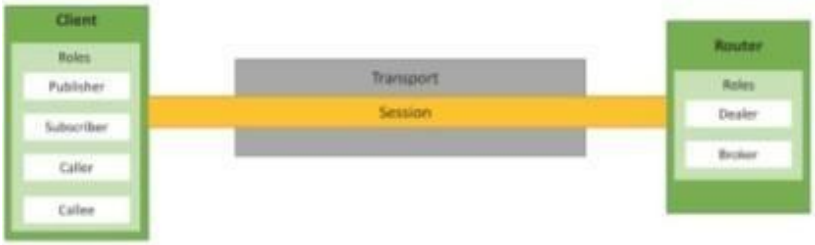
UNIT 4

WAMP Server

Web Application Messaging Protocol (WAMP) is a sub-protocol of Websocket which provides publish-subscribe and remote procedure call (RPC) messaging patterns.



WAMP

1. **Transport:**Transportischannelthatconnectstwo peers.
2. **Session:**Sessionisaconversationbetweentwopeersthatrunsoveratransport.
3. **Client:**Clientsarepeersthatcanhaveoneormoreroles.Inpublish-subscribemodelclient can have following roles:
    a) Publisher:Publisherpublishesevents(includingpayload)tothetopicmaintainedby the broker.
    b) Subscriber:Subscribersubscribestothetopicsandreceivestheeventsincludingthe payload.

InRPCmodelclientcanhavefollowingroles: –

1. **Caller:** Caller issues calls to the remote procedures along with call arguments. – Callee: Callee executes the procedures to which the calls are issued by the caller and returns the results back to the caller. • Router: Routers are peers that perform generic call and event routing. In publish-subscribe model Router has the role of a Broker: – Broker: Broker acts as a router and routes messages published to a topic to all subscribers subscribed to thetopic.

InRPCmodelRouterhastheroleofa Broker: –

1. Dealer:DealeractsarouterandroutesRPCcallsfromtheCallertotheCalleeandroutes results from Callee toCaller.
2. ApplicationCode:ApplicationcoderunsontheClients(Publisher,Subscriber,Calleeor Caller).

AWS

What is AWS EC2?

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers. Amazon EC2's simple web service interface allows you to obtain and configure capacity with minimal friction. It provides you with complete control of your computing resources and lets you run on Amazon's proven computing environment.

Amazon EC2 offers the broadest and deepest compute platform with choice of processor, storage, networking, operating system, and purchase model. We offer the fastest processors in the cloud and we are the only cloud with 400 Gbps ethernet networking. We have the most powerful GPU instances for machine learning training and graphics workloads, as well as the lowest cost-per-inference instances in the cloud.

What is AWS Auto Scaling?

AWS Auto Scaling is an Amazon service that lets you configure automatic scaling of AWS resources. It increases computing power or storage resources available for applications when loads increase, and reduces it when no longer needed.

