PROJECT REPORT FOR ‘ECOMMERCE’

|  |  |
| --- | --- |
| project submitted by : | trainer name : |
| 1. Gelson 2. VIKASH 3. SARJIL 4. RAHUL 5. ASHWINI 6. ganesh | VENKATASUBRAMANIAN R |

**PROJECT REQUIREMENT:**

**Django**

Django is a free and open source web application framework, written in Python. A web framework is a set of components that helps we to develop websites faster and easier.

When we're building a website, we always need a similar set of components: a way to handle user authentication (signing up, signing in, signing out), a management panel for wer website, forms, a way to upload files, etc.

Luckily for we, other people long ago noticed that web developers face similar problems when building a new site, so they teamed up and created frameworks (Django being one of them) that give we ready-made components to use.

Frameworks exist to save we from having to reinvent the wheel and to help alleviate some of the overhead when we’re building a new site.

**Why do we need a framework?**

To understand what Django is actually for, we need to take a closer look at the servers. The first thing is that the server needs to know that we want it to serve we a web page.

Imagine a mailbox (port) which is monitored for incoming letters (requests). This is done by a web server. The web server reads the letter and then sends a response with a webpage. But when we want to send something, we need to have some content. And Django is something that helps we create the content.

**What happens when someone requests a website from wer server?**

When a request comes to a web server, it's passed to Django which tries to figure out what is actually requested. It takes a web page address first and tries to figure out what to do. This part is done by Django's **urlresolver** (note that a website address is called a URL – Uniform Resource Locator – so the name *urlresolver* makes sense). It is not very smart – it takes a list of patterns and tries to match the URL. Django checks patterns from top to bottom and if something is matched, then Django passes the request to the associated function (which is called *view*).

Imagine a mail carrier with a letter. She is walking down the street and checks each house number against the one on the letter. If it matches, she puts the letter there. This is how the urlresolver works!

In the *view* function, all the interesting things are done: we can look at a database to look for some information. Maybe the user asked to change something in the data? Like a letter saying, "Please change the description of my job." The *view* can check if we are allowed to do that, then update the job description for we and send back a message: "Done!" Then the *view* generates a response and Django can send it to the user's web browser.

### Components

Despite having its own nomenclature, such as naming the callable objects generating the [HTTP](https://en.wikipedia.org/wiki/HTTP) responses "views",[[4]](https://en.wikipedia.org/wiki/Django_(web_framework)#cite_note-faq-mvc-4) the core Django framework can be seen as an [MVC](https://en.wikipedia.org/wiki/Model-view-controller) architecture.[[5]](https://en.wikipedia.org/wiki/Django_(web_framework)#cite_note-djangobook-mvc-5) It consists of an [object-relational mapper](https://en.wikipedia.org/wiki/Object-relational_mapping) (ORM) that mediates between [data models](https://en.wikipedia.org/wiki/Data_modeling) (defined as Python classes) and a [relational database](https://en.wikipedia.org/wiki/Relational_database) ("**M**odel"), a system for processing HTTP requests with a [web templating system](https://en.wikipedia.org/wiki/Web_template_system) ("**V**iew"), and a [regular-expression](https://en.wikipedia.org/wiki/Regular_expression)-based [URL](https://en.wikipedia.org/wiki/Uniform_Resource_Locator) dispatcher ("**C**ontroller").

Also included in the core framework are:

* a lightweight and standalone web server for development and testing
* a form serialization and validation system that can translate between [HTML](https://en.wikipedia.org/wiki/HTML) forms and values suitable for storage in the database
* a template system that utilizes the concept of [inheritance](https://en.wikipedia.org/wiki/Inheritance_(object-oriented_programming)) borrowed from object-oriented programming
* a [caching](https://en.wikipedia.org/wiki/Web_cache) framework that can use any of several cache methods
* support for [middleware](https://en.wikipedia.org/wiki/Middleware) classes that can intervene at various stages of request processing and carry out custom functions
* an internal dispatcher system that allows components of an application to communicate events to each other via pre-defined signals
* an [internationalization](https://en.wikipedia.org/wiki/Internationalization_and_localization) system, including translations of Django's own components into a variety of languages
* a [serialization](https://en.wikipedia.org/wiki/Serialization) system that can produce and read [XML](https://en.wikipedia.org/wiki/XML) and/or [JSON](https://en.wikipedia.org/wiki/JSON) representations of Django model instances
* a system for extending the capabilities of the template engine
* an interface to Python's built-in [unit test](https://en.wikipedia.org/wiki/Unit_test) framework

### Bundled applications

The main Django distribution also bundles a number of applications in its "contrib" package, including:

* an extensible authentication system
* the dynamic administrative interface
* tools for generating [RSS](https://en.wikipedia.org/wiki/RSS_(file_format)) and [Atom](https://en.wikipedia.org/wiki/Atom_(standard)) syndication feeds
* a "Sites" framework that allows one Django installation to run multiple websites, each with their own content and applications
* tools for generating [Google Sitemaps](https://en.wikipedia.org/wiki/Google_Sitemaps)
* built-in mitigation for [cross-site request forgery](https://en.wikipedia.org/wiki/Cross-site_request_forgery), [cross-site scripting](https://en.wikipedia.org/wiki/Cross-site_scripting), [SQL injection](https://en.wikipedia.org/wiki/SQL_injection), [password cracking](https://en.wikipedia.org/wiki/Password_cracking) and other typical web attacks, most of them turned on by default[[18]](https://en.wikipedia.org/wiki/Django_(web_framework)#cite_note-18)[[19]](https://en.wikipedia.org/wiki/Django_(web_framework)#cite_note-19)
* a framework for creating [GIS](https://en.wikipedia.org/wiki/Geographic_information_system) applications

