ABSTRACT

The entitled project “**INVENTORY AND SALES DATABASE MANAGEMENT SYSTEM**” is made keeping in mind all the aspects of the Inventory and sales. By all the aspects I mean, it will be capable of doing all the necessary operations/functions that are done in any Inventory and sales for example- Purchasing of a goods, Selling of a Goods ,keeping track of Inventory incoming and outgoing goods in detail,billing details etc.

Since all the work that is to be done by this software can also be done manually, but this consumes time and labour. So, this software will be a relief to those who have to do all this work manually. The knowledge of computers and programming has become a basic skill needed to survive in present information based on society.

The motive to make this project is to make such kind of software which is very easy to use. There will not be need of any training and the person who does not have much knowledge of computers can also use this. Through this project the details of the customers that have purchased the goods can be retrieved if necessary. All the records of the goods will be kept for further enquiries.

CONTENTS

Chapter 1

INTRODUCTION

1.1 Project Outline

This application is going to be used by:

1. Accountant/Biller
2. Business Dealer

This application is meant for users (Biller and Business Dealer) who can access the information from database about inventory goods. Our objective is so that executives and administrators can easily automate their work with less effort. For example, they can generate quotations for any enquiry for a goods in no time.
They can also have the customer's feedback to the show clear authority regarding flow of good in their business.

1.2 Project Objective

The objective of an inventory management project is to ensure that a business has the right amount of stock on hand to meet customer demand, while also minimizing excess inventory and associated costs.

This can include implementing systems and processes for tracking inventory levels, setting reorder points, forecasting demand, and managing inventory turnover.

The ultimate goal is to achieve a balance between having enough inventory to meet customer needs and not having so much that it ties up valuable resources such as cash and storage space.

1.3 Project Purpose

The System should be capable of performing the following:

- Stock control: This involves maintaining accurate records of the items in stock, including the quantities, locations, and costs of each item.

- Purchasing and procurement: This involves managing the process of purchasing and receiving new items, from placing orders to receiving and inspecting goods.

- System should be able to generate invoices and maintain transaction details.

- Data analysis and reporting: Inventory management systems often include tools for analyzing data and generating reports, such as inventory reports, sales reports, and purchase reports.

- Inventory optimization: This involves finding the optimal inventory level for a product to minimize holding costs and stockouts.

- Sales and customer service: Inventory management also includes monitoring and managing sales and customer demand, as well as ensuring that the right items are available to meet customer needs.

- Help the organization to maintain customer details like customer name, transaction/purchase details, goods purchased, purchase medium, purchase date etc.

- Distribution and logistics: This involve managing the movement of goods from the warehouse to the customer, including shipping, receiving and handling.
- Inventory forecasting: This involves predicting the future demand of the product and plan the inventory accordingly.

Chapter 2

SYSTEM SPECIFICATIONS

2.1 Hardware Requirements

Minimum Requirements:

- Processor: Intel Pentium 4/ AMD Athlon series
- RAM: 512MB
- Storage: 100MB

Recommended Requirements:

- Processor: Intel 11th generation I-series/AMD Ryzen Threadripper series
- RAM: 8GB
- Storage: 500GB

2.2 Software Requirements

OS:Windows,GNU/Linux Distributions, Mac OS, BSD, 64-bit

2.3 Tech Stack

Frontend

- **HTML 5:**
  HTML5 is a markup language used for structuring and presenting content on the World Wide Web. It is the fifth and last major HTML version that is a World Wide Web Consortium recommendation. The current specification is known as the HTML Living Standard.

- **CSS 3:**
  CSS3 is the latest version of the CSS specification. CSS3 adds several new styling features and improvements to enhance the web presentation capabilities. Note: Our CSS tutorial will help you to learn the fundamentals of the latest CSS3 language, from the basic to advanced topics step-by-step.

- **JavaScript:**
  JavaScript, often abbreviated JS, is a programming language that is one of the core technologies of the World Wide Web, alongside HTML and CSS. Over 97% of websites use JavaScript on the client side for web page behavior, often incorporating third-party libraries.