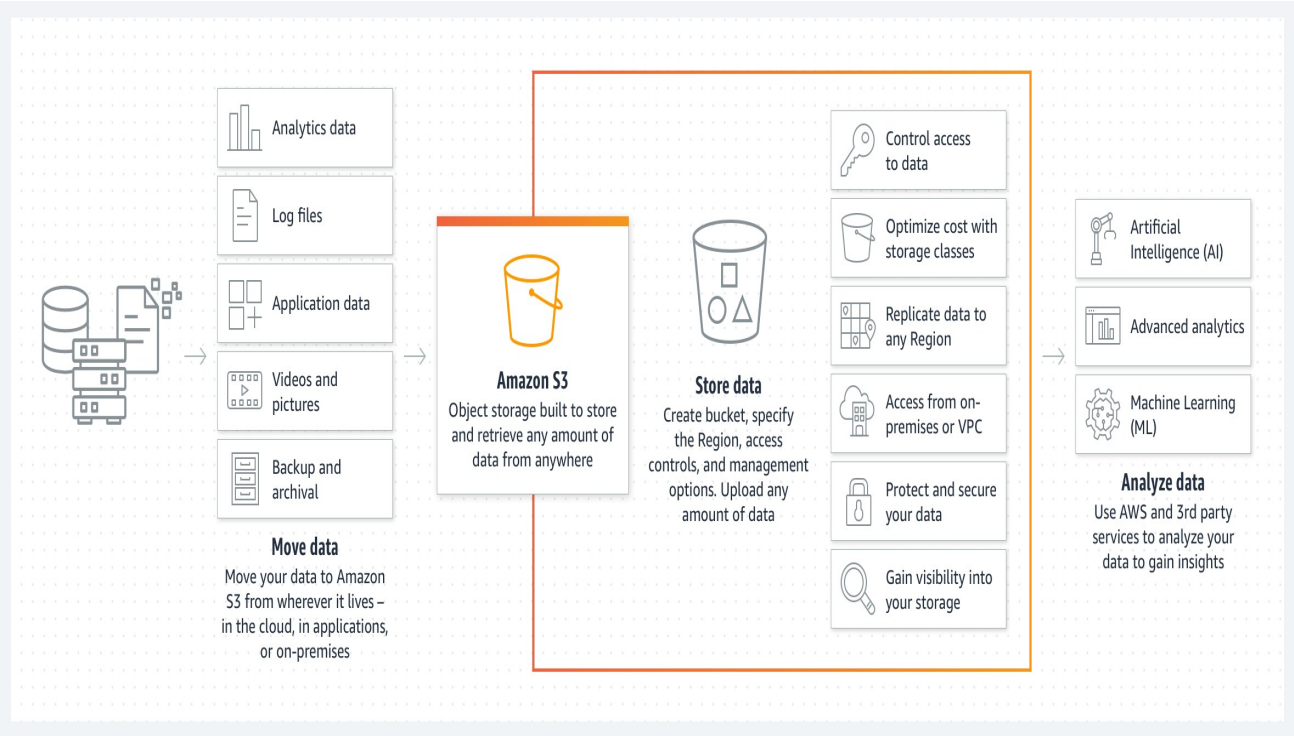
The AWS Auto Scaling Console provides a single user interface to use the auto scaling capabilities of various AWS services. AWS Auto Scaling can be used to scale Amazon Elastic Compute Cloud (EC2), EC2 Spot Fleet requests, Elastic Container Service (ECS), DynamoDB, and Amazon Aurora.

AWS Auto Scaling enables you to configure and manage scalability using **scaling strategies**—define how to optimize resource usage—preferring availability, cost, or a balance of the two. It is also possible to create custom scaling strategies.

You can also leverage **scaling plans**—these are policies that adjust resources using dynamic or predictive scaling.

Amazon S3

Amazon Simple Storage Service (Amazon S3) is an object storage service offering industry-leading scalability, data availability, security, and performance. Customers of all sizes and industries can store and protect any amount of data for virtually any use case, such as data lakes, cloud-native applications, and mobile apps. With cost-effective storage classes and easy-to-use management features, you can optimize costs, organize data, and configure fine-tuned access controls to meet specific business, organizational, and compliance requirements.



