

ABSTRACT

This project aims to make use of some python graphic libraries to help out the software developers in drawing Data Flow Diagrams.

It is always time consuming to generate this diagram in between the development processes. The alignment, data flow, size, all these things are literally confusing. So it will be helpful if we could cut out such issues.

This project “Programmed DFD” performs as an online tool that can be efficiently used by anyone who knows how to draw a DFD. It takes the details such as modules, functions and databases as input and finally renders the generated DFD image as output.

INTRODUCTION

ProDFD

ProDFD is a web application that aims to generate Data Flow Diagrams for simple software projects. It is accomplished with the help of Python Django framework and MongoDB database. As of now ProDFD can generate zero level, first level, and child diagram of first level DFDs at a single click which can be downloaded to the user's system. The one notable feature of ProDFD is, it does not force the users to signup into the system for the service. Instead, DFD generation process can be done easily without registration.

It wasn't easy to make this possible. Especially the data collection part and the drawing part. The initial risk was to connect Django to MongoDB as Django does not directly support NoSQL databases. So I had to use Djongo connector for this purpose. The data collection part is partially dynamic which made it more difficult to complete. Most of the parts are done with the help of JQuery and JavaScript. The next risk was difficulty in handling pixel values as it handles tons of numbers that are dependent on each other. For drawing purpose the Python graphics libraries PIL and CV2 are used.

PYTHON

Python is an interpreted ,object oriented ,high level programming language with dynamic semantics. Its high level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for rapid application development as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Python Life Cycle

The life cycle methods

The Python system defines a life-cycle for activities via predefined life-cycle methods. The most important methods are:

Step1: Definition

Python defines its classes with keyword 'class' which is defined in Python interpreter.

Step2: Initialization

When an instance of the class is created, `__init__` method defined in the class is called. It initializes the attributes for newly created class instance. A namespace is also allocated for object's instance variables. `__new__` method is also called where the instance gets created.

Step3: Access and Manipulation

Methods defined in a class can be used for accessing or modifying the state of an object. These are accessories and manipulators respectively. A class instance is used to call these methods.

Step4: Destrction

Every object that gets created, needs to be destroyed. This is done with Python garbage collection (that is reference counting).

Python's development cycle is dramatically shorter than that of traditional tools. In Python, there are no compile or link steps -- Python programs simply import modules at runtime and use the objects they contain. Because of this, Python programs run immediately after changes are made. And in cases where dynamic module reloading can be used, it's even possible to change and reload parts of a running program without stopping it at all. Figure shows Python's impact on the development cycle.

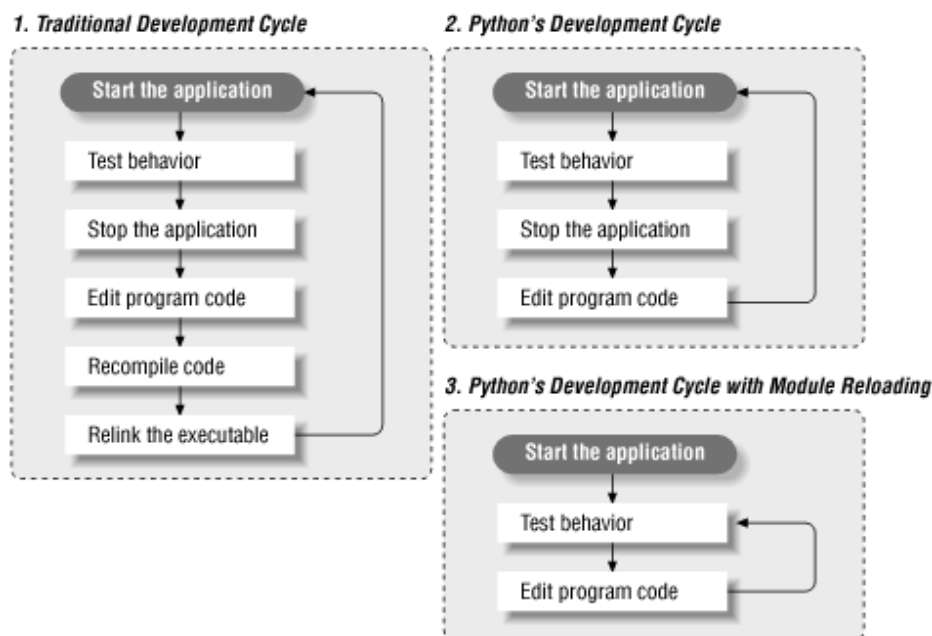


Figure : Development cycles

More generally, the entire development process in Python is an exercise in rapid prototyping. Python lends itself to experimental, interactive program development, and encourages developing systems incrementally by testing components in isolation and putting them together later. In fact, we've seen that we can switch from testing components (unit tests) to testing whole systems (integration tests) arbitrarily, as illustrated in following Figure.

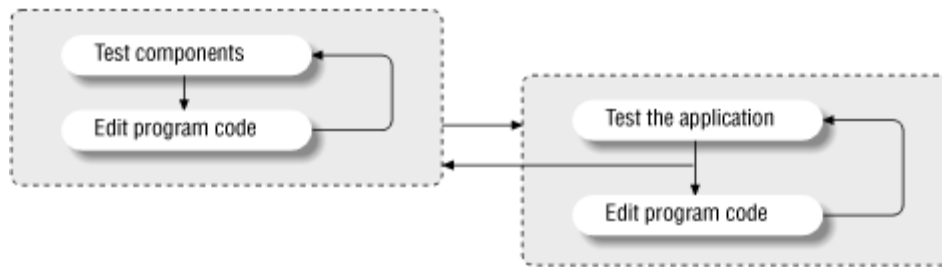


Figure : Incremental development

Django

Django is a Web framework written in Python. A Web framework is a software that supports the development of dynamic Web sites, applications, and services. It provides a set of tools and functionalities that solves many common problems associated with Web development, such as security features, database access, sessions, template processing, URL routing, internationalization, localization, and much more. Using a Web framework, such as Django, enables us to develop secure and reliable Webapplication very quickly in as standardized way, without having to reinvent the wheel. Django is one of the most popular Web frameworks written in Python. It's definitely the most complete, offering a wide range of features out of the box, such as a standalone Web server for development and testing, caching, middleware system, ORM, template engine, form processing, interface with Python's unit testing tools.

Django also comes with battery included, offering built-in applications such as an authentication system, an administrative interface with automatically generated pages for CRUD operations, generation of syndication feeds (RSS/Atom), sitemaps. There's even a Geographic Information System (GIS) framework built within Django. Django emphasizes reusability and "pluggability" of components, rapid development, and the principle of don't repeat yourself. Python is used throughout, even for settings files and data models. Django also provides an optional administrative create, read, update and delete interface that is generated dynamically through introspection and configured via admin models.

The development of Django is supported by the Django Software Foundation, and it's sponsored by companies like JetBrains and Instagram. Django has also been around for quite some time now. It's under active development for more than 12 years now,

proving to be a mature, reliable and secure Web framework. Django is now a thriving, collaborative open source project, with many thousands of users and contributors. While it does still have some features that reflect its origin, Django has evolved into a versatile framework that is capable of developing any type of website.

MongoDB

MongoDB is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability. MongoDB works on concept of collection and document.

Collection

Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

Document

A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

Djongo

Djongo is a smarter approach to pymongo programming. It is an extension to the traditional Django ORM framework. It maps python objects to MongoDB documents, a technique popularly referred to as Object Document Mapping or ODM.

Visual Studio Code

Visual Studio Code combines the simplicity of a source code editor with powerful developer tooling, like IntelliSense code completion and debugging. It supports macOS, Linux, and Windows.

With support for hundreds of languages, VS Code helps us to be instantly productive with syntax highlighting, bracket-matching, auto-indentation, box-selection, snippets, and more. Intuitive keyboard shortcuts, easy customization and community-contributed keyboard shortcut mappings lets us navigate code with ease.

SYSTEM ANALYSIS

PURPOSE

The time taken to draw a DFD is not negligible in between the software development process. Because, the alignment, data flow, size of each element, all these things are literally confusing, especially when it comes in between the stress of finishing up the project.

We know, there are many modules or APIs in different languages that can be used to plot such diagrams. So there shall be a programmatic solution to this problem if we could use these modules effectively. The purpose of the Programmed DFD is clear now.

AUDIENCE

The audience of this web application is any one who needs a Data Flow Diagram as part of their Software Development.

EXISTING SYSTEM

DFD can be drawn by using drawing tools provided by either Microsoft Office Word or some online tools that allows users to manually construct the diagram by using the available shapes. In either way drawing DFD is always time consuming as we need to take care of the data flow, size of elements, its alignment etc.

PROPOSED SYSTEM

The proposed system Programmed DFD enlightens the programmatic solution to this problem. It uses the graphic libraries PIL and CV2 provided by Python to plot the diagram, by collecting the details related to the project such as project name, modules, all the functions, table names etc.

There by this project aims to minimize the problem of time wastage at the documentation process. It generates level zero , one, and its child diagrams efficiently and effectively by investing only 5 to 10 minutes.

FUNCTIONAL REQUIREMENTS

Functional Requirements deal with defining software resource requirements and pre-requisites that need to be installed on a computer to provide optimal functioning of an application. These requirements or pre-requisites are generally not included in the software installation package and need to be installed separately before the software is installed. Functional specification is measured in terms of the output provided by the application. Requirement specification plays an important part in the analysis of a system. Only when the requirement specifications are properly given, it is possible to design a system, which will fit into required environment.

OPERATING SYSTEM	:	Windows10
FRONT END	:	Python, JavaScript, JQuery
BACK END	:	MongoDB
IDE	:	Microsoft Visual Studio Code

NON FUNCTIONAL REQUIREMENTS

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware. A hardware requirements list is often accompanied by a hardware compatibility list (HCL), especially in case of operating systems. An HCL lists tested, compatible and sometimes incompatible hardware devices for a particular operating system or application.

Hardware	:	Pentium
Speed	:	1.1 GHz
RAM	:	1GB , with 2GB swap
Hard Disk	:	20 GB
Key Board	:	Standard Windows Keyboard
Mouse	:	Two or Three Button Mouse
Monitor	:	SVGA

TOOL

FEATURES OF PYTHON

Python is a power full language, that is easy to implement and is use full in artificial intelligence, robotics, image processing etc.

Feature	Description
Easy to learn & use	Python is easy to learn and use. It is developer-friendly and high level programming language.
Expressive Language	Python language is more expressive means that it is more understandable and readable.
Interpreted Language	Python is an interpreted language i.e. interpreter executes the code line by line at a time. This makes debugging easy and thus suitable for beginners.
Cross-platform Language	Python can run equally on different platforms such as Windows, Linux, Unix and Macintosh etc. So, we can say that Python is a portable language.
Free & open source	Python language is freely available at <u>official web address</u> . The source-code is also available. Therefore it is open source.
Object-Oriented Language	Python supports object oriented language and concepts of classes and objects come into existence.
Extensible	It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our python code.
Large Standard Library	Python has a large and broad library and provides rich set of module and functions for rapid application development.
GUI Programming Support	Graphical user interfaces can be developed using Python.
Integrated	It can be easily integrated with languages like C, C++, JAVA etc.

FEATURES OF DJANGO

Rapid Development:

Django was designed with the intention to make a framework which takes less time to build web application. The project implementation phase is a very time taken but Django creates it rapidly.

Secure:

Django takes security seriously and helps developers to avoid many common security mistakes, such as SQL injection, cross-site scripting, cross-site request forgery etc. Its user authentication system provides a secure way to manage user accounts and passwords.

Scalable:

Django is scalable in nature and has ability to quickly and flexibly switch from small to large scale application project.