Backend

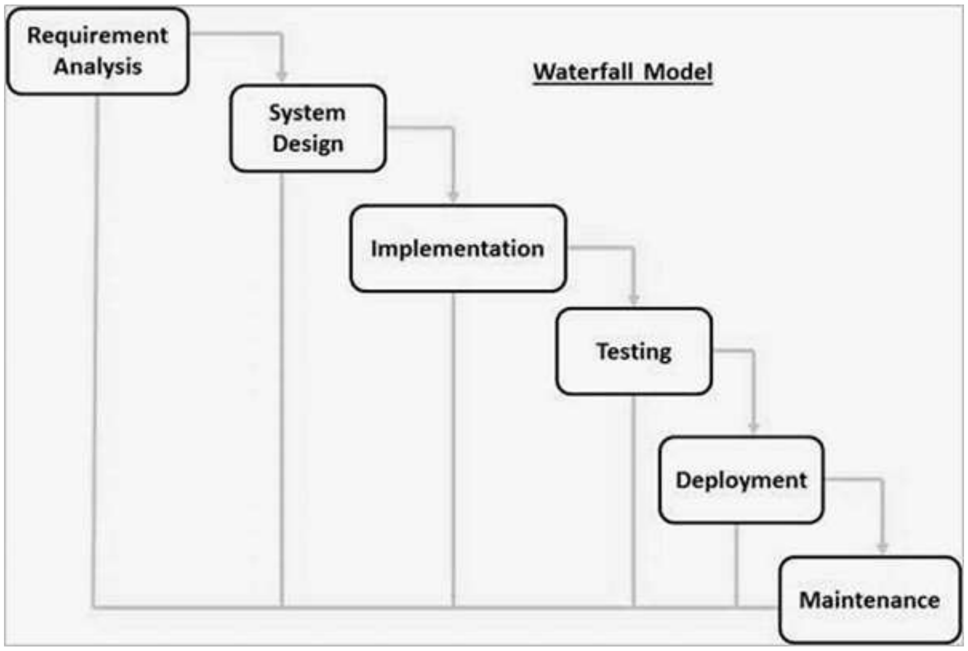
- **Django:**
  Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It’s free and open source.

- **SQLite3:**
- SQLite is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine. SQLite is the most used database engine in the world. SQLite is built into all mobile phones and most computers and comes bundled inside countless other applications that people use every day.

Chapter 3

DESIGN

3.1 Development model



- **Requirement Analysis:**
  All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.

- **Design stage:**
  The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.

- **Implementation:**
  With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.

- **Integration and Testing:**
  All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.