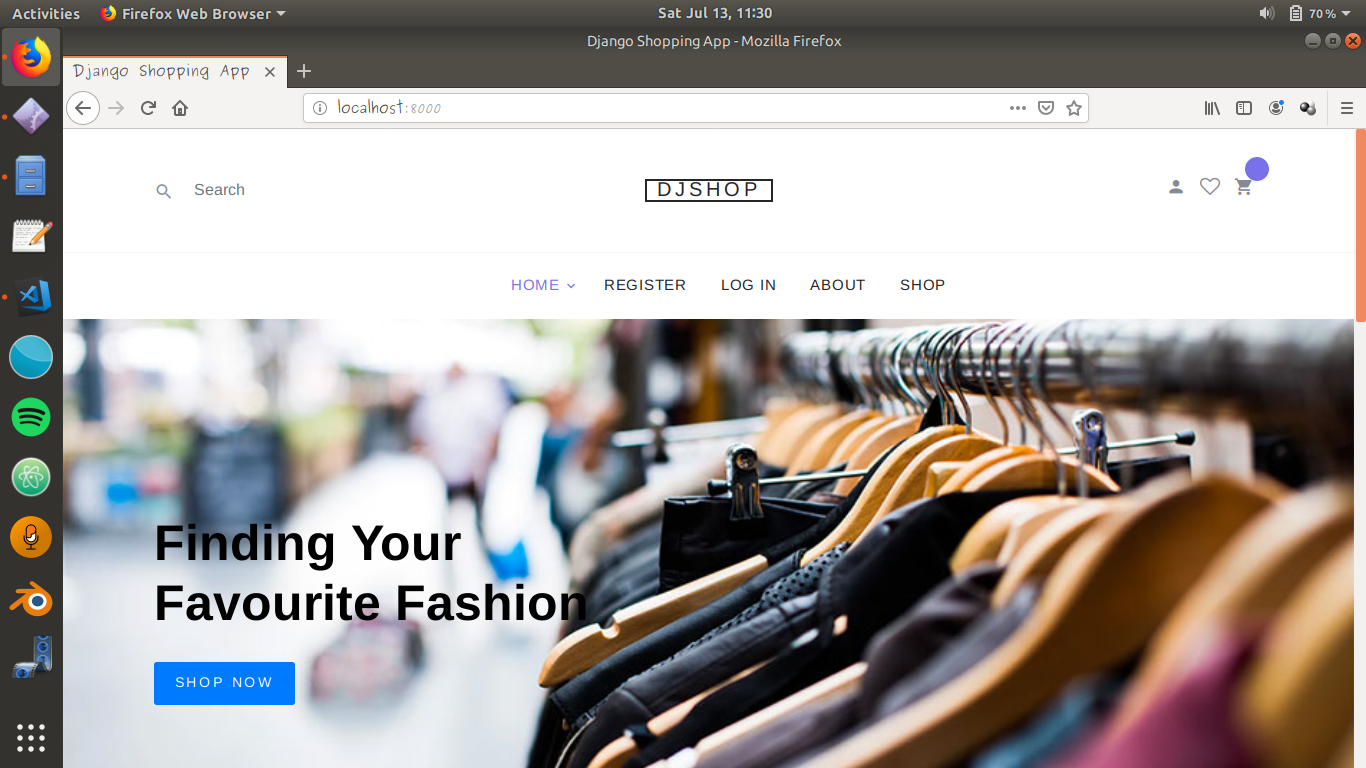
### Server arrangements

Django can be run in conjunction with [Apache](https://en.wikipedia.org/wiki/Apache_HTTP_Server), [Nginx](https://en.wikipedia.org/wiki/Nginx) using [WSGI](https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface), [Gunicorn](https://en.wikipedia.org/wiki/Gunicorn), or [Cherokee](https://en.wikipedia.org/wiki/Cherokee_(Webserver)) using flup (a Python module).[[23]](https://en.wikipedia.org/wiki/Django_(web_framework)#cite_note-23)[[24]](https://en.wikipedia.org/wiki/Django_(web_framework)#cite_note-24) Django also includes the ability to launch a [FastCGI](https://en.wikipedia.org/wiki/FastCGI) server, enabling use behind any web server which supports FastCGI, such as [Lighttpd](https://en.wikipedia.org/wiki/Lighttpd) or [Hiawatha](https://en.wikipedia.org/wiki/Hiawatha_(web_server)). It is also possible to use other [WSGI](https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface)-compliant web servers.[[25]](https://en.wikipedia.org/wiki/Django_(web_framework)#cite_note-25) Django officially supports four database backends: [PostgreSQL](https://en.wikipedia.org/wiki/PostgreSQL), [MySQL](https://en.wikipedia.org/wiki/MySQL), [SQLite](https://en.wikipedia.org/wiki/SQLite), and [Oracle](https://en.wikipedia.org/wiki/Oracle_Database). [Microsoft SQL Server](https://en.wikipedia.org/wiki/Microsoft_SQL_Server) can be used with django-mssql on [Microsoft](https://en.wikipedia.org/wiki/Microsoft) [operating systems](https://en.wikipedia.org/wiki/Operating_systems),[[26]](https://en.wikipedia.org/wiki/Django_(web_framework)#cite_note-26) while similarly external backends exist for [IBM Db2](https://en.wikipedia.org/wiki/IBM_Db2),[[27]](https://en.wikipedia.org/wiki/Django_(web_framework)#cite_note-27) [SQL Anywhere](https://en.wikipedia.org/wiki/SQL_Anywhere)[[28]](https://en.wikipedia.org/wiki/Django_(web_framework)#cite_note-28) and [Firebird](https://en.wikipedia.org/wiki/Firebird_(database_server)).[[29]](https://en.wikipedia.org/wiki/Django_(web_framework)#cite_note-29) There is a [fork](https://en.wikipedia.org/wiki/Fork_(software_development)) named django-nonrel, which supports [NoSQL](https://en.wikipedia.org/wiki/NoSQL_(concept)) databases, such as [MongoDB](https://en.wikipedia.org/wiki/MongoDB) and [Google App Engine](https://en.wikipedia.org/wiki/Google_App_Engine)'s Datastore.[[30]](https://en.wikipedia.org/wiki/Django_(web_framework)#cite_note-30)

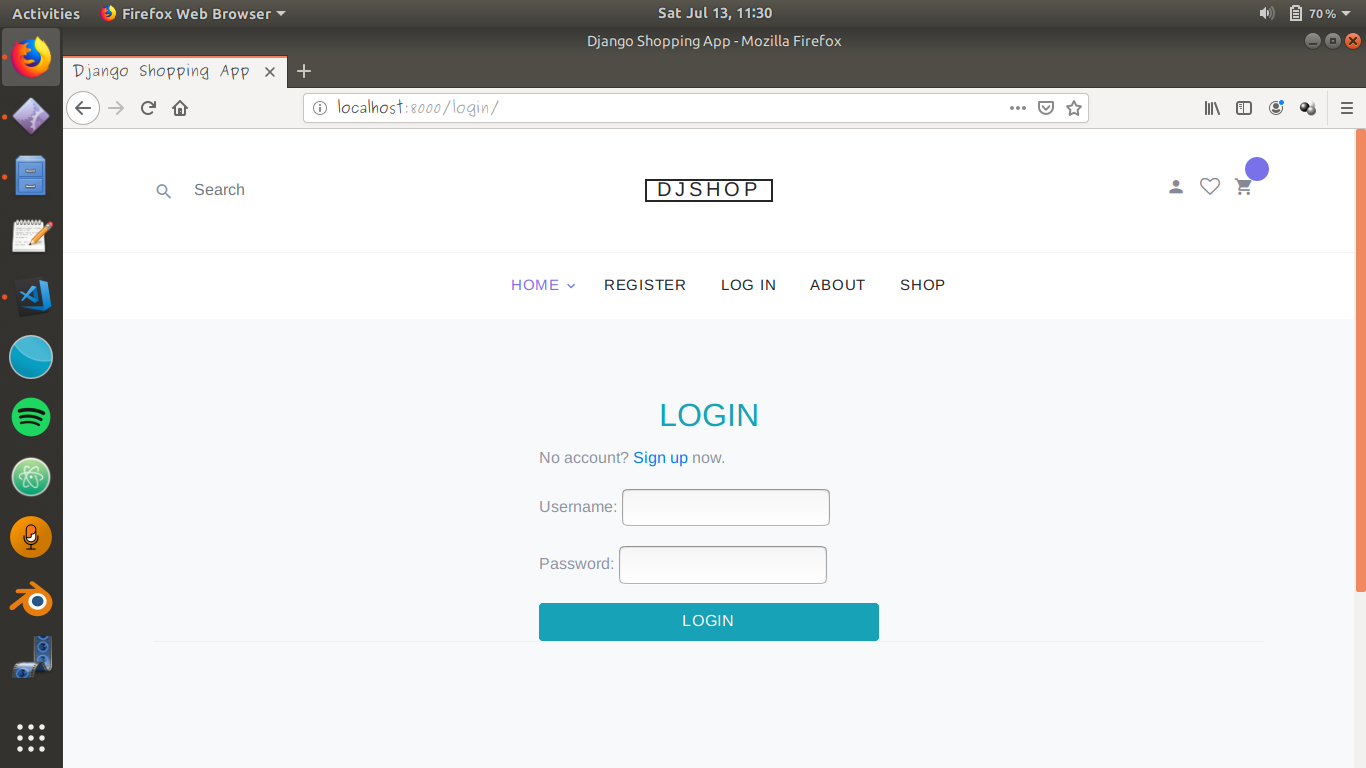
Django may also be run in conjunction with [Jython](https://en.wikipedia.org/wiki/Jython) on any [Java EE](https://en.wikipedia.org/wiki/Java_EE) application server such as [GlassFish](https://en.wikipedia.org/wiki/GlassFish) or [JBoss](https://en.wikipedia.org/wiki/JBoss). In this case django-jython must be installed in order to provide [JDBC](https://en.wikipedia.org/wiki/JDBC) drivers for database connectivity, which also can provide functionality to compile Django in to a .war suitable for deployment.[[31]](https://en.wikipedia.org/wiki/Django_(web_framework)#cite_note-31)

[Google App Engine](https://en.wikipedia.org/wiki/Google_App_Engine) includes support for Django version 1.x.x[[32]](https://en.wikipedia.org/wiki/Django_(web_framework)" \l "cite_note-32) as one of the bundled frameworks.