Fully loaded:

Django includes various helping task modules and libraries which can be used to handle common Web development tasks. Django takes care of user authentication, content administration, site maps, RSS feeds etc.

Versatile:

Django is versatile in nature which allows it to build applications for different-different domains. Now a days, Companies are using Django to build various types of applications like: content management systems, social networks sites or scientific computing platforms etc.

Open Source:

Django is an open source web application framework. It is publicly available without cost. It can be downloaded with source code from the public repository. Open source reduces the total cost of the application development.

Vast and Supported Community:

Django is an one of the most popular web framework. It has widely supportive community and channels to share and connect.

Excellent Documentation:

If we compare Django with other open source technologies, it offers the best documentation in the market. Better documentation of any technology is like a very well-established library for any developer. There, he can search for any function desired with ease with the time involving in the searching purpose only.

FEATURES OF MongoDB**Schema-less Database:**

A Schema-less database means one collection can hold different types of documents in it., a single collection can hold multiple documents and these documents may consist of the different numbers of fields, content, and size.

Document Oriented:

In MongoDB, all the data stored in the documents instead of tables like in RDBMS. In these documents, the data is stored in fields(key-value pair)

Indexing:

In MongoDB database, every field in the documents is indexed with primary and secondary indices this makes easier and takes less time to get or search data.

Scalability:

MongoDB provides horizontal scalability with the help of sharding. Sharding means to distribute data on multiple servers, here a large amount of data is partitioned into data chunks using the shard key, and these data chunks are evenly distributed across shards that reside across many physical servers.

Replication:

It creates multiple copies of the data and sends these copies to a different server so that if one server fails, then the data is retrieved from another server.

Aggregation:

It allows to perform operations on the grouped data and get a single result

High Performance:

The performance of MongoDB is very high and data persistence as compared to another database due to its features like scalability, indexing, replication, etc.

SOLUTION

MODULE DETAILS

Module I: Administrator

Administrator is the one who handles the system, he who can analyse the system performance, data generated by the system etc.

Functions:

- Login
- View Users
- Delete Users
- View Feedbacks
- View Images
- Delete Feedbacks
- Delete Images
- Filter Feedbacks
- Change Password
- Logout

Module II: User

Users are those who need to draw DFD as part of their software development project.

Functions:

- Provide Project Details
- Generate DFD
- Download Image
- Give Feedback

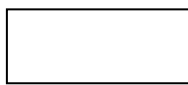
DATA FLOW DIAGRAM (DFD)

A data flow diagram is a graphical tool used to describe and analyse movement of data through a system. These are the central tool and the basis from which the other components are developed. The transformation of data from input to output, through processed may be described logically and independently of physical components associated with the system. These are known as the logical data flow diagrams. The physical data flow diagram shows the actual implements and movement of data between people, departments and workstations. A full description of a system actually consists of a set of data flow diagrams. Each component in a DFD is labelled with a descriptive name. Process is further identified with a number that will be used for identification purpose. The development of DFD's is done in several levels. Each process in lower level diagram can be broken down in to a more detailed DFD in the next level. The top level diagram is often called Context Diagram. It consists a single process bit, which plays vital role in studying the current system. The process in the context level diagram is exploded in to other process at the first level DFD.

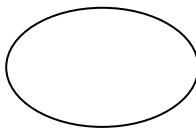
A DFD is also known as 'Bubble Chart' has the purpose of clarifying system requirements and identifying major transformations that will become programs in system design. So it is the starting point of the design to the lowest level of detail. A DFD consist of a series of bubbles joined by data flows in the system. Data flow diagrams are one of the three essential perspectives of the structured-systems analysis and design method SSADM. The sponsor of a project and the end users will need to be briefed and consulted throughout all stages of a system's evolution. With a data flow diagram, users are able to visualize how the system will operate, what the system will accomplish, and how the system will be implemented. The old system's data flow diagrams can be drawn up and compared with the new system's data flow diagrams to draw comparisons to implement a more efficient system. Data flow diagrams can be used to provide the end user with a physical idea of where the data they input ultimately has an effect upon the structure of the whole system from order to dispatch to report. How any system is developed can be determined through a data flow diagram model.

In DFD, there are four main symbols

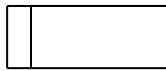
1. A square defines a source(originator) or destination of system data
2. An arrow identifies data flow. It is the pipeline through which the information flows
3. A circle or a bubble represents a process that transforms incoming data flow into outgoing data flows.
4. An open rectangle is a data store, data at rest or a temporary repository of data



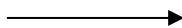
Represents source/destination data



Represents process

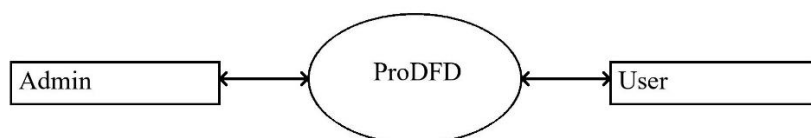


Represents data store

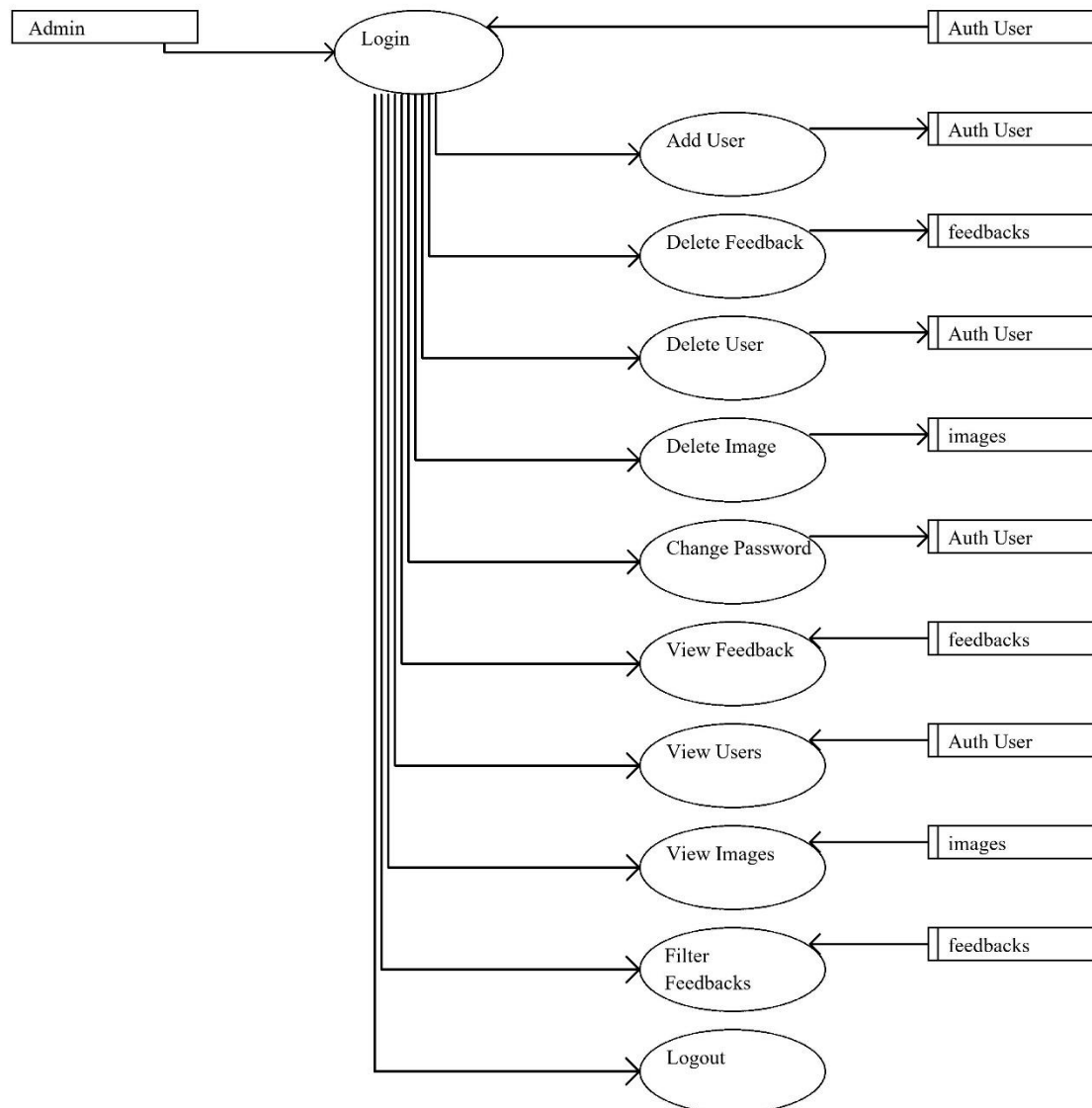


To represent data flow

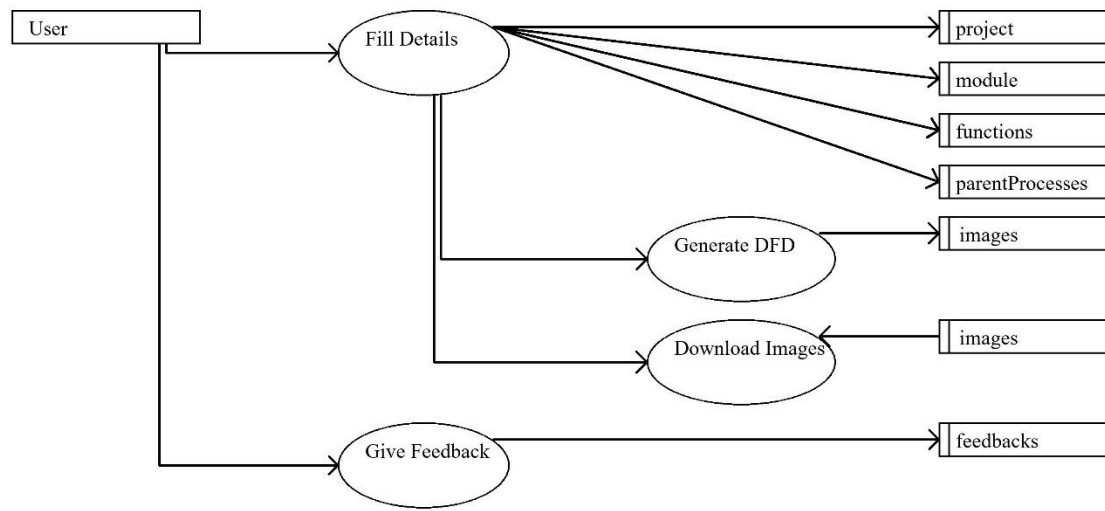
LEVEL ZERO DFD



LEVEL ONE DFD: ADMIN



LEVEL ONE DFD: USER



DATABASE DESIGN

MongoDB stores data records as documents which are gathered together in collections. A database stores one or more collections of documents. Collections are analogous to tables in RDBMS.

Collections

1. project

Field	Type	Description
id	Integer Field	Id of documents
pro_name	Char Field	Name of the project

2. modules

Field	Type	Description
id	Integer Field	Id of documents
pro_id	Integer Field	Id of the project
mod_name	Char Field	Name of the module

3. parentprocess

Field	Type	Description
id	Integer Field	Id of documents
mod_id	Integer Field	Id of the modules
p_process	Char Field	Name of function from which some other functions get generated

4. functions

Field	Type	Description
id	Integer Field	Id of documents
mod_id	Integer Field	Id of the module
process	Char Field	Name of the function
type_of_op	Char Field	Choice between select, update, none
tb_name	Char Field	Name of the table where data interacts
parent	Char Field	The function from which the current functions get generated

5. images

Field	Type	Description
id	Integer Field	Id of documents
pro_id	Integer Field	Id of the project
image	Image Field	Name of the image with extension

6. feedbacks

Field	Type	Description
id	Integer Field	Id of documents
usr_name	Char Field	Name of the user
usr_mail	Email Field	Email id of the user
usr_msg	Char Field	Messege from the user

UI COMPONENTS

Input type text

Input text controls are used to collect User data as a free text. On the web page, it will form a rectangle box where Users can input the data.