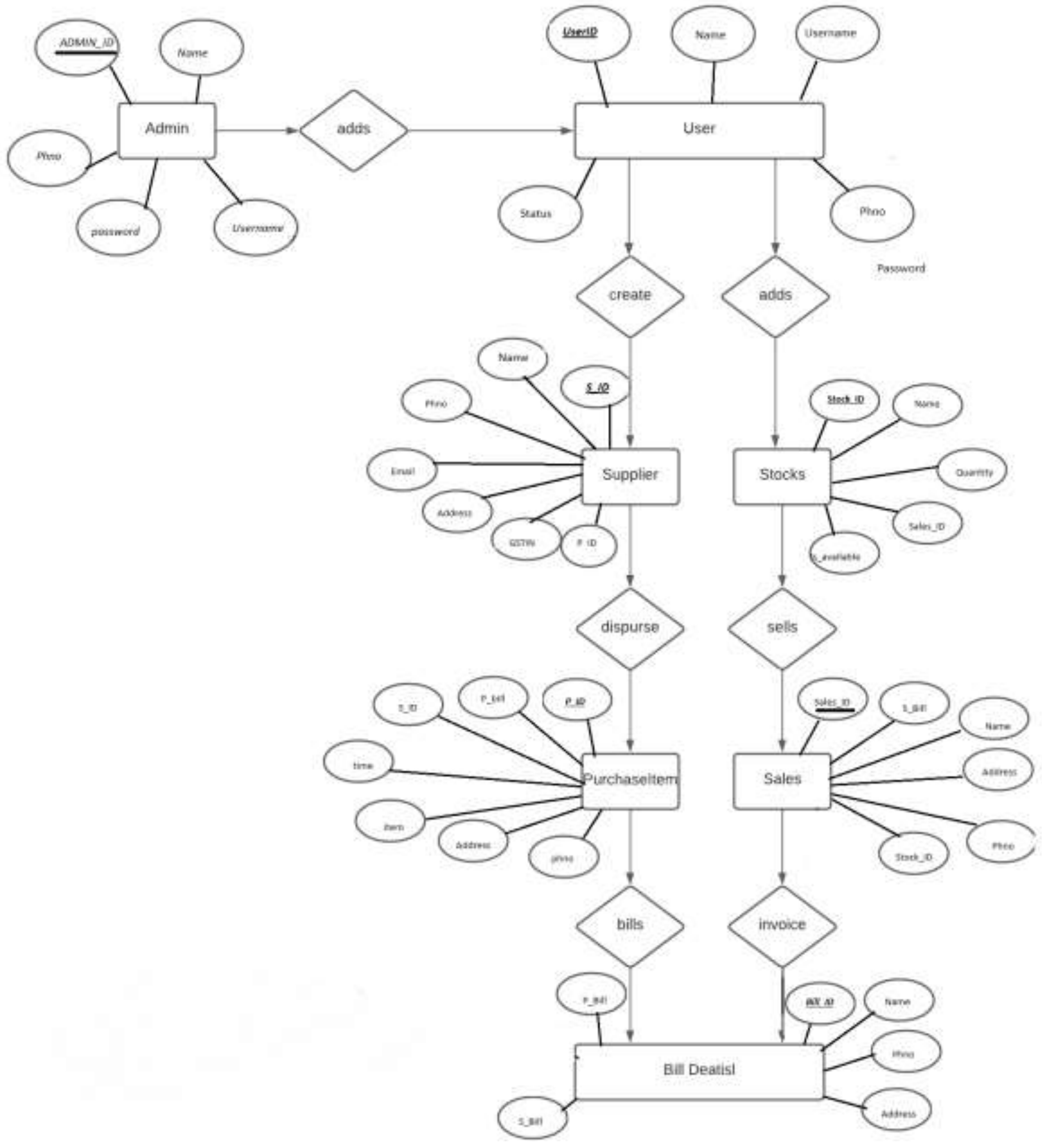
- **Deployment of System:**
  Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.