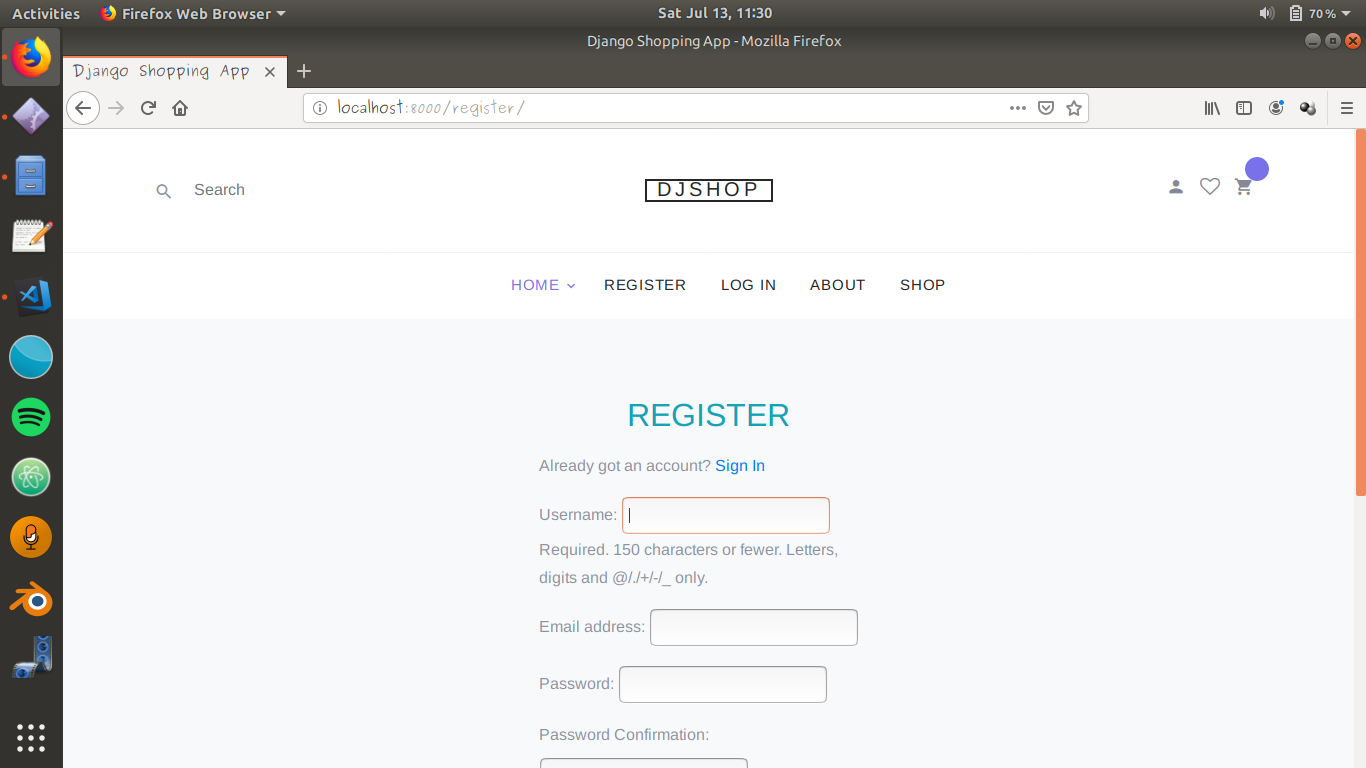
**PROJECT’S TEST ANALYSIS**



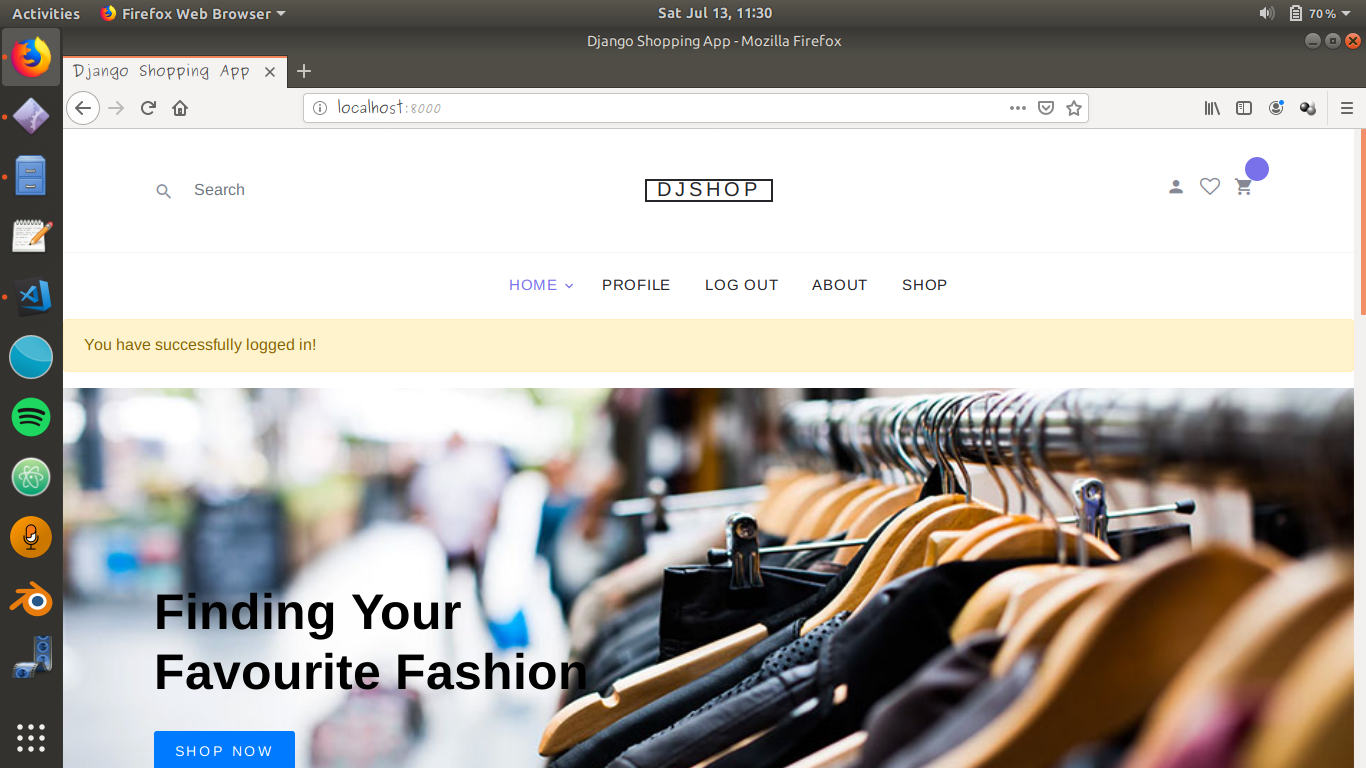
Home page.