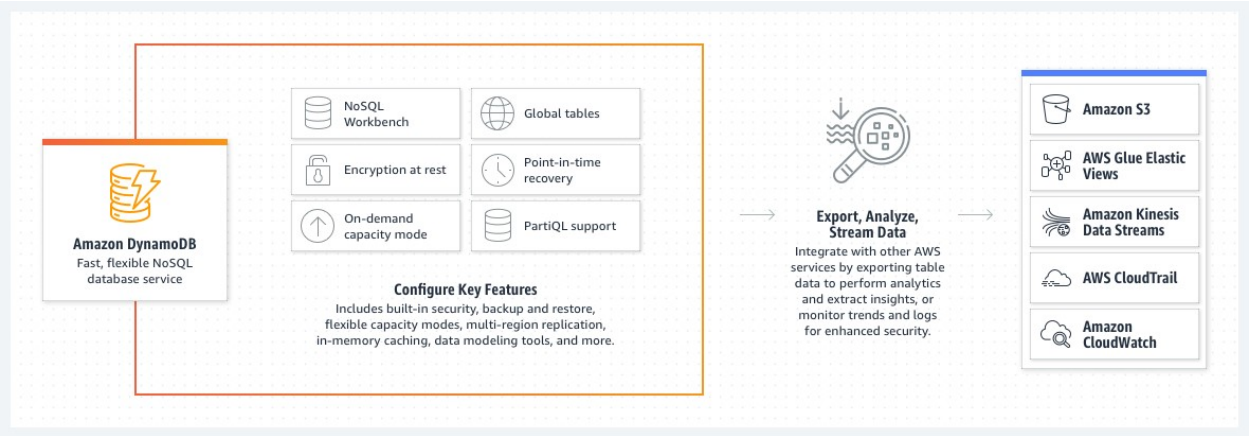
Amazon RDS

Amazon Relational Database Service (Amazon RDS) makes it easy to set up, operate, and scale a relational database in the cloud. It provides cost-efficient and resizable capacity while automating time-consuming administration tasks such as hardware provisioning, database setup, patching and backups. It frees you to focus on your applications so you can give them the fast performance, high availability, security and compatibility they need.

Amazon RDS is available on several database instance types - optimized for memory, performance or I/O - and provides you with six familiar database engines to-choose-from,- including AmazonAurora, PostgreSQL, MySQL, MariaDB, Oracle Database, and SQL Server. You can use the AWS Database Migration Service to easily migrate or replicate your existing databases to Amazon RDS.

Amazon DynamoDB

Amazon DynamoDB is a fully managed, serverless, key-value NoSQL database designed to run high-performance applications at any scale. DynamoDB offers built-in security, continuous backups, automated multi-region replication, in-memory caching, and data export tools.



Amazon Kinesis

Amazon Kinesis makes it easy to collect, process, and analyze real-time, streaming data so you can get timely insights and react quickly to new information. Amazon Kinesis offers key capabilities to cost-effectively process streaming data at any scale, along with the flexibility to choose the tools that best suit the requirements of your application. With Amazon Kinesis, you can ingest real-time data such as video, audio, application logs, website clickstreams, and IoT telemetry data for machine learning, analytics, and other applications. Amazon Kinesis enables you to process and analyze data as it arrives and respond instantly instead of having to wait until all your data is collected before the processing can begin.

SQS

Amazon Simple Queue Service (SQS) is a fully managed message queuing service that enables you to decouple and scale microservices, distributed systems, and serverless applications. SQS eliminates the complexity and overhead associated

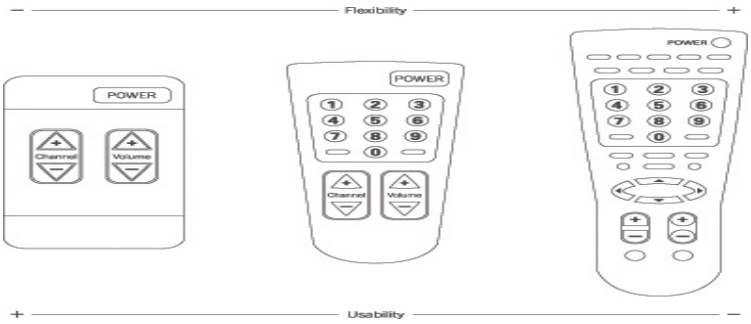
with managing and operating message-oriented middleware, and empowers developers to focus on differentiating work. Using SQS, you can send, store, and receive messages between software components at any volume, without losing messages or requiring other services to be available. Get started with SQS in minutes using the AWS console, Command Line Interface or SDK of your choice, and three simple commands.

SQS offers two types of message queues. Standard queues offer maximum throughput, best-effort ordering, and at-least-once delivery. SQS FIFO queues are designed to guarantee that messages are processed exactly once, in the exact order that they are sent.

IOT Specific Challenges

1) DECIDING ON THE LEVEL OF INTERACTIVITY OF A CONNECTED DEVICE

Adding interactivity to a device is a cost question. Buttons, switches, and the components required add to the bill of materials and increase development costs. It's also a usability question. Designers need to strike a balance between easy-to-use devices and ones with many features and functions. This is sometimes referred to as the "flexibility usability tradeoff" .