Input type email

This control is used to collect the email id. It should be in the email format

Input type submit

When input type is of submitting it performs the action defined in the form action and sends the form data to the server.

Select control

It is used to create drop-down list that enables the user to select one option out of multiple possible options.

TextArea

This input control allows the user to enter data of multiple lines. Typical usage of such input controls is for message, comments, addresses, description and so on.

Button

Represents a clickable button, used to submit forms or anywhere in a document for accessible, standard button functionality.

METHODOLOGY

The methodology used here is Iterative and Incremental Process Model. Iterative process starts with a simple implementation of a subset of the software requirements and iteratively enhances the evolving versions until the full system is implemented. At each iteration, design modifications are made and new functional capabilities are added. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental).

Following is the pictorial representation of Iterative and Incremental model:

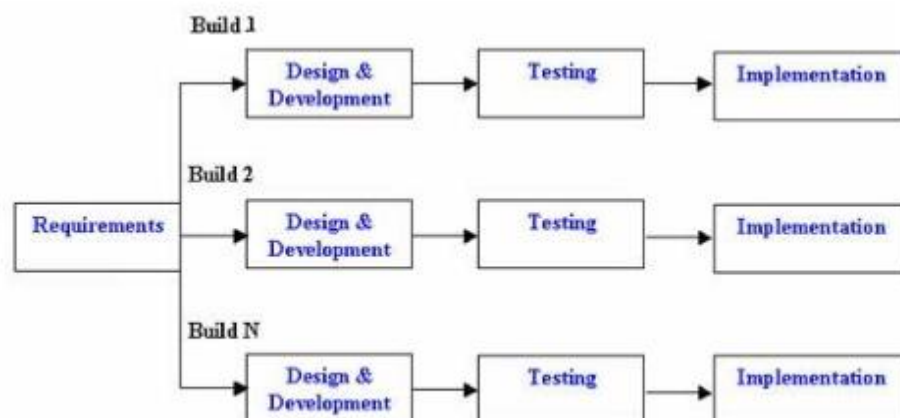


Fig: Iterative and Incremental Process Model

Iterative and Incremental development is a combination of both iterative design or iterative method and incremental build model for development. "During software development, more than one iteration of the software development cycle may be in progress at the same time." This process may be described as an "evolutionary acquisition" or "incremental build" approach."

In this incremental model, the whole requirement is divided into various builds. During each iteration, the development module goes through the requirements, design, implementation and testing phases. Each subsequent release of the module adds function to the previous release. The process continues till the complete system is ready as per the requirement.

The key to a successful use of an iterative software development lifecycle is rigorous validation of requirements, and verification & testing of each version of the software against those requirements within each cycle of the model. As the software evolves through successive cycles, tests must be repeated and extended to verify each version of the software.

Iterative Model - Pros and Cons

The advantage of this model is that there is a working model of the system at a very early stage of development, which makes it easier to find functional or design flaws.

Finding issues at an early stage of development enables to take corrective measures in a limited budget.

The disadvantage with this SDLC model is that it is applicable only to large and bulky software development projects. This is because it is hard to break a small software system into further small serviceable increments/modules.

The advantages of the Iterative and Incremental SDLC Model are as follows:

- Results are obtained early and periodically.
- Parallel development can be planned.
- Less costly to change the scope/requirements.
- With every increment, operational product is delivered.
- Issues, challenges and risks identified from each increment can be utilized/applied to the next increment.
- Initial Operating time is less.
- Better suited for large and mission-critical projects.

- During the life cycle, software is produced early which facilitates customer evaluation and feedback.

The disadvantages of the Iterative and Incremental SDLC Model are as follows –

- More resources may be required.
- Although cost of change is lesser, but it is not very suitable for changing requirements.
- More management attention is required.
- System architecture or design issues may arise because not all requirements are gathered in the beginning of the entire life cycle.
- Defining increments may require definition of the complete system.
- Not suitable for smaller projects.

LIMITATIONS

“ProgrammedDFD” is well suitable for projects with moderate number of functions and table interactions for each module. As it follows a vertically aligned structure it is hard to cope with increasing length of the diagram, since it should be contained in a single page. Due to the same reason, Levels of the diagram is limited to level one, and the remaining levels are congested into child diagrams of the level one.

CODE

process.html

Code used to add functions that need database updation

```
ar c=1;

function ins_add_trigger(i,count){
    if(temp1=="||temp1!=i) {
        inc=65;
        c=1;
    }

    let node = document.createElement('tr');

    node.innerHTML='<tr><td>Function</td><td style="width:600px;"><input
type="text" id="insert_fun'+i+c+'" name="insert_fun'+i+'" pattern="[^@]{1,30}" title="Only
30 characters are permitted"></td><td>Table</td><td><input type="text"
name="insert_tb'+i+'" pattern="[^@]{1,15}" title="Only 15 characters are
permitted"></td><td>Parent</td><td id="click1'+i+c+'"><select id="parent_list1'+i+c+'"
name="ins_select'+i+'" ></select></td></tr>';

    Top=document.getElementById('add_button1'+i).offsetTop;
    document.getElementById('insert_body'+i).appendChild(node);
    document.getElementById('add_button1'+i).style.top =Top+inc + 'px';
    let node2=document.getElementById("parent_list1"+i+c);
    node2.innerHTML='<option value="">Select</option>';
    document.getElementById("parent_list1"+i+c).onclick=function(){
        var j=0;
        while(j<parent_count+1){
            var cval=c-1;
            var parent=document.getElementById("parent_fun"+i+j).value;
            if(parent!=""){
                $(new Option(parent,parent,true)).appendTo("#parent_list1"+i+cval);
            }
            j++;
        }
        $(new Option("None","None",true)).appendTo("#parent_list1"+i+cval);
    };
    inc--;
```

```

temp1=i;
c++;
}

```

```

<!--Database Insert Functions-->
<table class="alt" style="width:1200px;margin-left:30px;">
  <thead>
    <div style="font size: larger; position: relative;left:30px;height:60px;
      width:1050px"><b>Functions Requiring Database Update</b></div>
  </thead>
  <tbody id="insert_body{ {i} }">
    <tr style="position:relative;">
      <td>Function</td>
      <td style="width:500px;"><input type="text" id="insert_fun{ {i} }0"
name="insert_fun{ {i} }" required pattern="[^\@]{1,30}"></td>
      <td>Table</td>
      <td><input type="text" name="insert_tb{ {i} }" required
pattern="[^\@]{1,15}"></td>
      <td>Parent</td><td id="click1{ {i} }0" ><select name="ins_select{ {i} }"
id="parent_list1{ {i} }0" onclick='first_ins_select("{ {i} }",1)' required><option
value="">Select</option></select></td>
      <button id="add_button1{ {i} }" type="button"
style="position:absolute;left:1300px;width:fit-content;margin-top :10px"
onclick='ins_add_trigger("{ {i} }","{ {forloop.counter} }"); return false'>ADD</button>
    </tr>
  </tbody>
</table>

```

view.py

Code to insert functions that need database updation

```

def func_insert(request):
    project_id=request.session.get('project_ide')
    modules=Modules.objects.filter(pro_id=project_id)

```

```

mod_count=Modules.objects.filter(pro_id=project_id).count()

for i in range(int(mod_count)):
    par_fun='parent_fun'+modules[i].mod_name
    par_fun_list=request.POST.getlist(par_fun)
    for a in par_fun_list:
        function=Parentprocess(mod_id=modules[i].id,p_process=a)
        function.save()

    ins_fun='insert_fun'+modules[i].mod_name
    ins_tb='insert_tb'+modules[i].mod_name
    ins_fun_list=request.POST.getlist(ins_fun)
    ins_tb_list=request.POST.getlist(ins_tb)
    ins_select='ins_select'+modules[i].mod_name
    ins_select_list=request.POST.getlist(ins_select)
    for a,b,c in zip(ins_fun_list,ins_tb_list,ins_select_list):
        if(a!="" and b!="" and c!=""):

function=Function(mod_id=modules[i].id,process=a,type_of_op='update',tb_name=b,parent=
c)

    function.save()

def generateDfd(request):
    project_name=request.session.get('project_name')
    project=Project(pro_name=project_name)
    pro_name=project.pro_name
    modules=Modules.objects.filter(pro_id=int(request.session.get('project_id')))
    mod_count=Modules.objects.filter(pro_id=int(request.session.get('project_id'))).count()

    modul=Modules.objects.filter(pro_id=int(request.session.get('project_id'))).values_list('pk',
flat=True)

```



```

module_list=[]
ins_fun_list=[]
ins_table_list=[]
sel_fun_list=[]
sel_table_list=[]
saf_list=[]
par_list=[]
sample()
mod_ids=request.session.get('module_ids')
for i in range(int(mod_count)):
    i_f_list=[]
    i_t_list=[]
    s_f_list=[]
    s_t_list=[]
    s_list=[]
    p_list=[]
    module_list.append(modules[i].mod_name)
    functions=Function.objects.filter(mod_id=modul[i])
    fun_count=Function.objects.filter(mod_id=modul[i]).count()
    parents=Parentprocess.objects.filter(mod_id=modul[i])
    par_count=Parentprocess.objects.filter(mod_id=modul[i]).count()
    for k in range(int(par_count)):
        p_list.append(parents[k].p_process)
    for j in range(int(fun_count)):
        op=functions[j].type_of_op
        if(op=="update"):
            i_f_list.append(functions[j].process)
            i_t_list.append(functions[j].tb_name)

```

```

elif(op=="select"):
    s_f_list.append(functions[j].process)
    s_t_list.append(functions[j].tb_name)
else:
    s_list.append(functions[j].process)
ui_f_list=[]
us_f_list=[]
us_list=[]
for j in i_f_list:
    if j not in ui_f_list:
        ui_f_list.append(j)
for j in s_f_list:
    if j not in us_f_list:
        us_f_list.append(j)
for j in s_list:
    if j not in us_list:
        us_list.append(j)
ins_fun_list.append(ui_f_list)
ins_table_list.append(i_t_list)
sel_fun_list.append(us_f_list)
sel_table_list.append(s_t_list)
saf_list.append(us_list)
par_list.append(p_list)

#LEVEL ZERO
img = Image.new('RGB',(3451,2423),"white")
draw = ImageDraw.Draw(img)

```

```

proname=""
if(len(pro_name)>15):
    proname=split_text(pro_name)

shape=[1480,1100,1970, 1400]
draw.ellipse(shape, fill ="#ffff", outline ="black",width=5)
font = ImageFont.truetype("times.ttf", 60, encoding="unic")
n=1210
q=1310
n2=1210
q2=1310
flag1=""
flag2=""

for i in range(int(mod_count)):
    if((i+1)%2==1 and flag1==0):
        p1=(800,n+200)
        p2=(1280,q+200)
        flag1=1
    elif((i+1)%2==1 and flag1==1):
        p1=(800,n-200)
        p2=(1280,q-200)
        flag1=0
    elif ((i+1)%2==0 and flag2==0):
        p1=(2170,n2+200)
        p2=(2650,q2+200)
        flag2=1
    elif((i+1)%2==0 and flag2==1):