Login page for registered users.



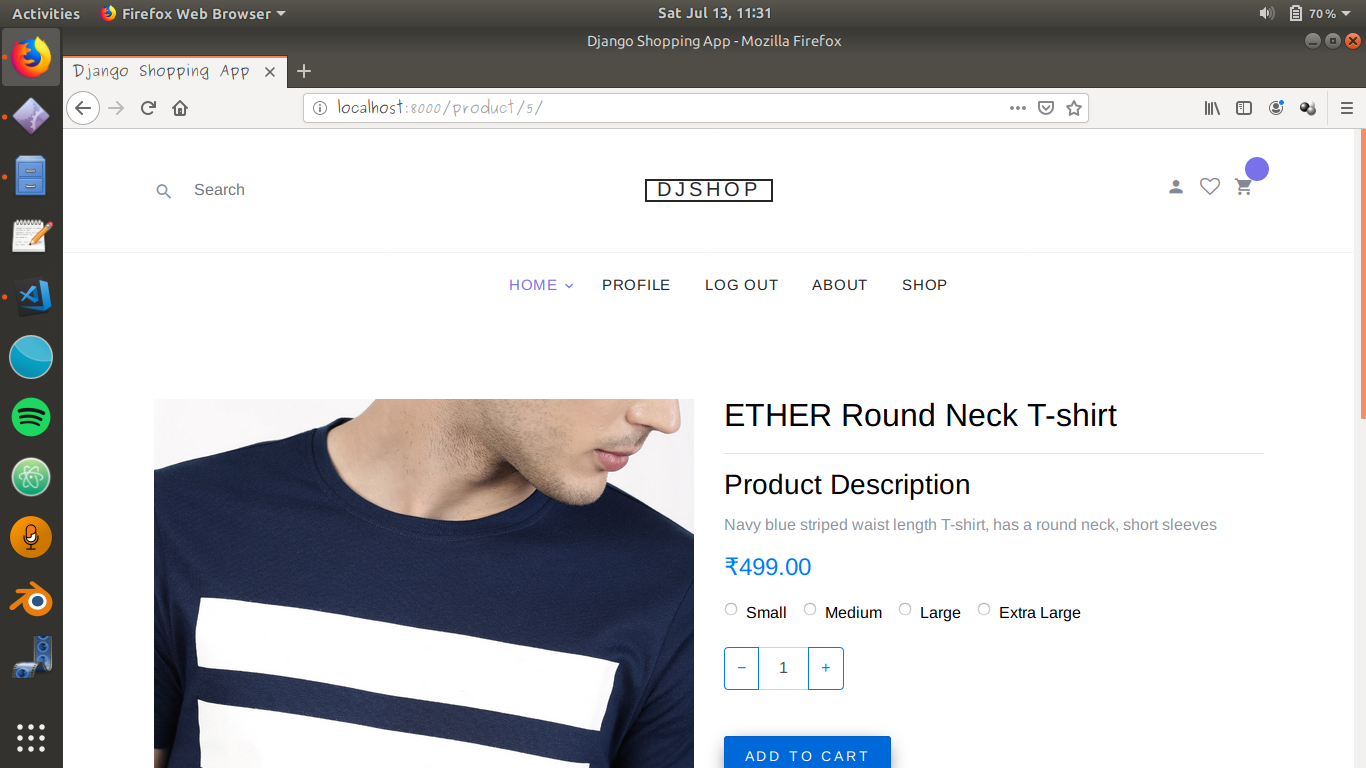
Registration page for new user.



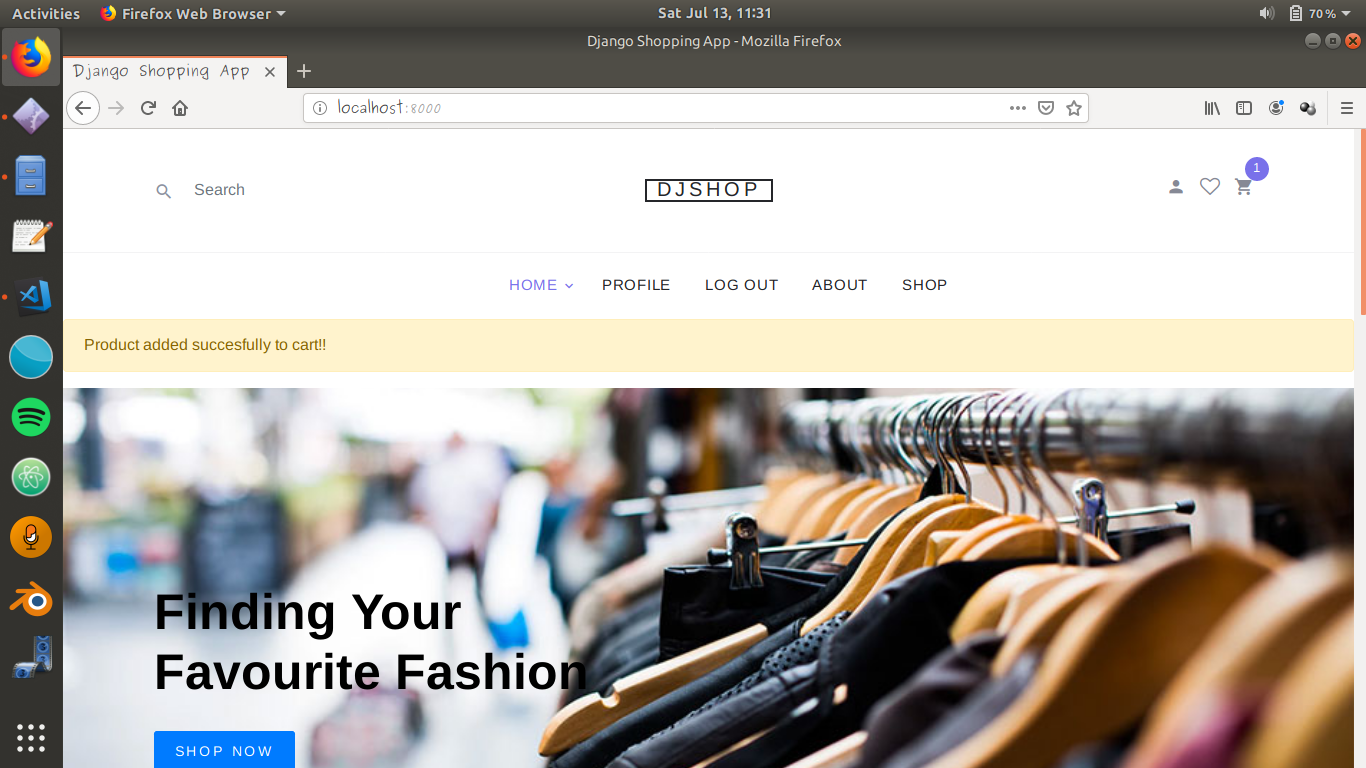
Succesfully logged in message.



