and-drop components, and real-time previews, all aimed at making website creation as straightforward as possible.

Additionally, the frontend was integrated with the backend to facilitate smooth data flow and interaction. This integration ensured that user inputs were accurately captured and processed by the backend systems, resulting in the automatic generation of websites tailored to user specifications.

The development was conducted using Python, Django, and various machine learning libraries for the backend, while the frontend utilized HTML, CSS, JavaScript, and relevant frameworks to create an efficient and user-friendly interface. The collaboration between the frontend and backend teams ensured the system's robust functionality and ease of use.

# Table of Contents

Contents

[Presented by 1](#_Toc171882655)

[Supervised by 1](#_Toc171882656)

[Abstract 2](#_Toc171882657)

[Table of Contents 3](#_Toc171882658)

[List of Figures 5](#_Toc171882659)

[List of Abbreviation 6](#_Toc171882660)

[Chapter 1: Work Done 7](#_Toc171882661)

[1.1. Website Template Retrieve 7](#_Toc171882662)

[1.2. Linking with the Backend 8](#_Toc171882663)

[Chapter 2: System Design and Architecture 11](#_Toc171882664)

[2.1. Overview 11](#_Toc171882665)

[2.2. System Architecture 11](#_Toc171882666)

[2.2.1. Block Diagram 11](#_Toc171882667)

[2.3. Module 1: Data Gathering and Preprocessing 13](#_Toc171882668)

[2.3.1. Functional Description 13](#_Toc171882669)

[2.3.2. Modular Decomposition 13](#_Toc171882670)

[2.3.3. Design Constraint 14](#_Toc171882671)

[2.4. Module 2: Retrieving 15](#_Toc171882672)

[2.4.1. Functional Description 15](#_Toc171882673)

[2.4.2. Modular Decomposition 15](#_Toc171882674)

[2.4.3. Design Constraints 15](#_Toc171882675)

[2.5. Module 3: Generation 16](#_Toc171882676)

[2.5.1. Functional Description 16](#_Toc171882677)

[2.5.2. Modular Decomposition 16](#_Toc171882678)

[2.5.3. Design Constraints 16](#_Toc171882679)

[Chapter 3: System Testing and Verification 17](#_Toc171882680)

[3.1. Testing Setup 17](#_Toc171882681)

[3.2. Testing Plan and Strategy 17](#_Toc171882682)

[3.3. Module Testing 17](#_Toc171882683)

[3.4. Integration Testing 19](#_Toc171882684)

[3.5. Testing Schedule 19](#_Toc171882685)

[Chapter 4: Conclusions and Future Work 20](#_Toc171882686)

[4.1. Faced Challenges 20](#_Toc171882687)

[4.2. Gained Experience 20](#_Toc171882688)

[4.3. Conclusions 21](#_Toc171882689)

[4.4. Future Work 21](#_Toc171882690)

[References 22](#_Toc171882691)

# List of Figures

Figure 2.1:Frontend Block Diagram………………………………………………………[1](#_bookmark9)5

Figure 2.2:Frontend Part-Dataset Gathering module1……………………………...…16

Figure 2.3:Frontend Part-Classification&Retrieve module2…………………………..16

Figure 2.4:Frontend Part- Generation module3………………………………………..16

# List of Abbreviation

NLP Natural Language Processing

HTML Hyper Text Markup Language

CSS: Cascading Style Sheets

# Chapter 1: Work Done

Introduction

This chapter provides an in-depth my work contribution for the development of Webby.

## Website Template Retrieve

At first since the dataset was not ready for us to use, we decide to gather it but ourselves, so we searched for website templates that are suitable for our task, where it’s specifications were no frontend framework is been used, only HTML, CSS, JS files, so we decide to initially cover 30 categories and each member in the frontend sub team will take 15 category to gather 15 category, so I gather website templates about [gifts, pets, games, electronics, nature, home interior, fashion, business, travel, wedding, dating, charity, photography, industry, magazines]

I downloaded them from <https://www.free-css.com/> then I wrote functions to deal with them to extract the zipped files then reach the folder that holds and website and delete any intermediate files as we agreed on certain folder hierarchy.

For the description generation we used image to text converter like <https://pallyy.com/tools/image-description-generator> and then modify on the output manullay by ourselves to ensure the visuals accuracy.

Now our dataset is completed, my other mate implemented the classification part and the It’s output is input of the retrieve module which I took it’s responsibility, the main idea of the retrieving module is that we want to retrieve the closet website based on the description entered by the user, so the returned category(class) from the classification I take this folder and loop over all its description and do the following:

-Apply the preprocessing part that include tokenizing the text into words. converting all words to lowercase. removing stopwords. lemmatizing the words.