```

```

        p1=(2170,n2-200)
        p2=(2650,q2-200)
        flag2=0
    elif((i+1)%2==1 and flag1==""):
        p1=(800,n)
        p2=(1280,q)
        flag1=0
    else:
        p1=(2170,n2)
        p2=(2650,q2)
        flag2=0

    draw.rectangle([p1,p2], fill = "#ffff", outline = "black",width=5)
    draw.text((p1[0]+20,p1[1]+10),modules[i].mod_name,'black',font)
    if(proname!=""):
        draw.text((1540,1180),proname[0], 'black', font)
        draw.text((1540,1260),proname[1], 'black', font)
    else:
        draw.text((1630,1200),pro_name, 'black', font)

    now = datetime.now()
    current_time = now.strftime("%H_%M_%S")

    imgname='level_zero_'+current_time+'.jpg'
    img.save('media/'+imgname)
    path = r'media/'+imgname
    imag = cv2.imread(path,None)

```

```
n=1250
p=1480
q=1250
m2=1970
n2=1250
q2=1250
flag1=""
flag2=""
```

```
for i in range(int(mod_count)):
    if((i+1)%2==1 and flag1==0):
        p1=(1280,n+200)
        p2=(p+1,q+1)
        flag1=1
    elif((i+1)%2==1 and flag1==1):
        p1=(1280,n-200)
        p2=(p-1,q-1)

        flag1=0
    elif ((i+1)%2==0 and flag2==0):
        p1=(m2+1,n2+1)
        p2=(2170,q2+200)
        flag2=1
    elif((i+1)%2==0 and flag2==1):
        p1=(m2-1,n2-1)
        p2=(2170,q2-200)
        flag2=0
    elif((i+1)%2==1 and flag1==""):
```

```

        p1=(1280,n)
        p2=(p,q)
        flag1=0
    else:
        p1=(m2,n2)
        p2=(2170,q2)
        flag2=0
    k1=cv2.arrowsLine(imag, p1, p2, (0,0,0), (5), 8,0,0.1)
    k2=cv2.arrowsLine(imag, p2, p1, (0,0,0), (5), 8,0,0.1)
    cv2.imwrite("media/"+imgname,k1)
    cv2.imwrite("media/"+imgname,k2)
project_id=request.session.get('project_id')
imgname='media/'+imgname
images=Images(pro_id=project_id,image=imgname )
images.save()

```

#LEVEL ONE

```

for i in range(int(mod_count)):
    img1 = Image.new('RGB',(3451,5423),"white")
    draw1 = ImageDraw.Draw(img1)
    mod_name=module_list[i]
    # Drawing module
    draw1.rectangle([(150,200),(700,300)], fill = "#ffff", outline = "black",width=5)
    draw1.text((200,220),mod_name,"black",font)
    n=1100
    p=200
    q=1600

```

```

r=450
rx1=2850
ry1=200
rx2=3350
ry2=300
sn=2000
sp=500
sq=2550
sr=750
sub_parent=[]
none_fun=Function.objects.filter(mod_id=modul[i],parent='None')
none_c=Function.objects.filter(mod_id=modul[i],parent='None').count()
none_list=[]
for d in range(int(none_c)):
    if(none_fun[d].process not in none_list):
        none_list.append(none_fun[d].process)
        if((none_fun[d].process).casefold()=='register'):
            none_list.append('Login')
for j in range(int(len(none_list))):
    if(p!=5150):
        parent_name=none_list[j]
        if(len(parent_name)>15):
            parnt_name=split_text(parent_name)
        #Drawing Parent
        n=1100
        q=1600
        draw1.ellipse([(n,p),(q,r)],fill="#fff",outline="black",width=5)
        dx=n

```

```

dy=p
if(len(parent_name)>15):
    draw1.text((dx+75,dy+50),parnt_name[0], "black",font)
    draw1.text((dx+75,dy+130),parnt_name[1], "black",font)
else:
    draw1.text((dx+80,dy+50),parent_name, "black",font)
sp=r+50
sr=r+300
p=r+50
r=r+300
#fetching tables of parent function
ct=0
for k in ins_fun_list[i]:
    if(parent_name.casefold()==k.casefold()):

function=Function.objects.filter(mod_id=modul[i],process=k,type_of_op="update")

count=Function.objects.filter(mod_id=modul[i],process=k,type_of_op="update").count()
    for l in range(int(count)):
        table_name=function[l].tb_name
        draw1.rectangle([rx1,ry1,rx2,ry2],fill="#fff",outline="black",width=5)
        trx1=rx1
        dy=ry1
        draw1.line([(trx1+30,ry1),(trx1+30,ry2)],fill="black",width=5,join=None)
        draw1.line([(rx2,ry1),(rx2-3,ry2)],fill="white",width=5,join=None)
        draw1.text((trx1+50,dy+20),table_name,"black",font)
        ry1=ry2+50
        ry2=ry2+150
        ct=ct+1

```



```

for f in sel_fun_list[i]:
    if(parent_name.casefold()==f.casefold()):

funcio=Function.objects.filter(mod_id=modul[i],process=f,type_of_op="select")

sel_count=Function.objects.filter(mod_id=modul[i],process=f,type_of_op="select").count()

    for w in range(int(sel_count)):
        table_name=funcio[w].tb_name
        draw1.rectangle([rx1,ry1,rx2,ry2],fill="#fff",outline="black",width=5)
        trx1=rx1
        dy=ry1
        draw1.line([(trx1+30,ry1),(trx1+30,ry2)],fill="black",width=5,join=None)
        draw1.line([(rx2,ry1),(rx2-3,ry2)],fill="white",width=5,join=None)
        draw1.text((trx1+60,dy+20),table_name,"black",font)
        ry1=ry2+50
        ry2=ry2+150
        ct=ct+1

if(ct>1):
    p=ry1
    r=ry1+250
    sp=ry1
    sr=ry1+250
else:
    ry1=sp
    ry2=sp+100

#Fetching Sub processes

func=Function.objects.filter(mod_id=modul[i],parent=parent_name)

f_count=Function.objects.filter(mod_id=modul[i],parent=parent_name).count()

child_fun_list=[]

```

```

flag=0
for m in range(int(f_count)):
    if func[m].process not in child_fun_list:
        if((func[m].process).casefold()=='login'):
            continue
        child_fun_list.append(func[m].process)
for m in range(int(len(child_fun_list))):
    function_name=child_fun_list[m]
    if(len(function_name)>15):
        functn_name=split_text(function_name)
        #Drawing sub process
        draw1.ellipse([(sn,sp),(sq,sr)],fill="#fff",outline="black",width=5)
        if(len(function_name)>15):
            draw1.text((sn+75,sp+50),functn_name[0],"black",font)
            draw1.text((sn+75,sp+130),functn_name[1],"black",font)
        else:
            draw1.text((sn+80,sp+50),function_name,"black",font)
        sp=sr+50
        sr=sr+300
        p=sp
        r=sr
        ct=0

fn1=Function.objects.filter(mod_id=modul[i],process=function_name,type_of_op='update')

f_c_1=Function.objects.filter(mod_id=modul[i],process=function_name,type_of_op='update'
).count()

for h in range(int(f_c_1)):
    table_name=fn1[h].tb_name

```

```

draw1.rectangle([rx1,ry1,rx2,ry2],fill="#fff",outline="black",width=5)
lx1=rx1
ly=ry1
draw1.line([(lx1+30,ry1),(lx1+30,ry2)],fill="black",width=5,join=None)
draw1.line([(rx2,ry1),(rx2-3,ry2)],fill="white",width=5,join=None)
draw1.text((lx1+60,ly+20),table_name,"black",font)
ry1=ry2+50
ry2=ry2+150
ct=ct+1

```

```

fn2=Function.objects.filter(mod_id=modul[i],process=function_name,type_of_op='select')

```

```

f_c_2=Function.objects.filter(mod_id=modul[i],process=function_name,type_of_op='select').
count()

```

```

for r in range(int(f_c_2)):
    table_name=fn2[r].tb_name
    draw1.rectangle([rx1,ry1,rx2,ry2],fill="#fff",outline="black",width=5)
    lx1=rx1
    ly=ry1
    draw1.line([(lx1+30,ry1),(lx1+30,ry2)],fill="black",width=5,join=None)
    draw1.line([(rx2,ry1),(rx2-3,ry2)],fill="white",width=5,join=None)
    draw1.text((lx1+60,ly+20),table_name,"black",font)
    ry1=ry2+50
    ry2=ry2+150
    ct=ct+1
if(ct>1):
    sp=ry1
    sr=ry1+250
    p=ry1

```

```

        r=ry1+250
    else:
        p=sp
        r=sr
        ry1=sp
        ry2=sp+100

    if(Function.objects.filter(mod_id=modul[i],parent=function_name)):
        sub_parent.append(function_name)

    imgname='level_one_'+str(project_id)+mod_name+'.jpg'
    img1.save('media/'+imgname)

#Draw Lines in Level One
path = r'media/'+imgname
im = cv2.imread(path,None)

l1,l2,l3,l4=600,300,600,325
a1,a2,a3,a4=600,325,1100,325
s11,s12,s13,s14=1400,450,1400,625
sa1,sa2,sa3,sa4=1400,625,2000,625
ra1,ra2,ra3,ra4=1553,250,2850,250
sra1,sra2,sra3,sra4=2503,550,2850,550

for j in range(int(len(none_list))):
    line1=cv2.line(im, (l1,l2), (l3,l4), (0,0,0), (5))
    aline1=cv2.arrowedLine(im, (a1,a2),(a3,a4), (0,0,0), (5), 8,0,0.07)
    cv2.imwrite("media/"+imgname,line1)
    cv2.imwrite("media/"+imgname,aline1)

l1=l1-20
l3=l3-20

```

```

a1=a1-20

l4=l4+300

a2=a2+300

a4=a4+300

parent_name=none_list[j]

ct,ct0,ct1=0,0,0

for k in ins_fun_list[i]:

    if(parent_name.casefold()==k.casefold()):

function=Function.objects.filter(mod_id=modul[i],process=k,type_of_op="update")

count=Function.objects.filter(mod_id=modul[i],process=k,type_of_op="update").count()

    for l in range(int(count)):

        aline3=cv2.arrowedLine(im, (ra1,ra2),(ra3,ra4), (0,0,0), (5), 5,0,0.03)

        cv2.imwrite("media/"+imgname,aline3)

        ra4=ra4+150

        ct0=ct0+1

    for f in sel_fun_list[i]:

        if(parent_name.casefold()==f.casefold()):

function=Function.objects.filter(mod_id=modul[i],process=f,type_of_op="select")

sel_count=Function.objects.filter(mod_id=modul[i],process=f,type_of_op="select").count()

        for w in range(int(sel_count)):

            if(ct>=1):

                aline3=cv2.arrowedLine(im,(ra3,ra4),(ra1+30,ra2+75), (0,0,0), (5), 8,0,0.03)

            else:

                aline3=cv2.arrowedLine(im,(ra3,ra4),(ra1,ra2), (0,0,0), (5), 8,0,0.03)

            cv2.imwrite("media/"+imgname,aline3)

            ra4=ra4+150

```

```

        ct1=ct1+1
ct=ct0+ct1
if(ct>1):
    l4=ra4+150
    a2=ra4+150
    a4=ra4+150
    sl4=ra4+75
    sa2=ra4+75
    sa4=ra4+75
    ra2=ra4
    sra2=ra4
    sra4=ra4
else:
    ra2=sra2-75
    sra2=sra2-75
    ra4=ra2
    sra4=sra2
if(none_list[0].casefold()=='register'):
    alinelog=cv2.arrowedLine(im, (1350,450),(1350,498), (0,0,0), (5), 8,0,0.5)
    cv2.imwrite("media/"+imgname,alinelog)
func=Function.objects.filter(mod_id=modul[i],parent=parent_name)
f_count=Function.objects.filter(mod_id=modul[i],parent=parent_name).count()
child_fun_list2=[]
for m in range(int(f_count)):
    if func[m].process not in child_fun_list2:
        if((func[m].process).casefold()=='login'):
            continue
        child_fun_list2.append(func[m].process)

