**5. Design Overview:**

**5.1. Design Rationale:**

Currently, we are building a web application for this application to be successful we think it is necessary that the application holds below mentioned qualities. It should:

* Work as per desired during peak hours with multiple users accessing the website simultaneously.
* Model the data access through a centralized data store to have high security enforced through a central server.
* Support high volume usage by having load balancing and fault tolerance mechanisms.
* Support accessing the application from different environments (different devices, browsers etc.)
* Be scalable and each tier should scale horizontally.
* Have high performance. Presentation tier should be able to cache requests, so network utilization is minimized, and the load is reduced on the Application and Data tiers.
* Be highly available. We should be able to make each layer fault-tolerant independently. If the Application tier server is down and caching is sufficient, the Presentation tier should be able to process web requests using the cache. We should be able to add redundant database servers to have fault-tolerance for DB tier.

Given the above needs, we have finalized to design our architecture following **client-server**

**Architectural style**, and to use **4-tier architectural pattern** model.

Along with above architectural needs, it is necessary that application development follows certain good practices as highlighted below.

* Separation of concerns - business logic should be independent and should be able to be used from different presentation tiers (mobile, tablets, desktops).
* Parallel development by separate team’s - developers of UI and business logic should be able to focus exclusively on their part.
* Test driven - should allow easy unit/ functional/ integration testing for large scale applications.

We have decided to go ahead with **MVC pattern** to take into consideration the above developmental needs.

Overall, we have planned to develop our application using **Django framework (Python)** which satisfies our design need of using client-server architectural style and MVC pattern. Django uses a “shared-nothing” architecture, which means you can add hardware at any level – database servers, caching servers or Web/application servers. We have planned to use below tools and technologies for development of our application.

Table 5.1.1

|  |  |
| --- | --- |
| Server side programming | Python 2.7.11 |
| Framework | Django 1.8.12 |
| Client side programming | HTML, CSS, JavaScript |
| Database | MySql |
| Document sharing | Google Docs |
| Cloud Servers | AWS |
| Version Control | Git |

**5.2. Software Architecture:**

**5.2.1 High-level Architecture:**

The high-level architecture (HLA) is a general purpose architecture for distributed computer simulation systems. Below is the high-level architecture for our project.

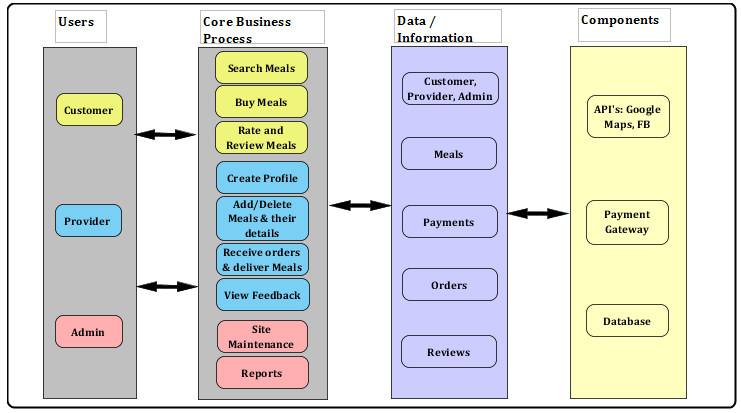


Figure x.x.x

**5.2.2 4-tier Architecture:**

In software engineering, multitier (n-tier) architecture is a client-server architecture. Basically here the presentation, application processing (business logic) and data management functions are physically separated. Below diagram indicates various layers in our application.