-Generates word embeddings for the descriptions and the user input using a pre-trained word2vec model. The embeddings represent the text in a high-dimensional vector space.

I used the word embedding as a feature because it holds the meaning of the word, not like other feature extractions methods known in the NLP like bag of words, n-grames, and so.

After I had the embeddings of each template and embeddings for the user input. I then calculate the cosine similarity between the user input and each description. To get the closest after that to ensure that we are retrieving the closet template in terms of the stored description compared to that of the user.

Also, I want to mention that I used the cosine similarity measure after I compared and test the results of most of the other distance measures like manhatten distance, equlidian distance and others.

Finally, the output in our hand is the closet template and now it’s ready for the modification part.

## Linking with the Backend

The linking or the integration with backend was the challenge that we’ve been asking for while we first introduce our project idea, for this task we needed a web framework that could link between backend and frontend, and after searching we chose Django. Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. It follows the model-template-views (MTV) architectural pattern, which is a variant of the model-view-controller (MVC) pattern. Django provides a robust set of features to help developers build web applications quickly and efficiently:

- ORM (Object-Relational Mapping): Django comes with a powerful ORM that allows developers to interact with databases like PostgreSQL, MySQL, SQLite, and Oracle using Python code instead of writing raw SQL queries. This ORM supports complex queries, joins, and transactions, making database management seamless.

- Built-in Admin Interface: Django includes a built-in admin interface that allows developers to manage application data through a web-based interface without needing to write additional code. This admin interface is highly customizable and can be tailored to fit the needs of any project.

- URL Routing: Django’s URL routing system enables developers to map URLs to views in a clean and intuitive manner. This routing system supports named URL patterns and dynamic URL segments, making it easy to create readable and maintainable URL structures.

- Security Features: Django includes several security features out of the box, such as protection against SQL injection, cross-site scripting (XSS), cross-site request forgery (CSRF), and clickjacking. These features help developers build secure web applications without needing to implement additional security measures.

- Template System: Django's template system allows developers to define HTML templates with dynamic content placeholders. It supports template inheritance, enabling the creation of reusable templates and reducing code duplication.

- Scalability and Performance: Django is designed to handle high-traffic applications and can be scaled horizontally by adding more web servers. It supports caching mechanisms and can integrate with various caching backends to improve performance.

- Extensibility: Django’s architecture is highly extensible, allowing developers to add custom middleware, template tags, and filters. The extensive ecosystem of third-party packages further enhances Django’s capabilities, providing solutions for common tasks like user authentication, RESTful API development, and social media integration.

- Comprehensive Documentation: Django’s comprehensive documentation provides detailed guides, tutorials, and references for all aspects of the framework. This documentation, along with an active community, ensures that developers can find the support and resources they need to build robust applications.

And all of those features I worked on by myself in our project not just a raw description of what Django introduces and by leveraging Django’s features, our project achieved seamless integration between the backend and frontend, allowing for efficient data management, dynamic content rendering, and a scalable architecture.

But we faced a critical problem here where the fact the retrieve the website based on a user input so how we achieve this dynamic link?

First let’s introduce the structure of a Django project that I and my mate learned:

**Project Structure**

* **manage.py**: A command-line utility that lets us interact with our Django project. It is used for various tasks such as running the development server, creating migrations, and interacting with the database.
* **settings.py**: Contains configuration settings for the entire Django project, including database configuration, installed applications, middleware, static files, and more.
* **urls.py**: Defines the URL patterns for the project. It maps URLs to views, enabling Django to route HTTP requests to the appropriate view function or class.

**App Structure**

Django encourages the development of applications as reusable modules. Each application contains the following components:

* **models.py**: Defines the data models for the application. Models represent the structure of your database tables and include fields, relationships, and methods for interacting with the data.
* **views.py**: Contains view functions or classes that handle HTTP requests and return HTTP responses. Views process data, interact with models, and render templates.
* **urls.py**: Defines URL patterns specific to the application. These patterns are included in the project's main urls.py to route requests to the appropriate views within the app.
* **admin.py**: Registers models with the Django admin interface, allowing you to manage application data through a web-based interface.
* **apps.py**: Contains application configuration, such as the name and label of the app. This file is used by Django to configure and initialize the app.
* **migrations/**: A directory that contains database migration files. Migrations are used to apply changes to the database schema, such as creating or modifying tables.
* **templates/**: A directory that contains HTML templates for the application. Templates are used to render dynamic content.
* **static/**: A directory that contains static files such as CSS, JavaScript, and images.