However, simple devices with little features have other disadvantages. They often rely on other devices for full-feature interaction. This makes them dependent on things outside of the designer's control, such as a working Internet connection or a compatible browser. Imagine you couldn't change the temperature in your house because of problems with your Internet service provider.

An example of this approach was the first Tado thermostat. Users controlled it using a website and smartphone app. But the product offered a minimal degree of control on the device, too. A single physical button let users switch the heating on or off as a fallback option.

Tado has since introduced a new generation that features capacitive touch buttons and a display to adjust temperature up and down right on the device

Only having few controls on a device can be advantageous if you know your product will change and develop. In this case, supporting a physical interface can become a burden when system features are changed or abandoned.

There are ways to balance the benefits and drawbacks of physical controls. An example is Cloudwash, BERG's connected washing machine prototype (see Figure 8-33). The physical device has some controls that only cover selected functions. Full control is delivered through a smartphone app. But users can perform the everyday tasks without needing the smartphone app. Also, the device uses a combination of physical controls with displays.

2) MOBILE AND WEB UIS

A lot of connected products use mobile apps or web interfaces as their primary means of control or split the interaction between them and the device itself. There are good reasons to do this:

- App and web interfaces are much easier to change and develop in response to changing or developing requirements. If you add or remove a feature, updating or rethinking the UI of an app or a web interface accordingly is less of a problem than when you have a fixed interface on an embedded device.
- App and web development have a large, active developer community with many useful resources easily available. There is a lot of documentation for best practice solutions and even libraries developers can use to solve particular problems as opposed to developing from scratch.
- Smartphones and personal computers provide ample processing power to allow for features an embedded device might not be capable of (like speech recognition or using a camera). High-resolution color screens and rich input capabilities make it easier to create interfaces that move between simple and complex when necessary. (For example, allowing text input when necessary, but not bloating the interface with a keyboard for most of the time.)
- Users carry smartphones with them, so user location can become a trigger or data point used by the product. Because a smartphone is a personal device, interfaces can be tailored to a particular user.
- App stores handle the distribution of software updates so developers don't have to.
- App platforms offer notification channels that are irrespective of the user's location or that of the connected product. This means a connected product can notify a user even if they aren't near it.

But there are also reasons why you might avoid relying on a mobile phone or web UIs:

- Smartphones aren't reliably available all the time: users might switch them off or turn them to silent mode, or the devices might run out of battery or lose a signal. This all makes smartphones less appropriate for critical notifications that need to be received immediately.
- Web UIs that require a personal computer, but also using a smartphone can be slower than on-device controls in the right context. For example, some users of connected locks complain that the process of getting out their smartphone and launching the right app takes them longer than getting out their keys and unlocking a door.
- When functions or alarms need to be available to anyone in certain locations, whether they have a smartphone or not, on-device interfaces are necessary.
- Less tech-savvy users can have difficulty building accurate mental models of how a system works. For example, during the research for BBC's Radiodan project, participants connecting to the radio's setup interface using their web browser were unsure if the radio would still work when they closed the browser window.

3) GLANCEABLE AND AMBIENT INTERFACES

We've already touched on how designers can create interactions that require less attention. Considering the ever-increasing amount of information around us, this will become ever more important. We'll soon all crave a less distracting environment.

This kind of interaction is already around us. Consider how a wall clock "disappears" into our surrounding. We completely ignore it for the most part. The moment we're interested in the time, we can get it at a glance. Even from across the room. We might only be able to see roughly the angle of the two main hands—but it's enough to give us an idea of what time it is. If we want more detail, we can walk closer to see exactly what time it is.