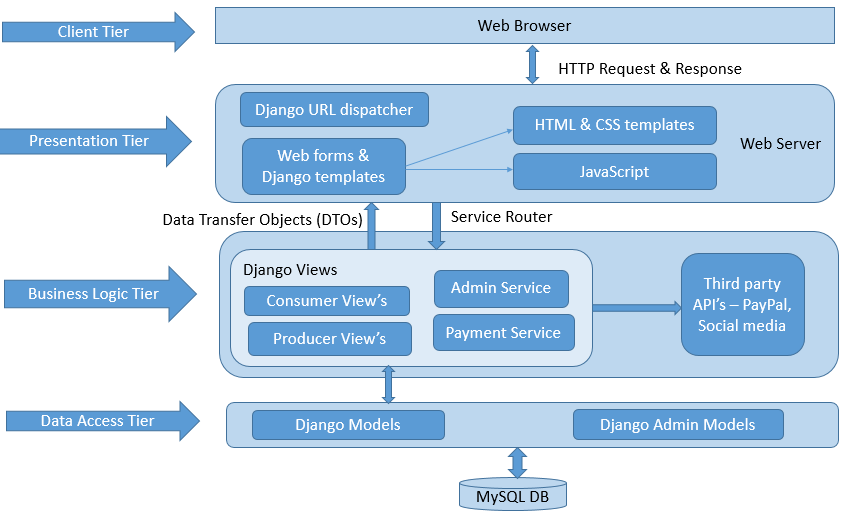


Figure x.x.x

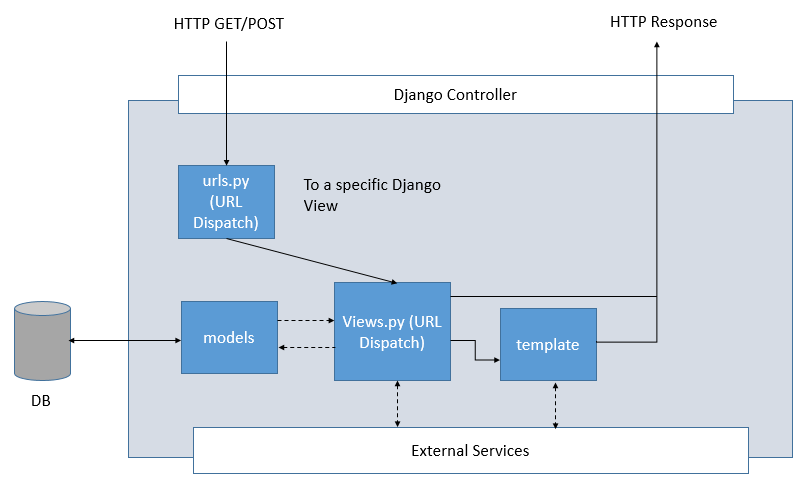
The first tier is the Client tier, facing the user. It is called user interface (UI), in our case it’s the web browser which our user will use to access our application from his machine (client).

Presentation tiers mainly helps to translate tasks and results to something the user can understand. It communicates with other tiers by which it puts out the results to the browser/client tier and all other tiers in the network. Dynamic HTML & CSS templates, and JavaScript are used to implement this layer in our application. The URL dispatcher (urls.py) maps the requested URL to a view function and calls it. This tier interacts with the Business logic tier through a set of services. Presentation tier holds the look and feel of the web application.

Business logic tier contains the business logic and algorithms (basically all the functionality that the user is expecting), and is comprised of several components. The Django views module (usually views.py) shown in the above figure performs the requested action, typically involves reading or writing to the database. It may include other tasks as well. They handle the functionality related to meals, consumers and providers etc. All these views interact with other third party components like Facebook and PayPal for user authentication and payment processing respectively.

Data Access tier holds the Django models and Django Admin models. The model (usually models.py) defines the data in Python code and interacts with it. We will be capturing all the necessary data required by the website in database tables. We are using the MySQL database here. Business logic tier interacts with this layer to perform CRUD (Create, Read, Update, Delete) operations on objects/records.

5.2.3 MVC Architecture:



Django is a free and open source web application framework, written in Python, which follows the Model-View-Controller architectural pattern.

The data access logic, business logic, and presentation logic – comprise a concept that’s sometimes called the Model-View-Controller (MVC) pattern of software architecture.

Django controller handles HTTP requests and responses. Django reads a settings file so that it knows what to load and set up. Django reads the URL config file that tells it what to do with the incoming requests from the browser. The urls.py resolves the incoming request (by matching it with regex) and redirects it to the appropriate view.

In Django, a “view” describes which data is presented, but a view normally delegates to a template, which describes how the data is presented. The Views actually hold the HTML pages and the python code. It also has an automatic web admin interface for editing the models.

Django models holds all the data that a web application needs. Models basically describes your data.

To summarize,

Models – Describes your data

Views – Controls what user’s see

Templates – How user’s see it

Controller – URL dispatcher

**References**:

1. <https://en.wikipedia.org/wiki/High-level_architecture>
2. <https://en.wikipedia.org/wiki/Multitier_architecture>
3. <https://djangosites.org/>