**How They Work Together**

* **Request Handling:**

When a user makes an HTTP request, Django uses the urls.py file to match the requested URL to a view.

The view function or class in views.py is called to handle the request. It can process input data, interact with models, and render templates.

* **Data Models:**

The models.py file defines the data structure for the application. Each model class represents a database table.

Views can query the database using the models to retrieve or modify data.

* **Templates:**

The templates/ directory contains HTML files with placeholders for dynamic content.

Views use templates to generate HTML responses by passing data to the template context.

* **Static Files:**

The static/ directory contains static assets like CSS and JavaScript files.

These files are served to the client to enhance the user interface and user experience.

* **Admin Interface:**

The admin.py file registers models with the Django admin site, providing a web-based interface for managing application data.

This interface allows administrators to perform CRUD operations on the data without writing additional code.

Now after illustrating the Django code structure and how it works let’s talk about how I managed to link the database with frontend part dynamically without a prior information about the tables name or fields, how I generalize views.py that is responsible for rendering data from to database to the HTML files, I made an assumption that most of our categories sell or buy some products, so mainly there will be tables for sellable objects and those sellable objects most of the time contains fields related to the price the amount the availability and others like that, so I wrote general code in the views.py that well be used for every generated website, that loops over the tables and fields that is in models.py to search for sellable tables and retrieve it to be rendered in the HTML template and by that criteria we could dynamically retrieve data from database without the prior knowledge of its content and without the need to implement a views.py code for each template.

# Chapter 2: System Design and Architecture

This chapter presents the detailed design and architecture of the project. It includes the main modules, their interactions, and the methodologies used. The aim is to provide a comprehensive understanding of how the project was realized, with enough details to allow replication and improvement.

## Overview

The project was designed to simplify website development using machine learning and natural language processing (NLP). The system was developed with the assumption that users would provide text input describing their desired Frontend part. The Frontend system is divided into three main modules:

* **Gathering Dataset with Descriptions**
* **Classification and retrieving of the template**
* **Generating Modified template**

## System Architecture

### Block Diagram



*Figure 2.1: Frontend Part Block Diagram*

Module1: Dataset Collecting & Preprocessing

A white arrow pointing to the right

Description automatically generated

*Figure 2.2: Frontend Part-Dataset Gathering module1*

Module2: Classification & Retrieving

A screenshot of a computer

Description automatically generated

*Figure 2.3: Frontend Part-Classification & Retrieve module2*

Module3: Generation

A diagram of a function

Description automatically generated

*Figure 2.4: Frontend Part-* *Generation module3*

## Module 1: Data Gathering and Preprocessing

### Functional Description

Making our own dataset as there was no such a dataset was published for websites and manually giving a description for each template website.

### Modular Decomposition

**Templates gathering:**

* + - * Collecting website templates for 15 categories.
      * Each website template contains html\css\js files, some of them contain bootstrap and jquery files.

**Descriptions formulation:**

* + - * We open each index.html in the dataset to write keywords features that describe the website appearance well.
      * We used image to text generator and pass to it the website image and then enhance the output description.

**Dataset Preparing:**

* + - * **Extract zipped folders**. Example: Extracted Animals & Pets Free Website Template - Free-CSS.com.zip into C:\Users\Tobi\Desktop\pets\Animals & Pets Free Website Template - Free-CSS.com
      * **Then delete those zipped folders**. Example: Deleted C:\Users\Tobi\Desktop\charity\Charitable Organization Free Website Template - Free-CSS.com.zip
      * **Move inner folders to there main folder**. Example: Moved C:\Users\Tobi\Desktop\pets\Animals & Pets Free Website Template - Free-CSS.com\animals-and-pets to C:\Users\Tobi\Desktop\pets\animals-and-pets
      * **Move and Delete Intermediate Folders**. Example: Moved C:\Users\Tobi\Desktop\fashion\Affection Free Website Template - Free-CSS.com\affection to C:\Users\Tobi\Desktop\fashion\affection

### Design Constraint

**Dataset Constraints**

* Each website templates consists of html\css\js files, No frameworks are used like Reactjs or Angular.
* Each description must include the visuals of the site, not frontend specifications, Only visuals that the normal user can figure out.
* The description must be not too long to not affect the processing time.
* Both of us must put the dataset folders in a the same hierarchy.

## Module 2: Retrieving

### Functional Description

Retrieving the closet website template based on the description taken from the user using the cosine similarity and word embeddings as a feature extracted for both the user input and the saved dataset descriptions.