```

```

for m in range(int(len(child_fun_list2))):
    function_name=child_fun_list2[m]
    line2=cv2.line(im, (sl1,sl2),(sl3,sl4), (0,0,0), (5))
    aline2=cv2.arrowedLine(im, (sa1,sa2),(sa3,sa4), (0,0,0), (5), 8,0,0.07)
    cv2.imwrite("media/"+imgname,line2)
    cv2.imwrite("media/"+imgname,aline2)
    l4=sl4+300
    a2=sl4+300
    a4=sl4+300
    sl1=sl1-20
    sl3=sl3-20
    sa1=sa1-20
    sl4=sl4+300
    sa2=sa2+300
    sa4=sa4+300
    ct,ct0,ct1=0,0,0

fn1=Function.objects.filter(mod_id=modul[i],process=function_name,type_of_op='update')

f_c_1=Function.objects.filter(mod_id=modul[i],process=function_name,type_of_op='update'
).count()

for h in range(int(f_c_1)):
    aline3=cv2.arrowedLine(im,(sra1,sra2),(sra3,sra4), (0,0,0), (5), 8,0,0.12)
    cv2.imwrite("media/"+imgname,aline3)
    sra4=sra4+150
    ct0=ct0+1

fn2=Function.objects.filter(mod_id=modul[i],process=function_name,type_of_op='select')

```

```

f_c_2=Function.objects.filter(mod_id=modul[i],process=function_name,type_of_op='select').
count()

    for r in range(int(f_c_2)):
        if(ct>=1):
            aline3=cv2.arrowedLine(im,(sra3,sra4),(sra1+30,sra2+75),    (0,0,0),    (5),
8,0,0.12)
        else:
            aline3=cv2.arrowedLine(im,(sra3,sra4),(sra1,sra2), (0,0,0), (5), 8,0,0.12)
        cv2.imwrite("media/"+imgname,aline3)
        sra4=sra4+150

        ct1=ct1+1
    ct=ct0+ct1
    if(ct>1):
        l4=sra4+75
        a2=sra4+75
        a4=sra4+75
        sl4=sra4+75
        sa2=sra4+75
        sa4=sra4+75
        ra2=sra4
        ra4=ra2
        sra2=sra4
    else:
        ra2=sa2-75
        sra2=sa2-75
        ra4=ra2
        sra4=sra2

```



```
s11=1400
s12=a2+125
s13=1400
s14=s12+175
sa1=1400
sa2=s14
sa4=s14

images=Images(pro_id=project_id,image=imgname)
images.save()

imgs=Images.objects.filter(pro_id=project_id)
img_count=Images.objects.filter(pro_id=project_id).count()
return render(request,'download.html',{'count':img_count,'i':imgs,"down":"Download "})
```

FUNCTIONS USED

JavaScript

- ***createElement()***: Used to create a new HTML element and attach it to the DOM tree.

Syntax: `document.createElement(html tag)`

Usage: `let node = document.createElement('tr');`

It creates a new html element tr and later append it to the DOM.

It is used to create new elements for getting function details.

- ***appendChild()***: It appends a node as the last child of a node.

Syntax: `node.appendChild(new_node)`

Usage: `document.getElementById('stand_alone_body'+i).appendChild(node);`

Here the newly created node is appended to the element with id 'stand_alone_body +i'.

It is used to add elements to the existing elements for collecting stand alone functions.

- ***appendTo()***: It returns the element that has the ID attribute with the specified value.

Syntax: `document.getElementById(elementID)`

Usage: `$(new Option("None","None",true)).appendTo("#parent_list3"+i+cval);`

Here newly created option element is appended to the select tag with id parent_list3+i+cval.

Here when the button is clicked, new elements for getting function details are added

JQuery

- ***ready()***: will only run once the page Document Object Model (DOM) is ready for JavaScript code to execute.

Syntax: `$(document).ready(function() { statements });`

Usage: `$(document).ready(function(){`

`var i=2;var top;`

`$("#insert_mod_btn").click(function(e){`

`$("#insert_mod_btn").animate({ "top": "+=73px" }); });`

Here on button click, it animates the button with id insert_mod_btn when the DOM is ready.

- ***animate()***: It performs a custom animation of a set of CSS properties.

Syntax: *(selector).animate({styles},speed,easing,callback)*

styles: Required. Specifies one or more CSS properties/values to animate.

Eg: width, top, margin etc.

speed: Optional. Specifies the speed of the animation. Default value is 400 milliseconds

Possible values:

- milliseconds (like 100, 1000, 5000, etc)
- "slow"
- "fast"

easing: Optional. Specifies the speed of the element in different points of the animation
Default value is "swing". Possible values:

- "swing" - moves slower at the beginning/end, but faster in the middle
- "linear" - moves in a constant speed

callback: Optional. A function to be executed after the animation completes.

Usage: `$("#insert_mod_btn").animate({ "top": "+=73px" });`

It animates the button with id insert_mod_btn to 73px downward. This button is used to add new modules.

- ***click()***: Triggers the click event, or attaches a function to run when a click event occurs.

Syntax: `$(selector).click()`

Usage: `$("#insert_mod_btn").click(function(e){ }`

Here it performs some actions on button with id insert_mod_btn is clicked.

It is the button used to add new functions.

Django

- ***request.session.get()***: Used to get the values of session variables

Syntax: `request.session.get(key)`

Usage: `project_id=request.session.get('project_id')`

It returns the project id and store it in variable project_id

Project details are managed through this session variable.

- ***request.method.getList()*** : method can be post or get.

Syntax: `request.POST.getlist('name',default=None)`:-Gets the list of values of the name parameter in a POST request or gets None if the parameter is not present

Usage: `ins_fun_list=request.POST.getlist(ins_fun)`

It returns the values of the element with name `ins_fun` and store it to `ins_fun_list`. Used to get all the details regarding the database insertion processes.

- ***messages.info()***: Used to send messages to the html page

Syntax: `messages.info(request,message)`

Usage: `messages.info(request, 'Your feedback has been submitted successfully!')`

It is used to send status of the feedback send.

- ***Model.objects.filter()***:- returns a multiple database objects that matches the values given as parameter

Syntax: `Model.object.filter(tb_attribute='value',...)`

Usage: `function=Function.objects.filter(mod_id=modul[i])`

Used to filter out the contents of the collection Function based on the `modul_id` stored in `modul[i]`, and store the objects in variable functions.

- ***Model.save()***: Used to save data

Syntax:

`Model.save(force_insert=False, force_update=False, using=DEFAULT_DB_ALIAS, update_fields=None)`

1st 2 parameters are not usually used, because Django automatically detects the operation.

Using: specifies the primary key alias

Update_fields:Used to specify which fields need to be updated, and others are not updated.

Usage: `function.save()`

It saves the updated content to the collection Function

Python

- ***zip()***: Takes iterables (can be zero or more), aggregates them in a tuple, and return it.

Syntax: `zip(*iterables)`

Iterables can be list,string,dict etc.

Usage: for a,b,c in zip(ins_fun_list,ins_tb_list,ins_select_list):

```
if(a!="" and b!="" and c!=""):
    function=Function(mod_id=modules[i].id,process=a,type_of_op='update',tb_name=b,parent=c)
    function.save()
```

Here zip makes a iterate over the ins_fun_list, b iterate over the ins_tb_list, and c iterate over the ins_select_lit. It is used to get the field values and store it to the collection

range(): The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.

Syntax: range(start, stop, step)

Start: : Optional. An integer number specifying at which position to start. Default is 0

Stop: Required. An integer number specifying at which position to stop (not included).

Step: Optional. An integer number specifying the incrementation. Default is 1

Usage: for k in range(int(par_count)):

```
p_list.append(parents[k].p_process)
```

Here k iterates from zero to par_count and perform the operation it is used to get a list of parent processes

➤ **append():** Appends an element to the end of the list.

Syntax: list.append(elmnt)

Elmnt: Required. An element of any type (string, number, object etc.)

Usage: for k in range(int(par_count)):

```
p_list.append(parents[k].p_process)
```

Here value of parents[k].p_process is appended to the list p_list, to get the parent process list

➤ **casefold():** It is used for caseless matching, i.e. ignores cases when comparing.

Usage: if((none_fun[d].process).casefold()=='register'):

```
none_list.append('Login')
```

Here the process name is checked against the name 'register' ignoring the case

PIL

- **PIL.Image.new():** creates a new image with the given mode and size.

Syntax: `PIL.Image.new(mode, size, color)`

mode: The mode to use for the new image. (It could be RGB, RGBA)

size: A 2-tuple containing (width, height) in pixels.

color: What color to use for the image. Default is black. If given, this should be a single integer or floating point value for single-band modes, and a tuple for multi-band modes.

Return Value: An Image object.

Usage: `img = Image.new('RGB',(3451,2423),"white")`

It is used for creating new image for drawing basic shapes for the DFD.

- **ImageDraw.draw():**

Syntax: `ImageDraw.Draw(image):` Creates an object that can be used to draw in the given image.

Usage: `draw = ImageDraw.Draw(img)`

The shapes and text are drawn using draw object which is created to draw in img image.

- **ImageDraw.Draw.ellipse()-** Draws an ellipse inside the given bounding box.

Syntax: `PIL.ImageDraw.Draw.ellipse(xy, fill=None, outline=None)`

xy : Four points to define the bounding box. Sequence of either [(x0, y0), (x1, y1)] or [x0, y0, x1, y1].

outline : Color to use for the outline.

fill : Color to use for the fill.

Returns: An Image object in ellipse shape.

Usage: `draw.ellipse(shape, fill = "#ffff", outline = "black",width=5)`

Here shape contains the coordinates. It is used to draw the shape of processes

- **ImageDraw.Draw.rectangle()** Draws an rectangle.

Syntax: `PIL.ImageDraw.Draw.rectangle(xy, fill=None, outline=None)`

xy : Four points to define the bounding box. Sequence of either [(x0, y0), (x1, y1)] or [x0, y0, x1, y1]. The second point is just outside the drawn rectangle.

outline : Color to use for the outline.

fill : Color to use for the fill.

Returns: An Image object in rectangle shape.

Usage: `draw.rectangle([p1,p2], fill = "#ffff", outline = "black", width=5)`

It is used to draw the table in DFD

- **ImageDraw.Draw.text()** Draws the string at the given position.

Syntax: `ImageDraw.Draw.text(xy, text, fill=None, font=None, anchor=None, spacing=0, align="left")`

xy : Top left corner of the text.

text : Text to be drawn. If it contains any newline characters, the text is passed on to `multiline_text()`

fill : Color to use for the text.

font : An `ImageFont` instance.

spacing : If the text is passed on to `multiline_text()`, the number of pixels between lines.

align : If the text is passed on to `multiline_text()`, "left", "center" or "right".

Return Type: returns an image with text.

Usage: `draw.text((p1[0]+20,p1[1]+10),modules[i].mod_name,'black',font)`

It is used to write the names of project, modules, processes, and tables.

CV2

- **cv2.imread():** It is used for caseless matching, i.e. ignores cases when comparing.

Syntax: `cv2.imread(path, flag)`

path: A string representing the path of the image to be read.

flag: It specifies the way in which image should be read

Return Value: This method returns an image that is loaded from the specified file.

Usage: `imag = cv2.imread(path, None)`

It is used here to select the image to put lines to complete the DFD

- **cv2.line():** used to draw a line on any image.