- **Maintenance:**
  There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

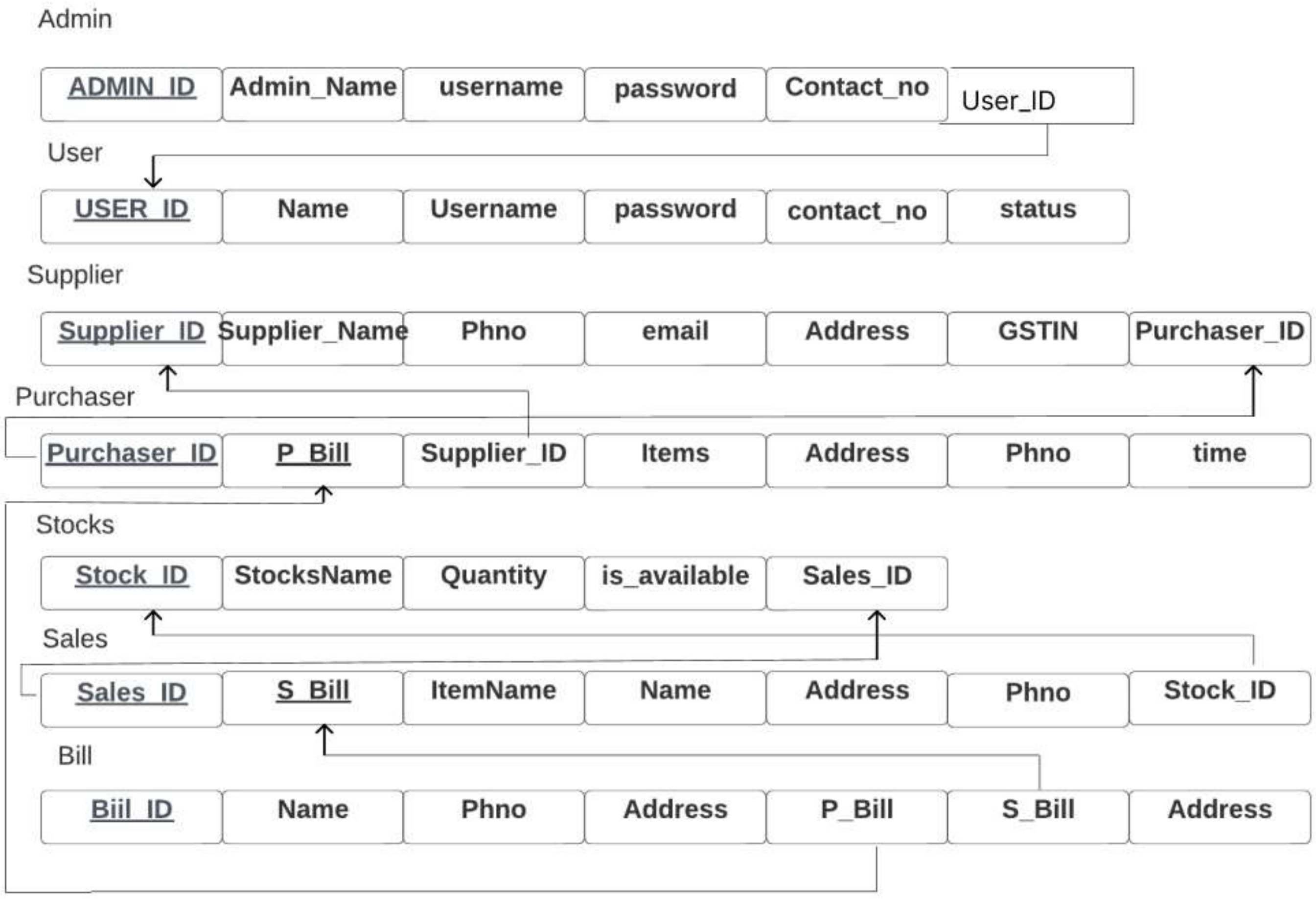
All these phases are cascaded to each other in which progress is seen as flowing steadily downwards , (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for previous phase and it is signed off, so the name "Waterfall Model". In this model, phases do not overlap.

3.2 Database Description

ER-Diagram



Schema Diagram



Chapter 4

IMPLEMENTATION AND CODING

4.1 Backend Models(Tables)

User Model

Contains the User table definition and attribute list, based and derived from the schema diagram
# src/models.py
class User(models.Model) => (attrs)
The User Table hereby referred to as the "User Object" further into this sectionof the documentation consists of the following attributes

1.) user_id :

Attribute definition:

(variable) user_id: IntegerField => (
primary_key: bool,
unique: bool
)

Source code:

user_id = models.IntegerField(
primary_key=True,
unique=True
)

Refers to the identification number of the user, **primary_key=True** parameter is setting the attribute to a primary key. The **unique=True** parameter is useful in referencing this particular attribute via ***Foreign Keys***.

2.)Name of User:

Attribute definition:

(variable) user_name: CharField =>(
max_length: int,
blank: bool, null: bool
)

Source code:

user_name = models.CharField(
max_length=100,
blank=False,
null=False
)

Refers to the dealer's name field the **NameField** method stores the name of the dealer class, **null=False** parameter dealer class,null=Falseparameter setting to the null attribute.max_length is setto (100 and it's restricts the number of characters used by the user by (100. **Blank=False**

3.) username

Attribute definition:

(variable) username: ForeignKey => (
(class) User,
(parameter) on_delete: (...) -> None,
(parameter) related_name: str | None
)

Source code:

username = models.ForeignKey(
User, on_delete=models.CASCADE,
related_name='d_username'
)

refers to the username of the user, as apparent here this attribute is a foreignkey that refers to **(class) User** , it has a parameter **(parameter) on_delete: (...) -> None** set to **models.CASCADE** which means on deletion of a record from
this table it reflects by comitting deletion on the referenced table as well. The **(parameter) related_name: str | None** parameter sets up an alternate name or view to avoid arising naming conflicts.