The ambient devices mentioned earlier are attempts to achieve similar characteristics. That can mean devices that only grab the user's attention when required, like the Ambient Umbrella, or devices that convey information in a way similar to how you sense what time of day it is by seeing, in the corner of your eye, how bright it is outside through a nearby window

Pre-attentive

Glanceable; no cognitive load required

Calm

Nonintrusive; seamless with environment (e.g., Evergreen, Friendly)

Universal

No language, characters, or numbers

Open

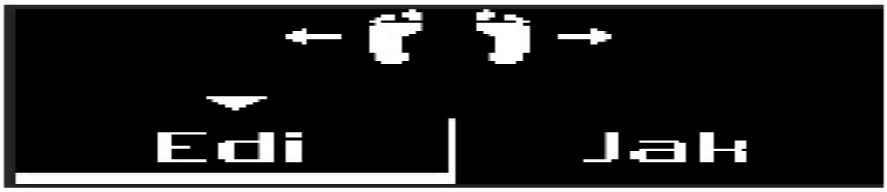
Able to represent multiple types of data; coded, private

4) WORKING WITH LIMITED INPUT AND OUTPUT CAPABILITIES

A particular challenge in connected devices is to work around limited input and output capabilities. Many connected devices only require complex input rarely and a simple interface is sufficient most of the time. So how can designers handle these rare, but complex interactions without having to bloat the interface with capabilities that aren't needed most of the time?

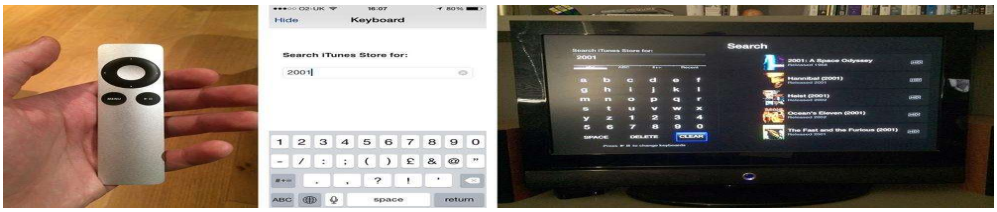
The Withings Smart Body Analyzer provides a great example. The smart body scale supports multiple users, each with their own accounts. It automatically identifies which user is standing on it by comparing the reading with its historic data. If the new reading is within a realistic extrapolation of only one account, it can only be that person.

This doesn't always work. When a reading is within range of more than one account, the device cannot reliably decide which user it is. In this case, the device needs user input. But instead of having specific controls for this situation, the designers found a solution that keeps the interface simple. They work with the capabilities available. In this case, the display shows a selection between the likely accounts. Users then lift either their left or the right foot to make a selection (see Figure 8-34).



Another strategy is to support devices with more capabilities. Navigating and controlling Apple TV with the minimalistic remote is sufficient for most of the time. When it comes to entering search terms or login credentials, using the remote becomes tedious. By supporting an iPhone as an alternative input device, users can switch to the more capable device when required (see Figure 8-35).