### Modular Decomposition

**Preprocesses a set of text descriptions:**

* + - * Tokenizes the text into words.
      * Converts all words to lowercase.
      * Removes stopwords (common words like "the", "and", "is", etc.).
      * Lemmatizes the words (converts words to their base form,"running" to "run").

**Generates word embeddings for the descriptions and the user input:**

* + - * It generates word embeddings for a list of texts using a pre-trained word2vec model. The embeddings represent the text in a high-dimensional vector space.

**Computes the cosine similarity between the user input and each description**.

**Retrieves the top K descriptions:**

* + - * It retrieves the top K descriptions that are most similar to the user input and maps them to corresponding website folder.

### Design Constraints

**Text Data Quality and Format**

* **Cleanliness**: The text data must be clean and properly formatted for the preprocessing steps to be effective.
* **Consistency**: Text descriptions and user input should be consistent in terms of language and style to ensure accurate preprocessing and embedding.

**Computational Resources**

* **Memory**: The use of large word embeddings like word2vec can be memory-intensive, so sufficient memory must be available.
* **Processing Power**: Generating embeddings and calculating cosine similarities for large datasets can require significant processing power.

## Module 3: Generation

### Functional Description

Modify specific elements in an HTML content string to generate the desired template based on provided details. It allows dynamic customization of HTML content based on user-provided details, enabling easy updates to key parts of the HTML without manually editing the file.

### Modular Decomposition

**HTML Modification:**

* + - * Uses regular expressions to find and replace specific HTML tags with updated content.
      * Modifies HTML content by updating the title, main header, location, and hotline information based on provided details.

**Feedback Collection:**

* + - * Collects user feedback to review and correct extracted details from a description.
      * Prompts the user to review extracted details, integrates corrected information, and returns updated details for improved accuracy.

### Design Constraints

**Text Data Quality and Format**

* **Regex Accuracy**: The use of regular expressions for HTML modification assumes well-formed HTML and specific patterns.
* **User Input**: Assumes that user-provided details for feedback are accurate and correctly formatted.

# Chapter 3: System Testing and Verification

## Testing Setup

I used self-unit tests to test the model after each addition to its implementation.

## Testing Plan and Strategy

The tests are compared to what it gives vs what is expected.

## Module Testing

#### **3.3.1. Module 1: Data Gathering and Preprocessing**

**Test Cases:**

* **Template Gathering:**
  + **Test Case 1:** Ensure website templates for 15 categories are collected.
    - **Procedure:** Verify that each category contains the expected number of website templates with HTML, CSS, and JS files.
    - **Expected Result:** templates are present and accessible.
* **Descriptions Formulation:**
  + **Test Case 2:** Ensure keywords and features accurately describe each website's appearance.
    - **Procedure:** Compare manually written descriptions and those generated by the image-to-text generator.
    - **Expected Result:** Descriptions accurately reflect the visual aspects of the websites.
* **Dataset Preparing:**
  + **Test Case 3:** Ensure that zipped folders are correctly extracted.
    - **Procedure:** Verify that extracted folders are in the expected directory structure.
    - **Expected Result:** All zipped folders are extracted without errors.
  + **Test Case 4:** Ensure that zipped folders are deleted after extraction.
    - **Procedure:** Check the directory for the presence of original zip files after extraction.
    - **Expected Result:** All zipped folders are deleted.
  + **Test Case 5:** Ensure that inner folders are moved to the main directory correctly.
    - **Procedure:** Verify the final directory structure after moving inner folders.
    - **Expected Result:** Inner folders are correctly moved to the main directory.
  + **Test Case 6:** Ensure that intermediate folders are deleted after moving inner folders.
    - **Procedure:** Check the directory for the presence of intermediate folders after moving.
    - **Expected Result:** Intermediate folders are deleted.

#### **3.3.2. Module 2: Retrieving**

**Test Cases:**

* **Preprocessing Text Descriptions:**
  + **Test Case 1:** Ensure tokenization, lowercasing, stopword removal, and lemmatization are performed correctly.
    - **Procedure:** Compare the preprocessed text with expected output.
    - **Expected Result:** Text is tokenized, lowercased, stopwords removed, and words lemmatized correctly.
* **Generating Word Embeddings:**
  + **Test Case 2:** Ensure word embeddings are generated correctly for text descriptions and user input.
    - **Procedure:** Verify the generated embeddings using a pre-trained word2vec model.
    - **Expected Result:** Embeddings accurately represent the text in a high-dimensional vector space.
* **Computing Cosine Similarity:**
  + **Test Case 3:** Ensure cosine similarity is calculated correctly between user input and each description.
    - **Procedure:** Compare calculated similarities with expected values.
    - **Expected Result:** Cosine similarities are accurate.
* **Retrieving Top K Descriptions:**
  + **Test Case 4:** Ensure the top K most similar descriptions are retrieved accurately.
    - **Procedure:** Verify the list of retrieved descriptions and their corresponding website folders.
    - **Expected Result:** The top K descriptions are the most similar to the user input.