4.) last_login:

Attribute definition:

(variable) last_login: DateTimeField => (
(parameter) auto_now: bool
)

Source code:

last_login = models.DateTimeField(auto_now=True)

The **last_login** attribute refers to the last time the user had logged in. the **(parameter) auto_now: bool** is set to **True**, the **auto_now** parameter automatically sets the login time based on server timezone.

5.) contact:

<u>Attribute definition:</u>

(variable) contact: IntegerField => (
null: bool,
blank: bool
)

<u>Source code:</u>

contact = models.IntegerField(
null=False,
blank=False
)

refers to the contact details of the user class.the **(parameter)null: bool** is set to **False, Blank=False**

6.) Created:

<u>Attribute definition:</u>

(variable) created: DateTimeField => (
(parameter) auto_created: bool
)

<u>Source code:</u>

created = models.DateTimeField(
auto_created=True
)

The **created** attribute refers to the date the user profile was created, **(parameter) auto_created: bool**   is set to **True**, this is a method passed in as a parameter which sets the value to current date and time based on the server region and timezone,

Stocks Model

Contains the Stocks table definition and attribute list, based and derived from the schema diagram

1.) stock_id:

Attribute definition:

(variable) stock_id: IntegerField => (
(parameter) primary_key: bool,
(parameter) unique: bool
)

Source code:

```
stock_id  =  models.IntegerField(
primary_key=True,
unique=True
)
```

The **stock_id** attribute refers to the numeric id of the vehicle entries present in the table, it is a **primary key** and it is set to be **unique**.

2.) sales_id:

Attribute definition:

(variable) sales_id: ForeignKey =>(
(class) Sales,
(parameter) on_delete: (...) -> None,
(parameter) to_field: str | None
)

Source code:

```
stock_id = models.ForeignKey(
Dealer,
to_field=sales_id',
on_delete=models.CASCADE
)
```

refers to the id of the user, as apparent here this attribute is a foreign keythat refers to **(class) sales**, it has a parameter **(parameter) on delete: (…) -> None** set to **models.CASCADE** which means on deletion of a record from this table it reflects by comitting deletion on the referenced table as well. The **to field** is set to **sales id**

3.) Quantity_img:

Attribute definition:

(variable)Quantity: Integer => (
blank: Any
)

Source code:

Quantity = models.Integer(
blank=False
)

The **Quantity** attribute contains the quantity uploaded to the server stored at **/media/** folder in the **MEDIA DIR** directory of the server, **blank** is set to **False** so an image is required for every record uploaded.

4.) name:

Attribute definition:

(variable) name: CharField => (
(parameter) max_length: int | None,
(parameter) blank: bool
)

Source code:

name = models.CharField(
max_length=100,
blank=False
)

The **name** attribute refers to the vehicle name, it's **max_length** is set to **100** and **blank** is **False** so it cannot be left empty.

5.) is_available:

Attribute definition:

(variable) is_available: CharField =>(
(parameter) max_length: int | None,
(parameter) choices: _FieldChoices | None,
(parameter) blank: bool
)

Source code:

Is_avialable = models.CharField
(max_length=100,
choices=False
)

Sales Model

Contains the Customer table definition and attribute list, based and derived from the schema diagram

1.) Sales_id:

Attribute definition:

(variable) Sales_id: IntegerField => (
(parameter) primary_key: bool,
(parameter) unique: bool
)

Source code:

Sales_id = models.IntegerField(
primary_key=True,
unique=True
)

The **Sales_id** attribute refers to numeric customer id, it is a **primary key**, and it is **unique**.

2.) email:

Attribute definition:

(variable) email: CharField => (
(parameter) max_length: int | None,
(parameter) blank: bool
)

Source code:

email = models.CharField(

max_length=100,

blank=False

)

The **email** attribute refers to the customer's email.it's **max_length** is set to **100** and it cannot be left **blank**.

3.) Contact:

<u>Attribute definition:</u>

(variable) contact: IntegerField =>(

(parameter) blank: bool

)

<u>Source code:</u>

contact = models.IntegerField(

blank=False

)

The **contact** attribute refers to the contact information of the customer, it cannot be left **blank**.

4.) Sales_Bill:

<u>Attribute definition:</u>

(variable) Sales_Bill: CharField => (

(parameter) max_length: int | None,

(parameter) blank: bool

)

The **Sales_Bill** attribute refers to the customer's Bill name in which he/she resides. It's **max_length** is set to **100** and it cannot be left **blank**.