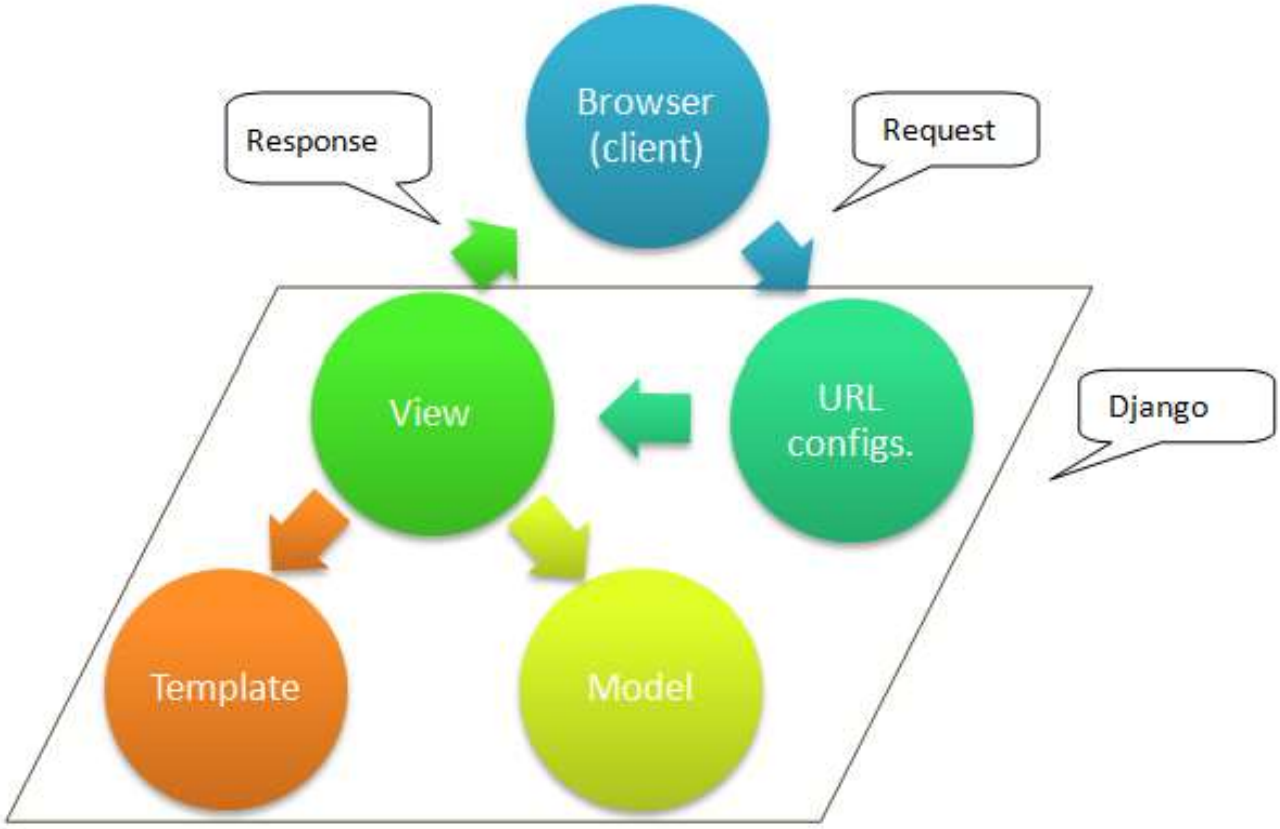
The Amazon Fire TV remote (see Figure 8-36) uses speech recognition to overcome this same hurdle. Its remote has a dedicated button for voice search and users can speak a search term into the remote rather than having to type. That way, the interface stays simple even though complex input is possible.



Django

Djangoispronounced JANG-oh.The“D”issilent.

“Djangoisahigh-levelPythonwebframeworkthat encourages rapid development and clean, pragmaticdesign.”



Django is based on MVT (Model-View-Template) architecture. MVT is a software design pattern for developing a web application.

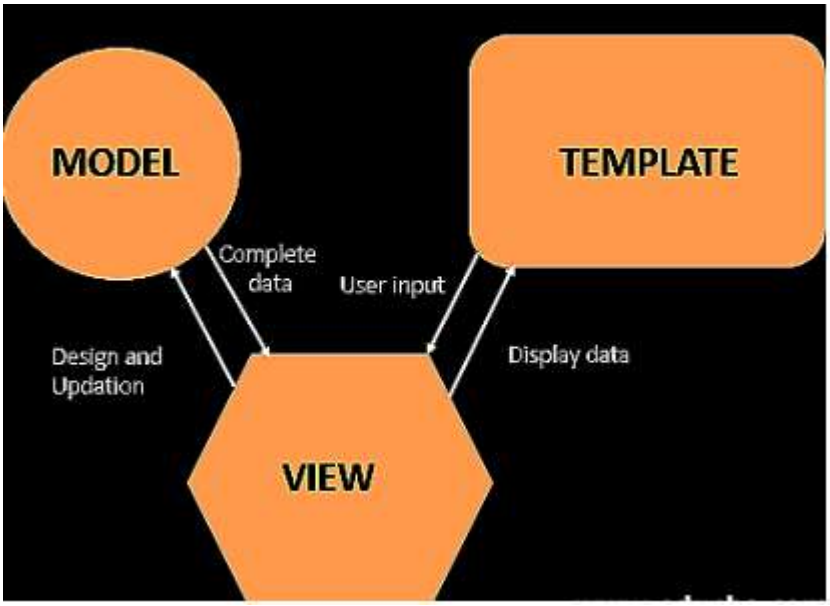
MVT Structure has the following three parts –

Model: The model is going to act as the interface of your data. It is responsible for maintaining data. It is the logical data structure behind the entire application and is represented by a database (generally relational databases such as MySql, Postgres).