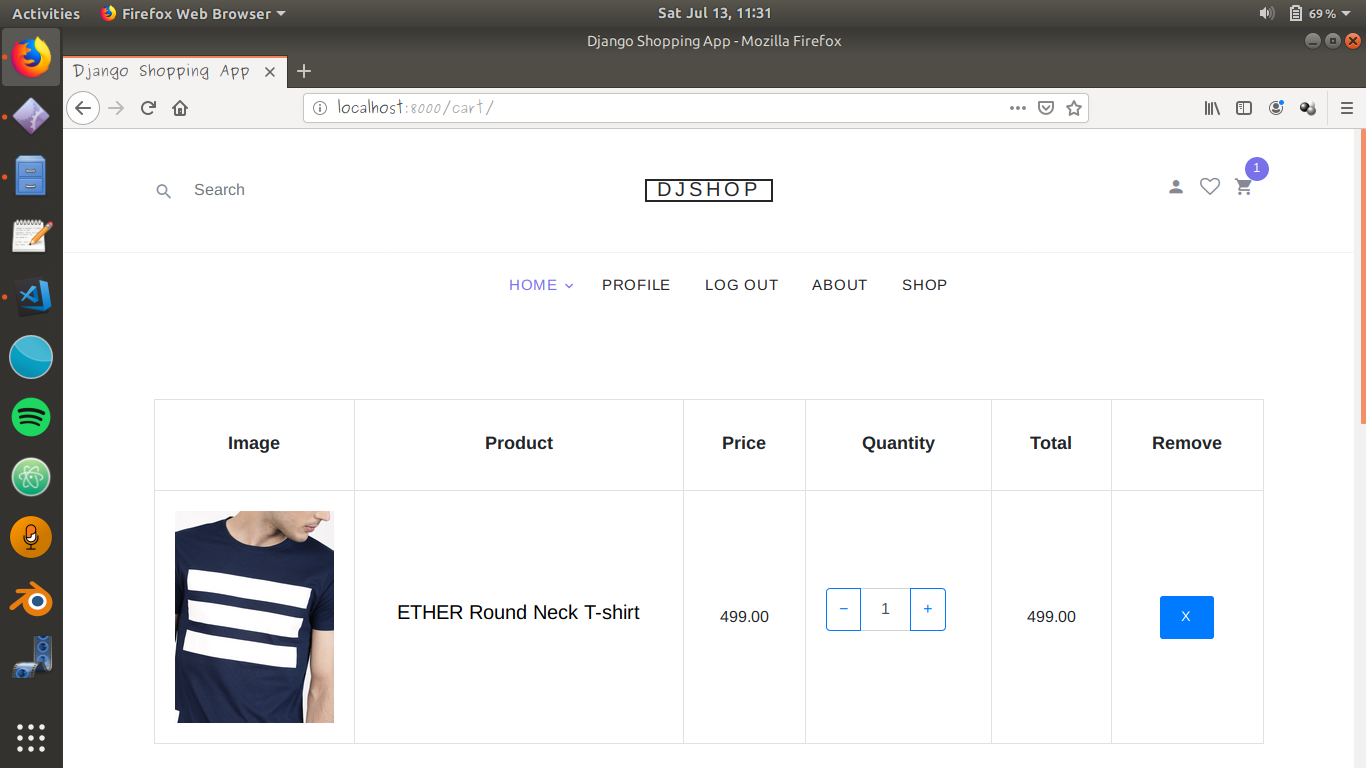
Some of the product image and their price description.



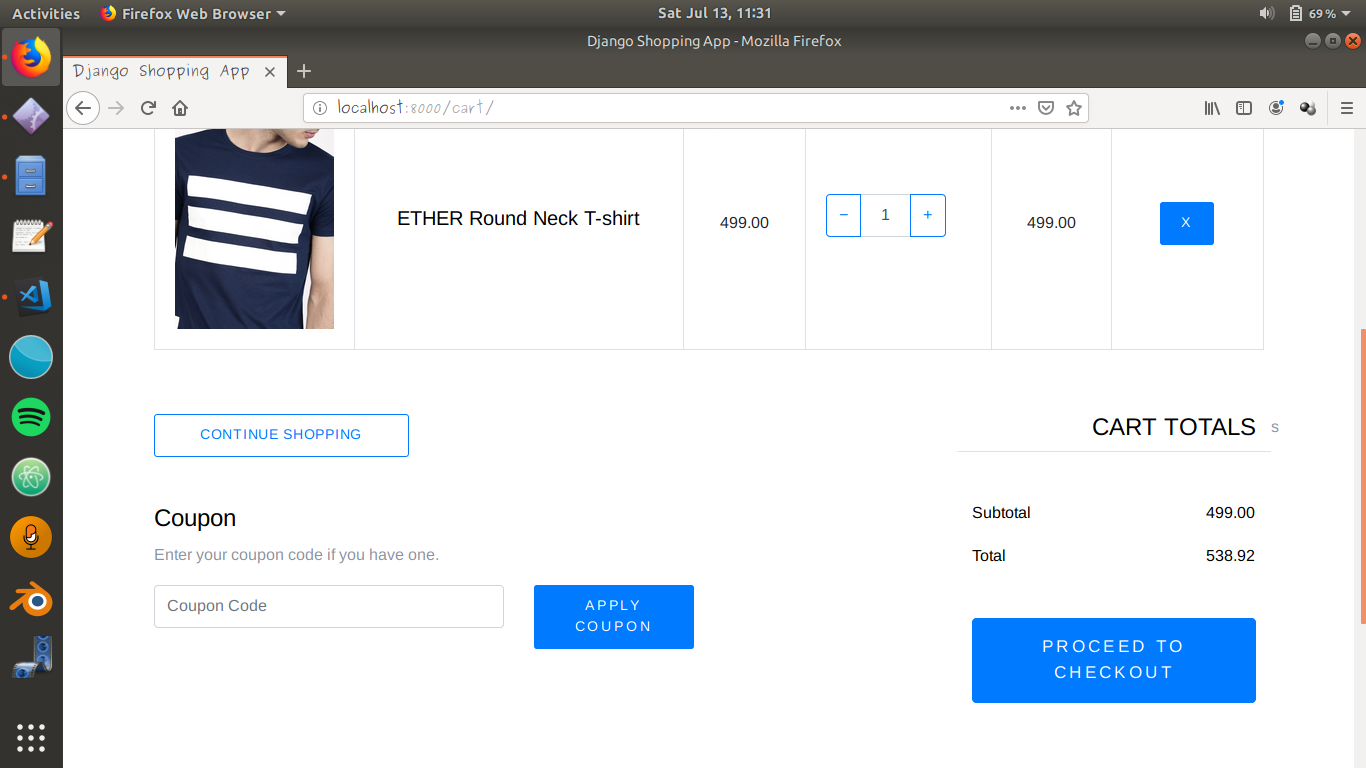
Product Size selection.