9.) Address:

<u>Attribute definition:</u>

(variable) address: TextField => (

(parameter) max_length: int | None

)

Source code:

address = models.TextField(

max_length=100,

blank=False

)

The **address** attribute refers to the customer’s address, it cannot be left **blank**.

Supplier Model

Contains the Sale table definition and attribute list, based and derived from the schema diagram

1.) Supplier_id:

Attribute definition:

variable) supplier_id: ForeignKey => (
(class) Customer,
(parameter) to_field: str | None,
(parameter) on_delete: (...) -> None
)

Source code:

supplier_id = models.ForeignKey(
Customer,
to_field='customer_id',
on_delete=models.CASCADE
)

The **customer_id** attribute refers to the numeric id of the referenced customer.

2.) purchase_id:

Attribute definition:

(variable) purchase_id: IntegerField =>(
(parameter) primary_key: bool,
(parameter) unique: bool
)

Source code:

```
purchase_id = models.IntegerField(
primary_key=True,
unique=True
)
```

The **sale_id** attribute refers to the numeric id of a sale made.

3.) Phno:

Attribute definition:

```
(variable)phno: =>(
(parameter) to_field: number | None,
(parameter) on_delete: (...) -> None
)
```

Source code:

```
phno = models.integer (
to_field=phone,
on_delete=models.CASCADE
)
```

The phno attribute refers to the numeric id of the referenced vehicle object.

4.) Description:

Attribute definition:

5.) order_date:

Attribute definition:

```
(variable) order_date: DateTimeField =>(
(variable) order_date: DateTimeField
)
```

Source code:

```
order_date  =  models.DateTimeField(
auto_created=True
```

)

The **order_date** attribute refers to the date the sale was made.

6.) GSTIN:

<u>Attribute definition:</u>

(variable) GSTIN: varcharField => (
(parameter) blank: bool
)

<u>Source code:</u>

GSTIN = models.varcharField(
blank=False
)

The **cost** refers to the cost paid by the customer to the dealership.

7.) deal_date:

<u>Attribute definition:</u>

8.) Status:

<u>Attribute definition:</u>

(variable) status: CharField => (
(parameter) max_length: int | None,
(parameter) blank: bool,
(parameter) choices: _FieldChoices | None
)

Billing Model

Contains the Billing table definition and attribute list, based and derived from the schema diagram

1.) biil_id:

Attribute definition:

```
(variable) bill_id: PrimaryKey => (
(class) Sale,
(parameter) to_field: str | None,
(parameter) on_delete: (...) -> None
)
```

Source code:

```
bill_id = models.PrimaryKey(
Sale,
to_field=bill_id',
on_delete=models.CASCADE
)
```

The bill_id attribute refers to the numeric id referenced in the Sale Model/Table.

2.) P_Billno:

Attribute definition:

```
(variable) P_Billno: IntegerField =>(
(variable) P_Billno: IntegerField
)
```

Source code:

```
P_Billno = models.IntegerField(
blank=False
)
```

The **tax** attribute refers to the amount of tax applied to the bill of purchase.

3.) S_Billno:

Attribute definition:

```
(variable) S_Billno: IntegerField =>(
(variable) S_Billno: IntegerField
)
```

Source code:

```
S_Billno = models.IntegerField(
blank=False
)
```

The **tax** attribute refers to the amount of tax applied to the bill of sales.

4.) status:

Attribute definition:

```
(variable) status: CharField => (
(parameter) max_length: int | None,
(parameter) blank: bool,
(parameter) choices: _FieldChoices | None
)
```

Chapter 6

CONCLUSION

The project is the perfect system for the inventory and sales for business and company to manage their data and help their customers to buy and stock up item without any problem. It makes the process of buying a item very simpler.

In future, all the updates about the inventory will be centralized because of the application being online.

Provision for customer to access our system will be available.

Users will be able to visit our database being at their convience due to the system being online.

REFERENCES

- www.Google.com
- https://www.djangoproject.com/
- www.w3schools.com
- https://www.sqlite.org/docs.html