View: The View is the user interface — what you see in your browser when you render a website. It is represented by HTML/CSS/Javascript and Jinja files.

Template: A template consists of static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted.

Architecture:



Project Structure:

A Django Project when initialized contains basic files by default such as manage.py, view.py, etc. A simple project structure is enough to create a single-page application. Here are the major files and their explanations. Inside the prp_site folder (project folder) there will be the following files-

manage.py- This file is used to interact with your project via the command line(start the server, sync the database… etc). For getting the full list of commands that can be executed by manage.py type this code in the command window-

folder (prp_site) – This folder contains all the packages of your project. Initially, it contains four files –

init.py – It is a python package. It is invoked when the package or a module in the package is imported. We usually use this to execute package initialization code, for example for the initialization of package-level data.

settings.py – As the name indicates it contains all the website settings. In this file, we register any applications we create, the location of our static files, database configuration details, etc.

urls.py – In this file, we store all links of the project and functions to call.

wsgi.py – This file is used in deploying the project in WSGI. It is used to help your Django application communicate with the webserver.

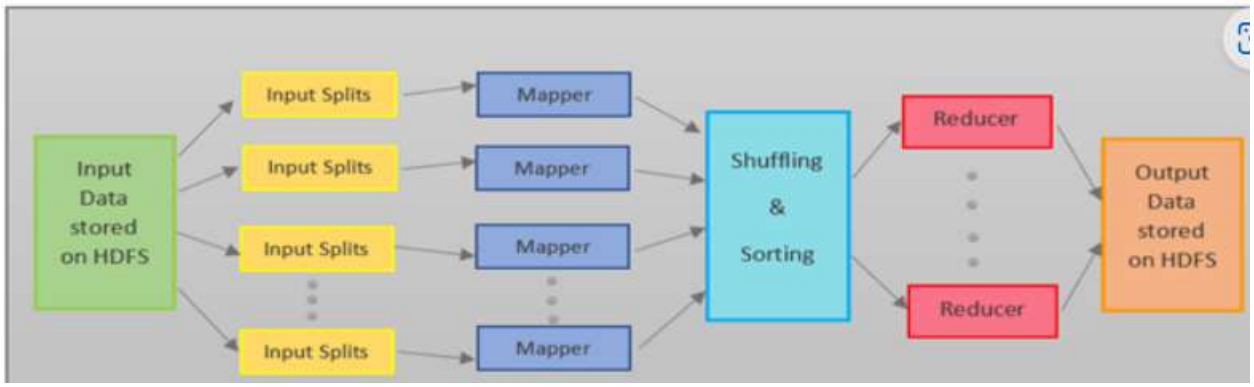
MAP Reduce Problem

MapReduce is a programming model and framework within the Hadoop ecosystem that enables efficient processing of big data by automatically distributing and parallelizing the computation. It consists of two fundamental tasks: Map and Reduce.

In the Map phase, the input data is divided into smaller chunks and processed independently in parallel across multiple nodes in a distributed computing environment. Each chunk is transformed or "mapped" into key-value pairs by applying a user-defined function. The output of the Map phase is a set of intermediate key-value pairs.

The Reduce phase follows the Map phase. It gathers the intermediate key-value pairs generated by the Map tasks, performs data shuffling to group together pairs with the same key, and then applies a user-defined reduction function to aggregate and process the data. The output of the Reduce phase is the final result of the computation**.**

MapReduce Architecture:



Map Reduce example process has the following phases:

Input Splits
Mapping
Shuffling
Sorting
Reducing