Syntax: `cv2.line(image, start_point, end_point, color, thickness)`

image: It is the image on which line is to be drawn.

start_point: It is the starting coordinates of line. The coordinates are represented as tuples of two values i.e. (X coordinate value, Y coordinate value).

end_point: It is the ending coordinates of line. The coordinates are represented as

tuples of two values i.e. (X coordinate value, Y coordinate value).

color: It is the color of line to be drawn.eg: (255, 0, 0) for blue color.

thickness: It is the thickness of the line in px.

Return Value: It returns an image.

Usage: `draw1.line([(rx2,ry1),(rx2-3,ry2)],fill="white",width=5,join=None)`

Used to connect arrowed lines within the DFD

- **cv2.arrowedLine()**: used to draw arrows pointing from the start point to the end point.

Syntax: `cv2.arrowedLine(image, start_point, end_point, color[, thickness[, line_type[, shift[, tipLength]]]])`

image: It is the image on which line is to be drawn.

start_point: It is the starting coordinates of line. The coordinates are represented as tuples of two values i.e. (X coordinate value, Y coordinate value).

end_point: It is the ending coordinates of line. The coordinates are represented as tuples of two values i.e. (X coordinate value, Y coordinate value).

color: It is the color of line to be drawn. For BGR, we pass a tuple. eg: (255, 0, 0) for blue color.

thickness: It is the thickness of the line in px.

line_type: It denotes the type of the line for drawing.

shift: It denotes number of fractional bits in the point coordinates.

tipLength: It denotes the length of the arrow tip in relation to the arrow length.

Return Value: It returns an image.

Usage: `aline3=cv2.arrowedLine(im2,(ra3,ra4),(ra1,ra2), (0,0,0), (5), 8,0,0.03)`

Arrowed lines are drawn using this function.

- **cv2.imwrite()**:used to save an image to any storage device.

Syntax: `cv2.imwrite(filename, image)`

filename: A string representing the file name. The filename must include image format like .jpg, .png, etc.

image: It is the image that is to be saved.

Return Value: It returns true if image is saved successfully.

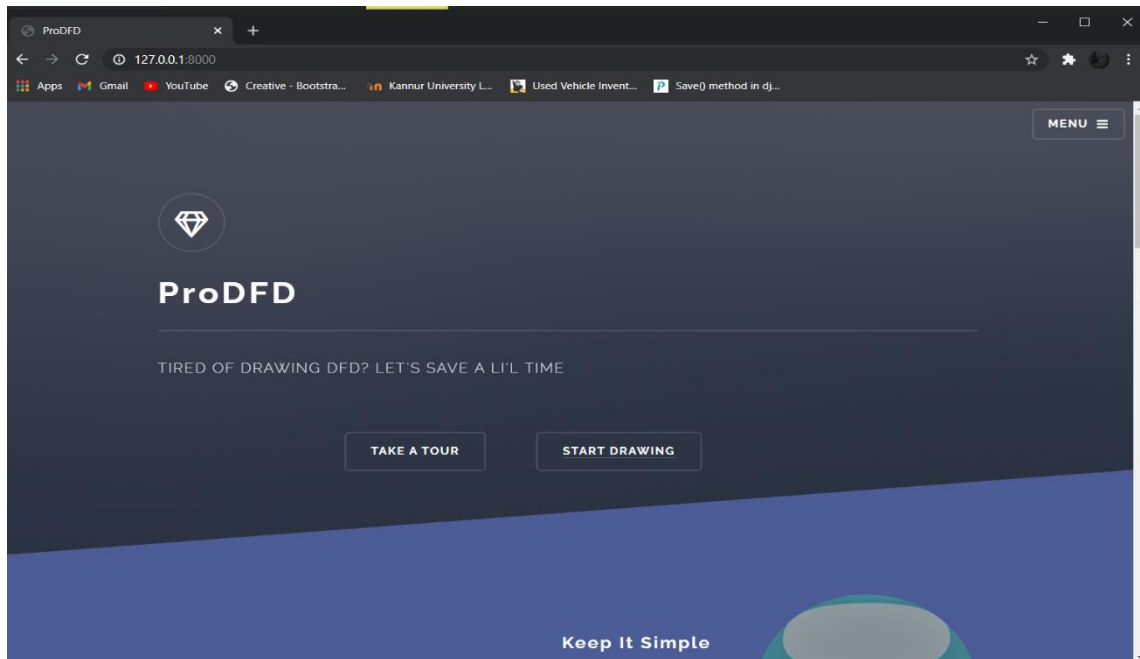
Usage: `cv2.imwrite("media/"+imgname2,aline3)`

The drawn image saved using this function.

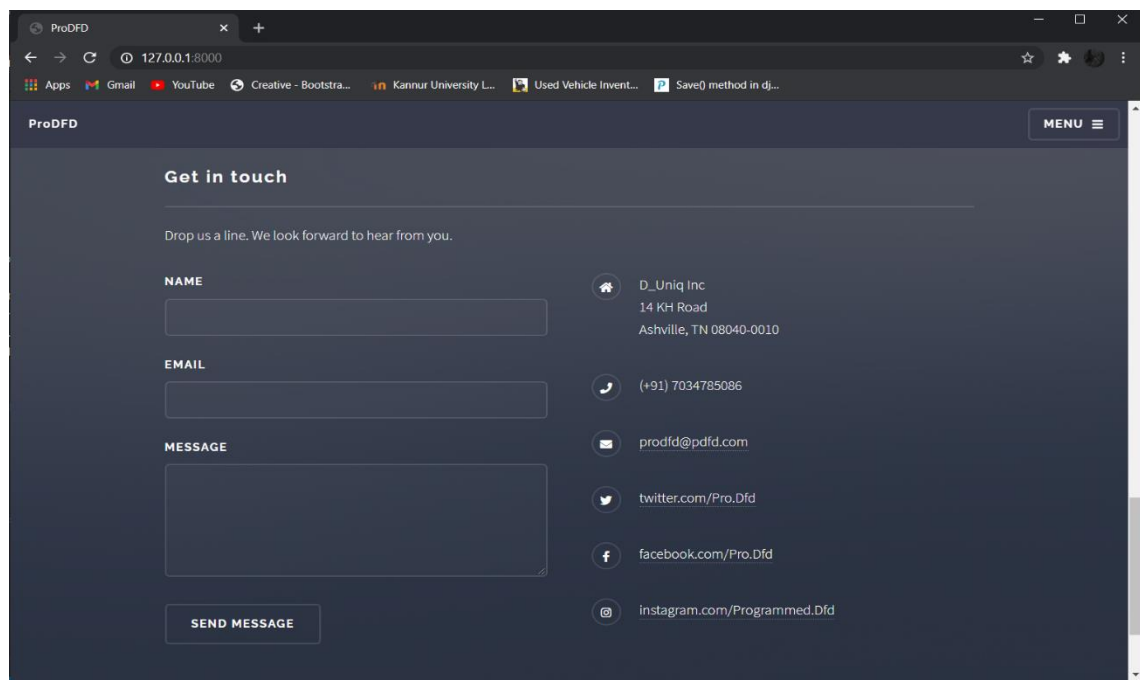
OUTPUT

(SCREENSHOTS)

ProDFD: Home page



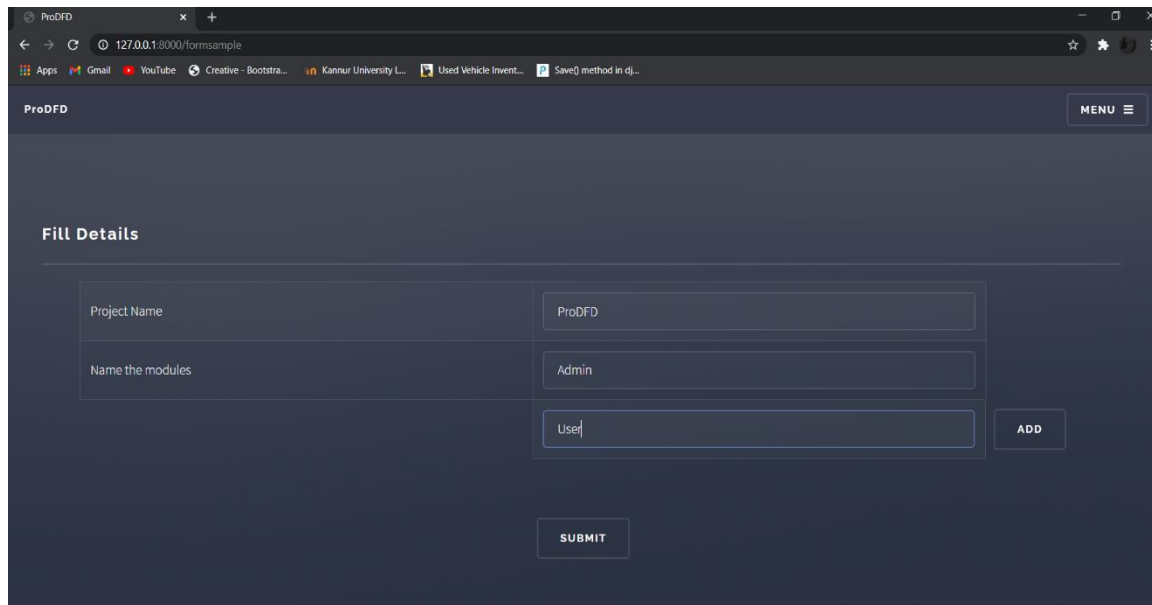
Home page contains **START DRAWING** button for a quick draw, or the user can select from the menu as well. The center part contains some brief instructions for drawing which can be pulled up by using **TAKE A TOUR** button and from the bottom the user can also provide the feedbacks



Forms

The forms are placed in 2 pages, in which 1st page collects the project name and modules, and the 2nd page collects detailed information such as functions, tables, data flow etc The ADD buttons given can be used to add new rows to form..