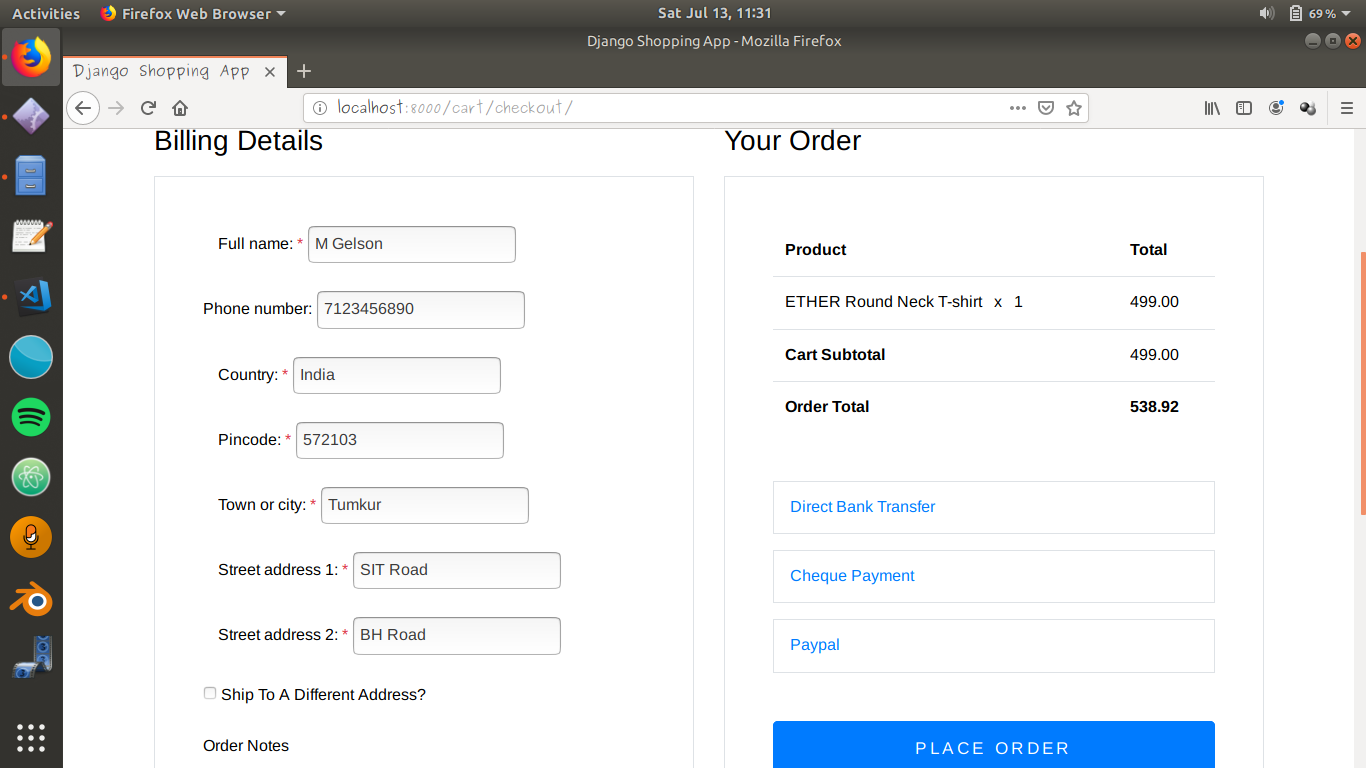
Product added successfully message when added.



Data of the product we added and can also remove the product.

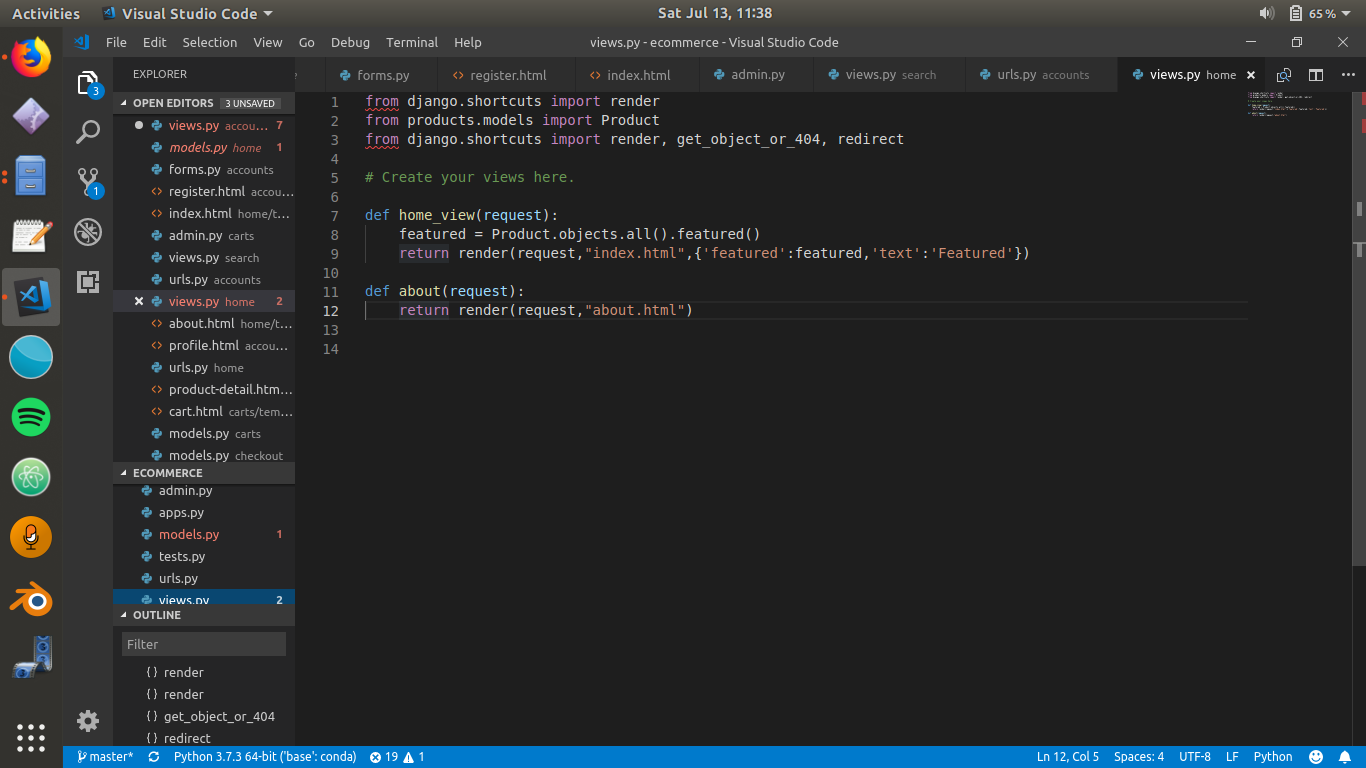


Product Order checkout.