#### **3.3.3. Module 3: Generation**

**Test Cases:**

* **HTML Modification:**
  + **Test Case 1:** Ensure the title tag is correctly updated based on provided details.
    - **Procedure:** Verify the modified HTML content for correct title tag.
    - **Expected Result:** Title tag matches the provided details.
  + **Test Case 2:** Ensure the main header is correctly updated.
    - **Procedure:** Verify the modified HTML content for correct header tag.
    - **Expected Result:** Header tag matches the provided details.
  + **Test Case 3:** Ensure location information is inserted correctly.
    - **Procedure:** Verify the modified HTML content for correct location information.
    - **Expected Result:** Location tag matches the provided details.
  + **Test Case 4:** Ensure hotline information is inserted correctly.
    - **Procedure:** Verify the modified HTML content for correct hotline information.
    - **Expected Result:** Hotline tag matches the provided details.
* **Feedback Collection:**
  + **Test Case 5:** Ensure user feedback is collected and integrated correctly.
    - **Procedure:** Prompt user to review and correct extracted details, then verify the updated details.
    - **Expected Result:** User feedback is accurately integrated into the final details.

## Integration Testing

After the integration of the models, the interface is manually tested for correctness.

## Testing Schedule

It is done as mentioned after the implementation of each model and whenever a model is finished it is integrated with its next module and been tested together along the way.

# Chapter 4: Conclusions and Future Work

## Faced Challenges

The amount of research that I spent to understand Django and how to link backend with frontend, specially that I was have a prior knowledge about Django, also the fact that the idea of the project is considered new idea that never go to the light before so searching for papers, reference or even dataset was so hard that we barely found related information, so most of the work was originally from out thinking and conclusions together and that’s was a hard challenge itself.

## Gained Experience

I gained experience in several key areas of software development, machine learning, and web technologies. Here are the highlights of our learning journey:

* Natural Language Processing (NLP):

Text Preprocessing: We became proficient in text preprocessing techniques such as tokenization, stopword removal, and lemmatization, which are essential for preparing textual data for machine learning models.

Text Embeddings: We explored various word embedding techniques such as Word2Vec to convert textual data into numerical representations that our machine learning models could work with.

* Machine Learning:

Model Training and Evaluation: We gained experience in training machine learning models to generate website code from user descriptions. This involved evaluating model performance too.

* Integration with Web Technologies: I learned how to integrate machine learning models into a web application using Django, enabling real-time interaction between the user and the model.
* Web Development with Django:

Project Structure: I became adept at organizing a Django project, understanding the roles of various files such as views.py, models.py, forms.py, and urls.py.

Template Rendering: I learned to dynamically generate and serve HTML content using Django templates, allowing for a personalized user experience based on their input.

* Software Engineering Practices:

Modular Design: I practiced designing our application in a modular way, separating concerns and making the codebase easier to maintain and extend.

Error Handling and Debugging: I improved my skills in identifying, diagnosing, and fixing bugs, as well as implementing robust error handling mechanisms to ensure a smooth user experience.

## Conclusions

ML-driven website generation, particularly using machine learning and NLP techniques, has significantly transformed how websites are created and customized. This project showcases how these advanced technologies can automate the translation of user descriptions into functional website code, streamlining the web development process. The integration of Django for backend linkage further demonstrates the potential for creating dynamic, database-driven websites efficiently. The success of this project highlights the feasibility of AI-driven solutions in web development, paving the way for more accessible and user-friendly tools that empower even non-technical users to create personalized and professional websites with ease. This innovation is poised to revolutionize the web development landscape, making high-quality website creation faster, more efficient, and more intuitive.

## Future Work

* + - Adding more and more categories in the dataset to include more options to users.
    - Enlarging the views.py criteria to include more general tables from the database.

# References

[1] <https://dzone.com/articles/the-power-of-template-based-document-generation-wi>

[2] <https://domino.ai/data-science-dictionary/feature-extraction>

[4] <https://www.analyticsvidhya.com/blog/2022/05/a-complete-guide-on-feature-extraction-techniques/>

[5] <https://serokell.io/blog/word2vec>

[6] <https://docs.djangoproject.com/en/dev/ref/contrib/admin/#overriding-admin-templates>

[7] <https://developer.mozilla.org/en-US/docs/Web/HTML>