Form1



ProDFD

MENU

Fill Details

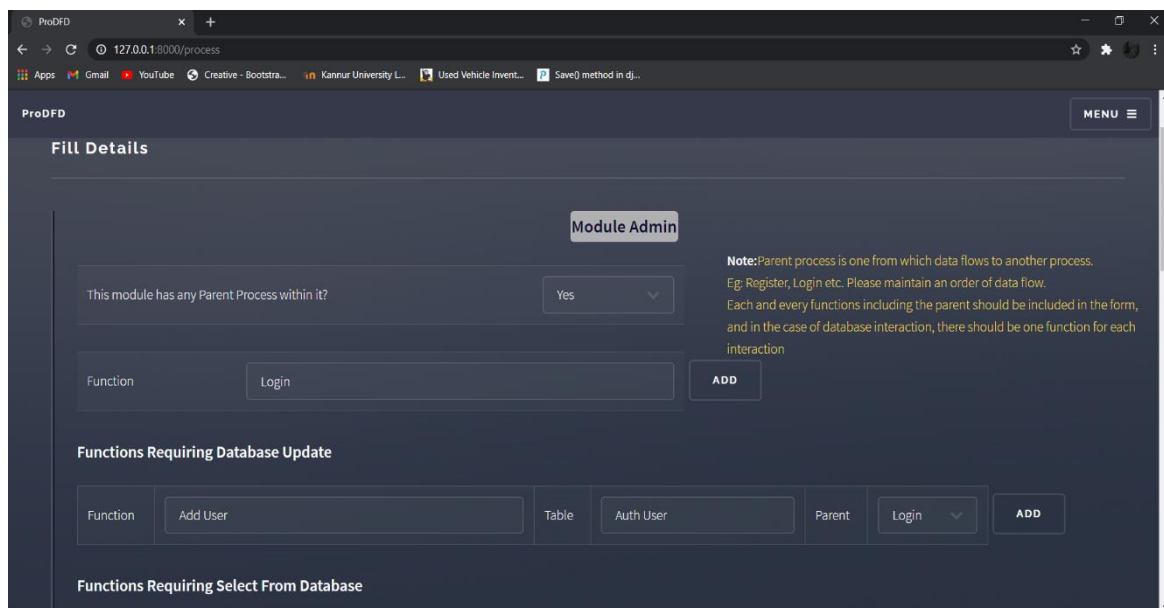
Project Name ProDFD

Name the modules Admin

User ADD

SUBMIT

Form2



ProDFD

MENU

Fill Details

Module Admin

This module has any Parent Process within it? Yes

Function Login ADD

Functions Requiring Database Update

Function	Table	Parent
Add User	Auth User	Login

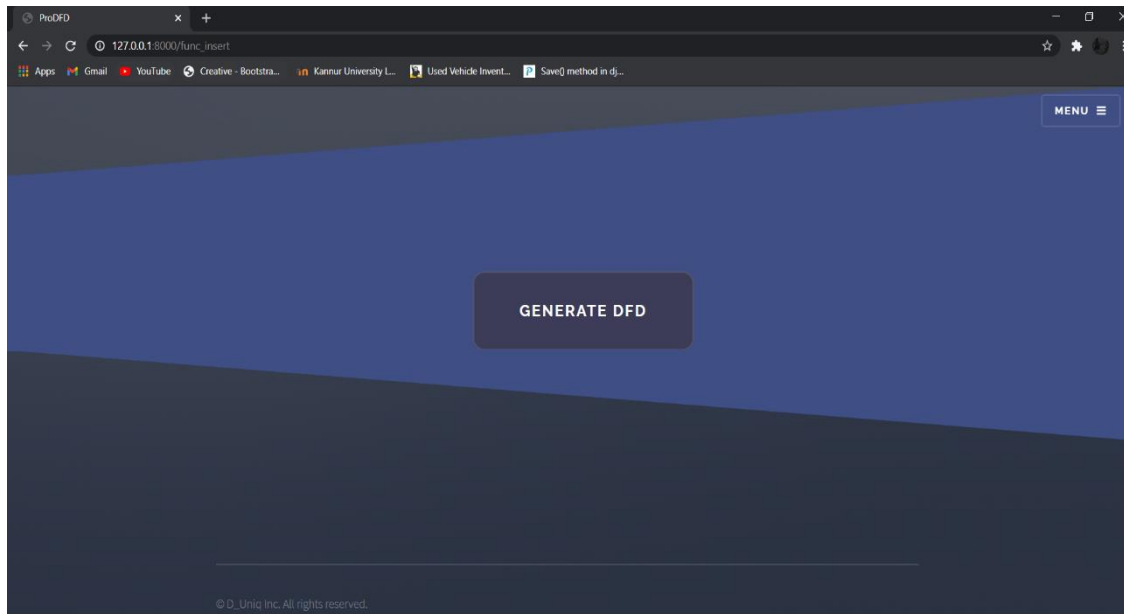
ADD

Functions Requiring Select From Database

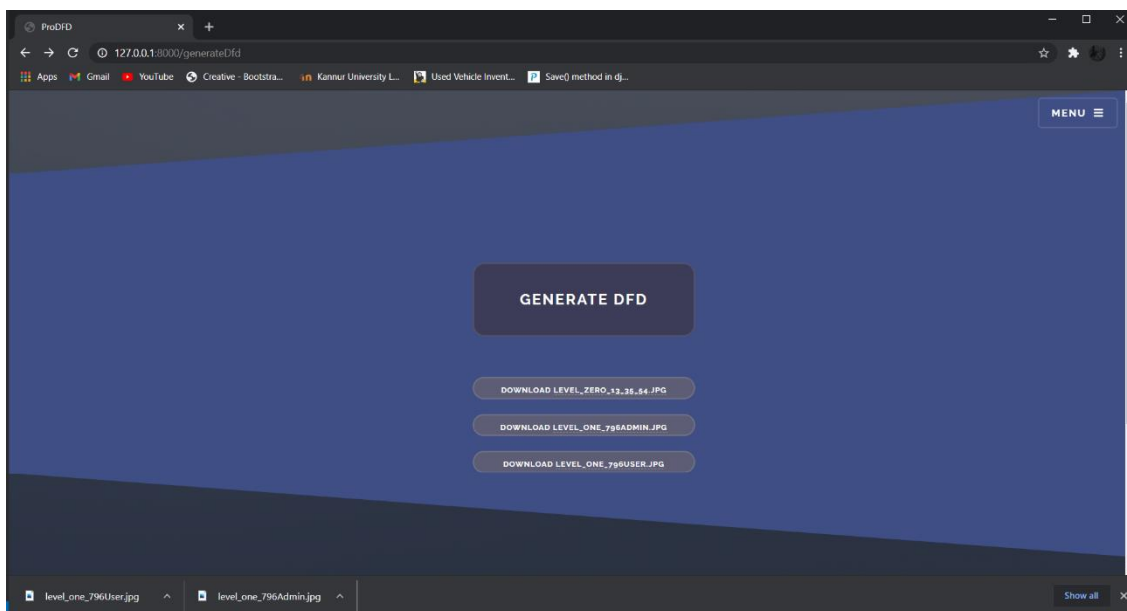
Note: Parent process is one from which data flows to another process.
Eg: Register, Login etc. Please maintain an order of data flow.
Each and every functions including the parent should be included in the form,
and in the case of database interaction, there should be one function for each interaction

Here parent process is collected 1st and then childs are added using select tag

Generate DFD

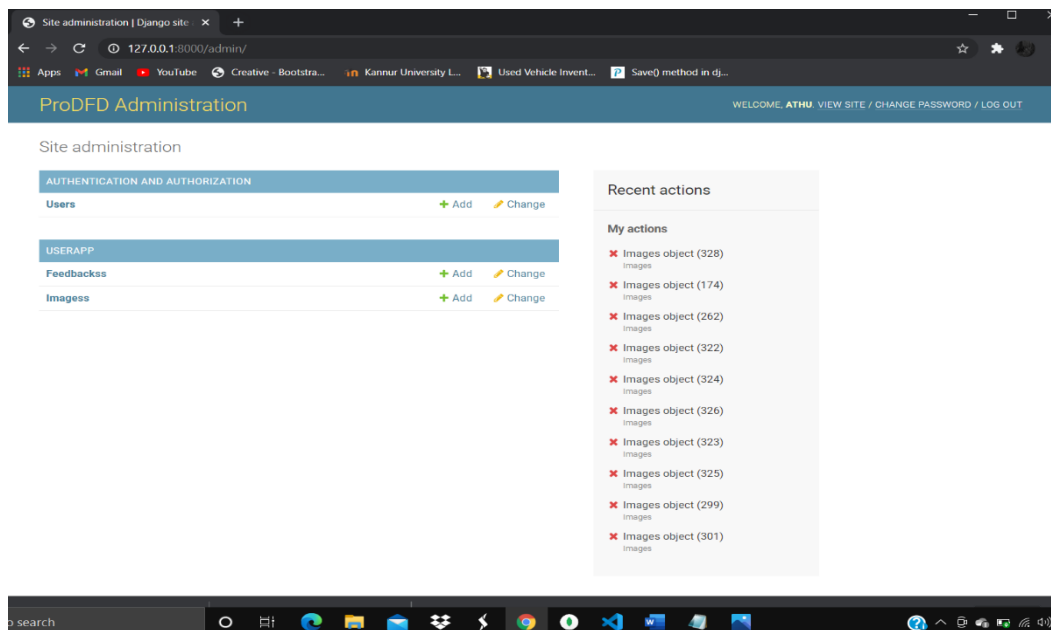


The button generates the DFDs and returns the images for downloading it, which can be seen below:

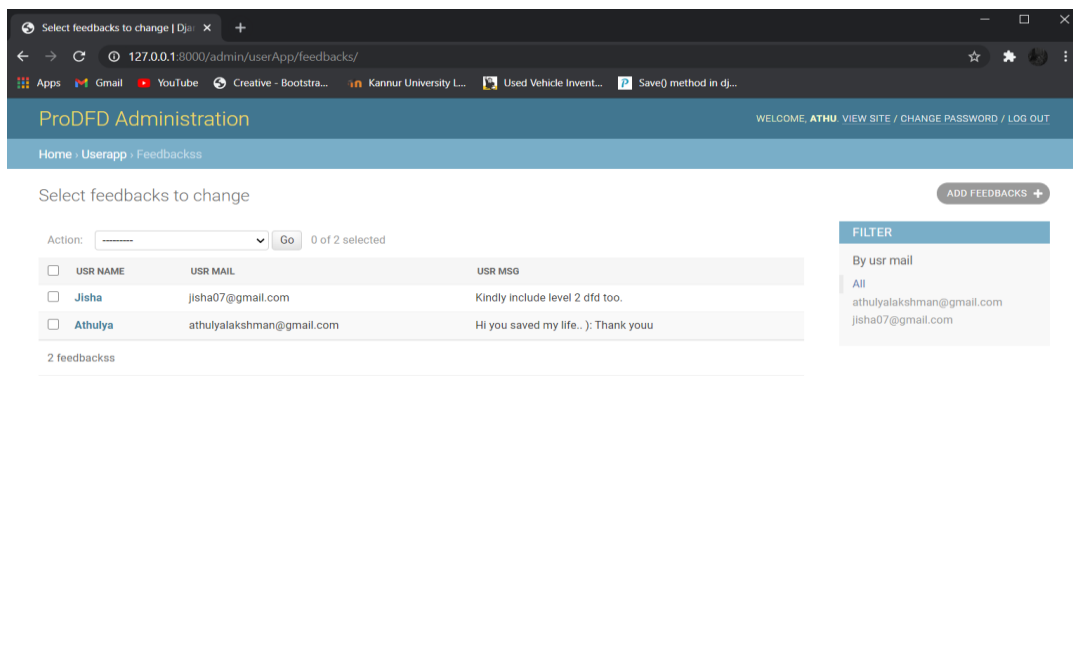


ProDFD Administration Home

The Django administration was enough to manage ProDFD. Its default features fulfilled multiple requirements of this site. The home page looks like this:

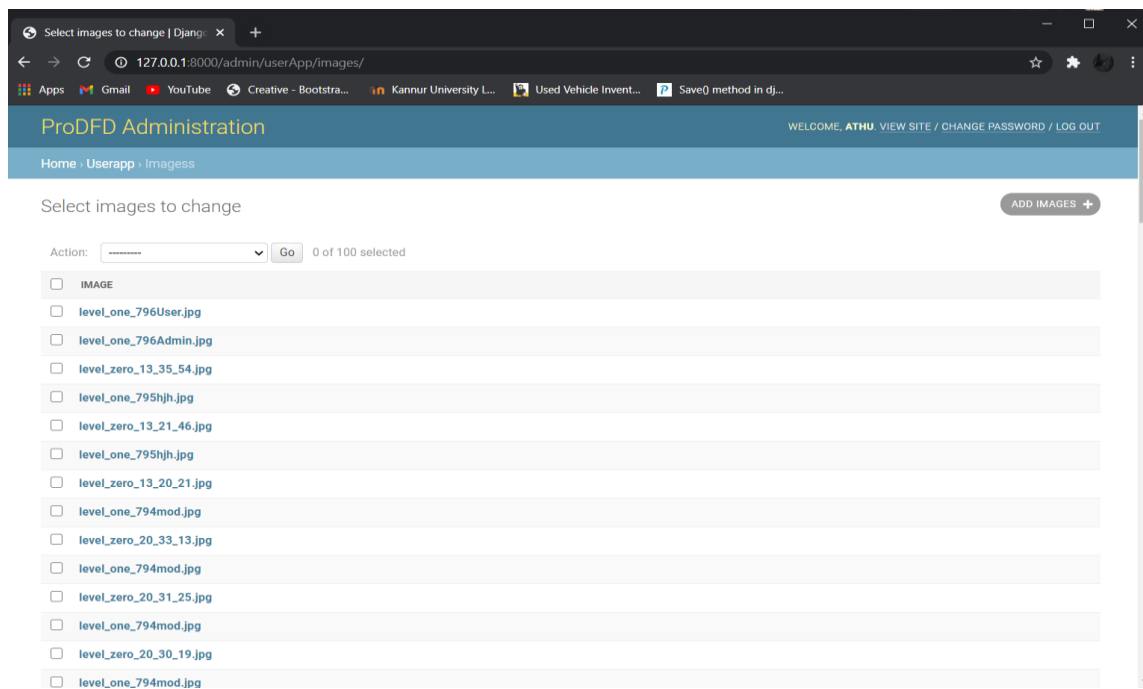


The collections: feedbacks and images are made available to manage from here. User details are also available to manage. Feedbacks are displayed as follows:

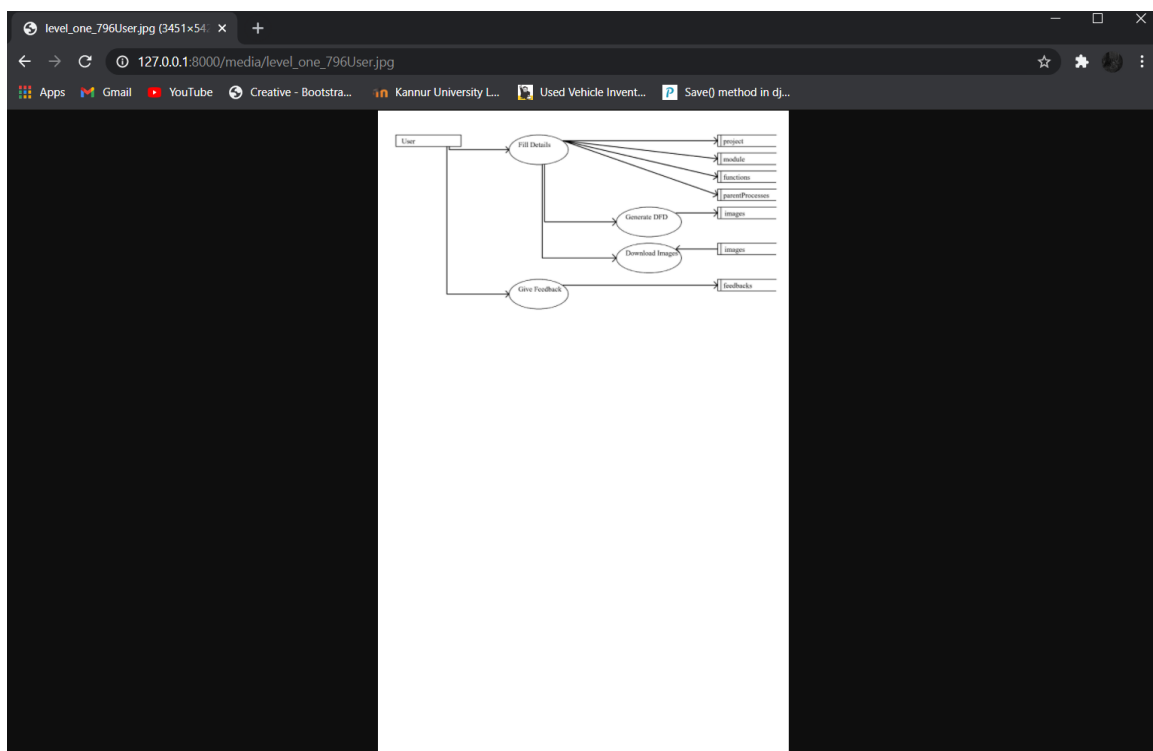


Feedbacks can also be filtered based on the email-id

The image collection is displayed as:



The image can be viewed by touching it, and it is displayed as :



FUTURE ENHANCEMENT

Programmed DFD has covered the immediate need of the developers , even though it can be widened to a level where we can generate much more diagrams related to a software; Menu Tree, Use Case Diagram etc. And also we can also consider the level 2 DFD, instead of congesting it into the level one's child. There by the developers can gauge more useful things out of it.

CONCLUSION

Programmed DFD is a time saver to those who are struggling to meet their project goal and craving to complete the documentation part. It helps to generate the Data Flow Diagram in just few clicks without even generating a user account.

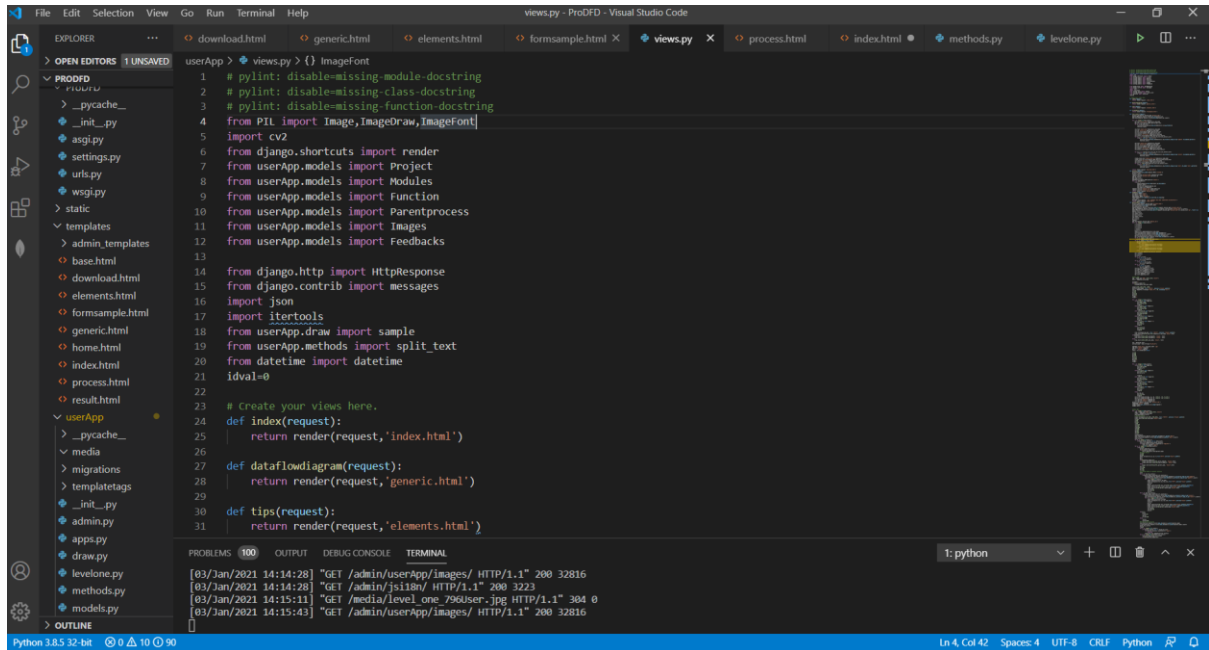
It aims to solve the little yet time consuming part in between the documentation process, in a programmatic way. As well as the time wastage considered in traditional approach, this web application would be a life saving asset for the users. It could meet the aim successfully by generating its own DFD as well.

REFERENCE

- “Download Python”, <https://www.python.org/downloads/>, 2020 [online]
Accessed:[05-Sept-2020]
- “Download MongoDB ”, <https://www.mongodb.com/try#community>, 2020[online],
Accessed:[05-Sept-2020]
- “Download MongoDB Compass”, <https://www.mongodb.com/try/download/compass>,
2020[online], Accessed:[05-Sept-2020]
- “Python Tutorial”,
<https://www.youtube.com/watch?v=QXeEoD0pB3E&list=PLsyeobzWxl7poL9JTVyndKe62ieoN-MZ3>, 2020[online], Accessed:[20-Sept-2020]
- “Django Tutorial”, <https://www.youtube.com/watch?v=OTmQOjSl0eg>, 2020[online],
Accessed:[01-Oct-2020]
- “Djongo Set up”, <https://medium.com/@9cv9official/django-to-mongodb-djongo-or-mongoengine-d9d56b836a3d>, 2020[online], Accessed:[05-Oct-2020]
- “Python Guide”, Python Cook Book: Recipes for Mastering Python - David Beazley & Brian K. Jones , Accessed: [20-Dec-2020]
- “Django Guide”, Django Cookbook Web Development with Django - Step by Step Guide- Beau Curtin, , Accessed: [25-Dec-2020]

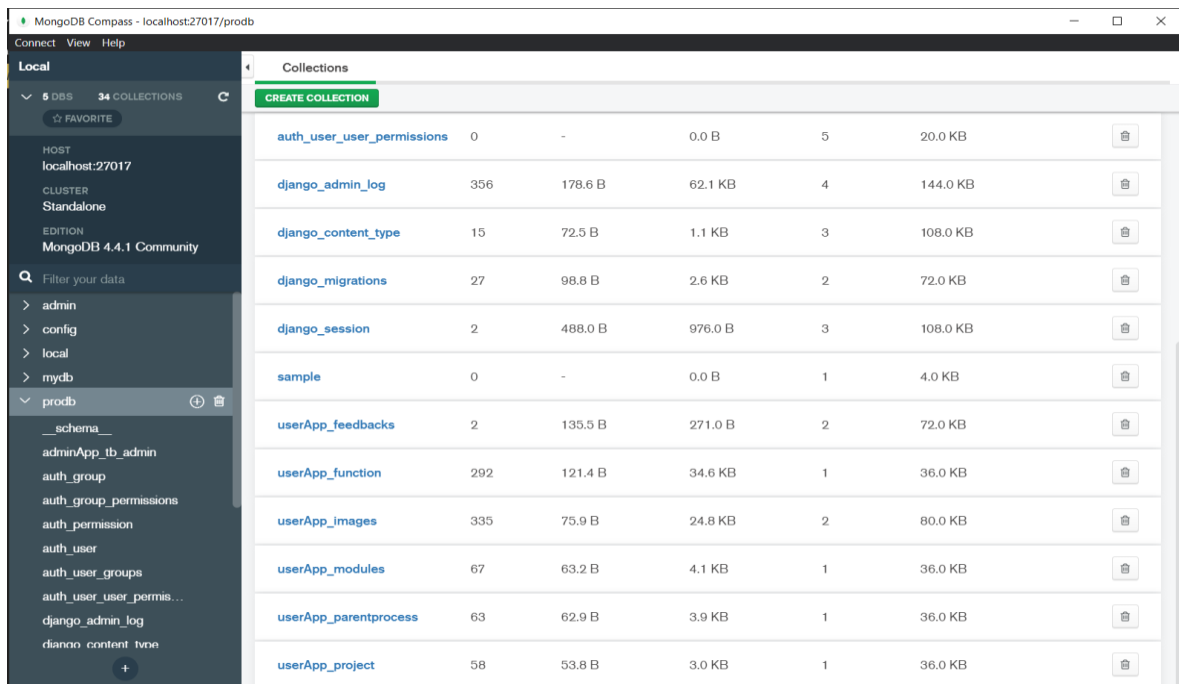
APPENDIX

Visual Studio Code



Python extensions are downloaded from VSCode . pip install command is used to install Django, PIL, CV2 etc from the terminal.

MongoDB



Here data are documents and are kept as Collections inside the database. Inside the collection, documents are arranged with id in embedded data model.