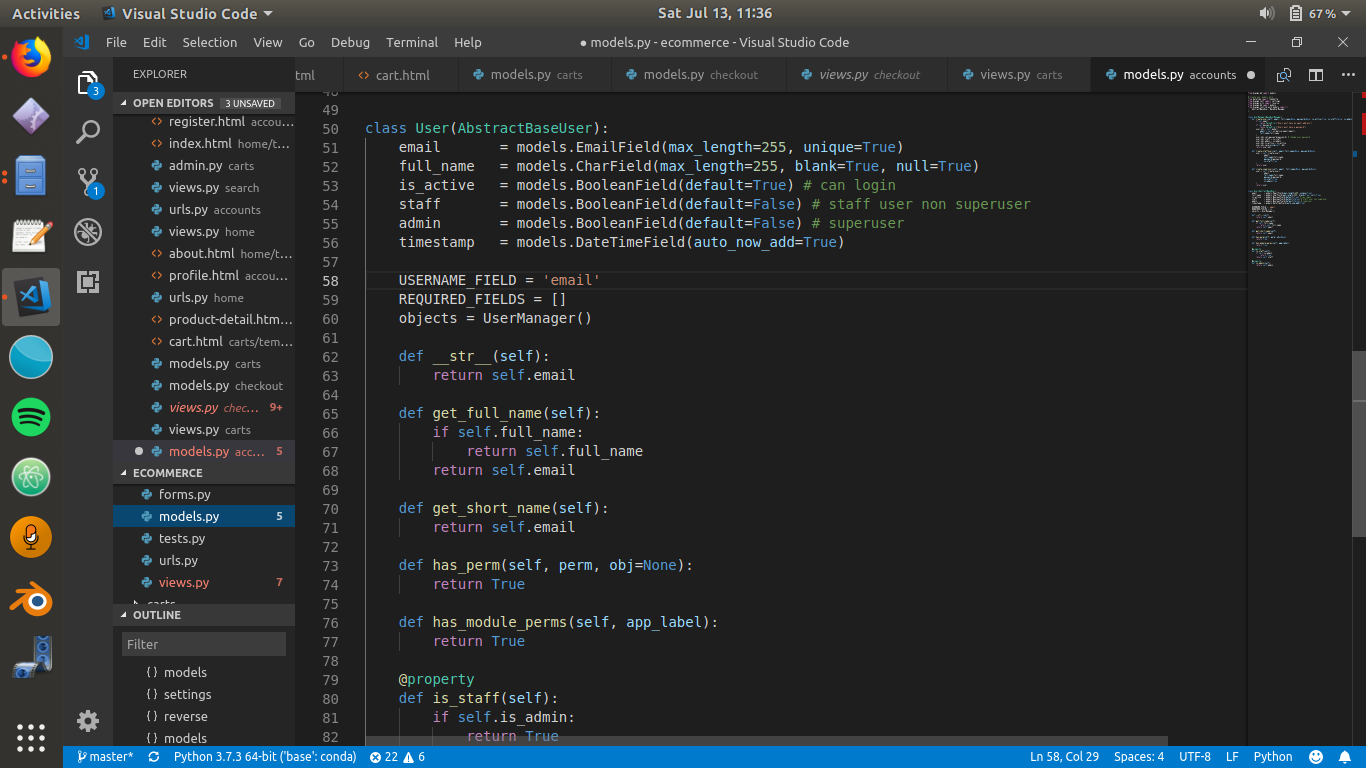


Order Payment page.

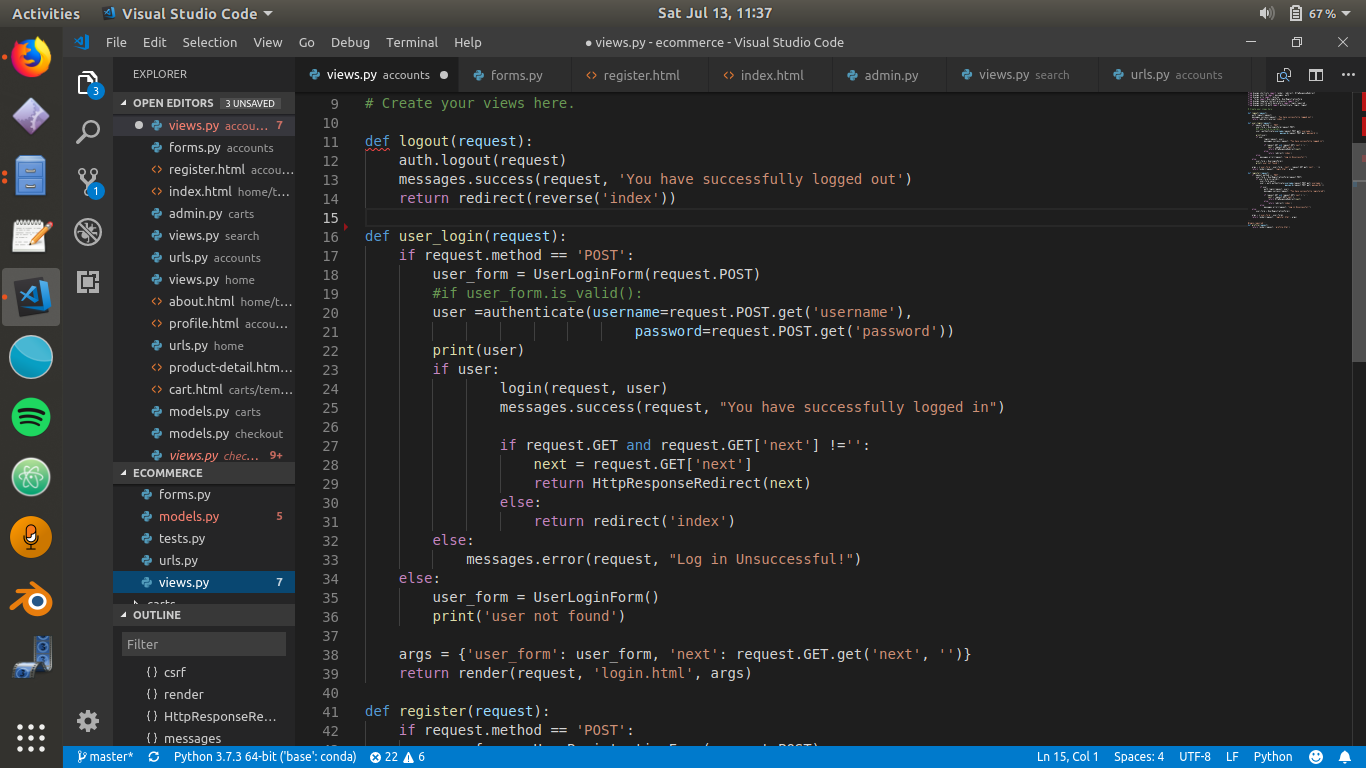
**CODE AND FUNCTIONALITY**



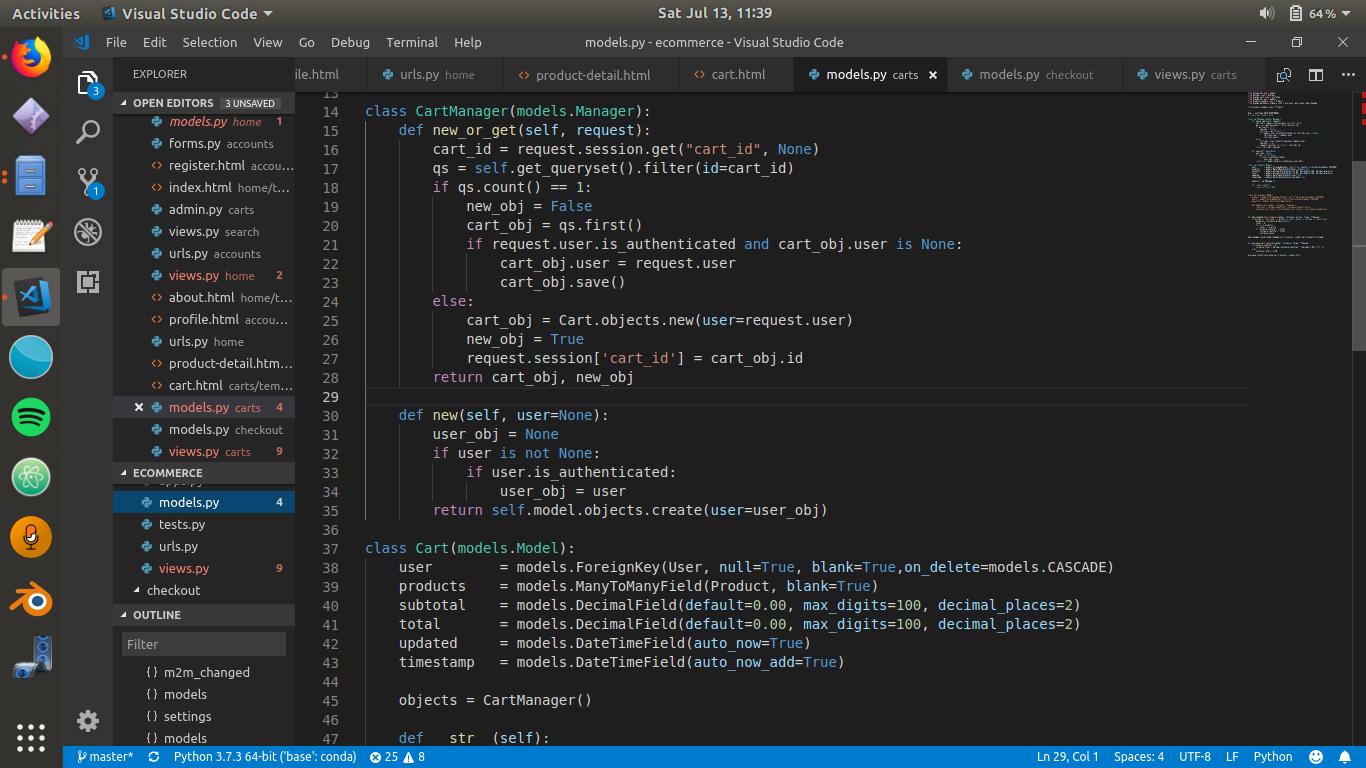
Home View with function home\_view and about.



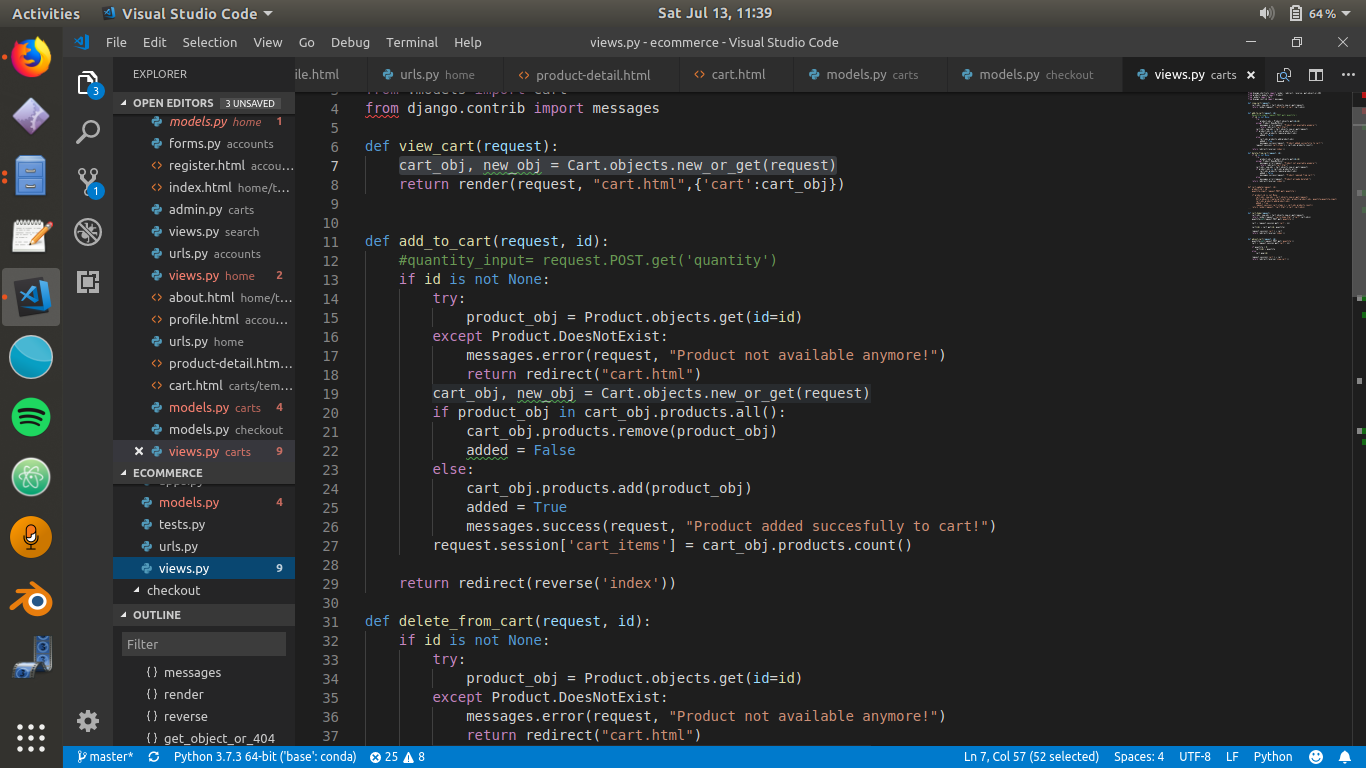
Account model with User class.



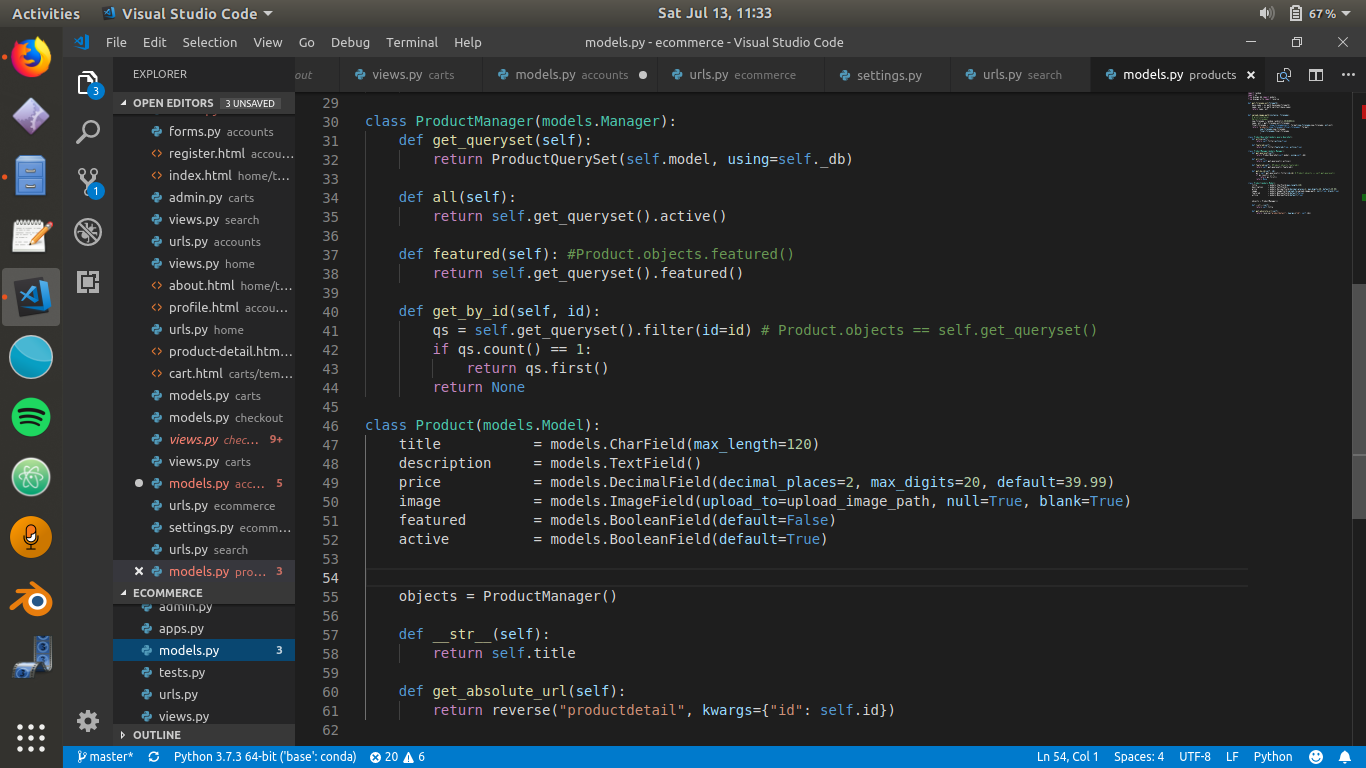
Account view with login, logout and registration function.



Cart model with Cart Manager class.



Cart view with view cart function, add\_to\_cart, delete from cart.



Product Model with Product manager class and product class.