ERP Code Review Document

Reviewer : ANAND RAJ B
Date : 19-09-2023
Branch : master

1 . INTRODUCTION

The Objective of this document to ensure the changes that are introduced to the current codebase aligns the code standards , functional requirements , system design and security best practices .

1.1 Scope of review

In this section , we'll define the scope of code review document and the area of coverage are as follows ,

- **Formatting**: Ensuring guidelines and standards.
- **Functionality:** Confirming that the code accomplishes its task and functional requirement.
- **Error Handling:** Checking for better error handling.
- **Modularity and Reusability:** Assessing code structure and potential for reuse and possibility for refactoring the codebase.
- **Security :** Checking for any exposure to sensitive information.
- **Performance:** Identifying performance bottlenecks .

1.2 Review and Feedback process

The code review process involves under **OWASP guidelines** and includes a SANDI model for code reviewing and **CEDAR** model for feedback and improvements .

2 . CODE FORMATTING AND DOCUMENTING

2.1 Code Style and Formatting

Maintaining standard code style and code formatter for better readability of code , some of the them as follows ,

- Use **PEP8** or **black** formatter for better code style and formatting
- REF : Link_1 , Link_2

2.2 Naming conventions

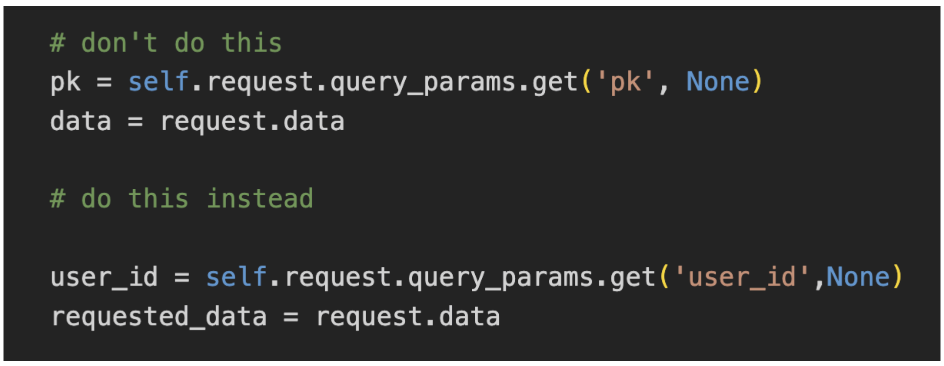
Some of the naming errors in the code base are mentioned as follows ,

- Usage of **same variable name** in all management folder

  E.g sales_name , purchase_order

- It is recommended to follow variable name as follows ,
  - for objects : *Sales_name_obj*
  - for querysets : *Purchase_order_qryst*

- Give valid names for variables .



```python
# don't do this
pk = self.request.query_params.get('pk', None)
data = request.data

# do this instead

user_id = self.request.query_params.get('user_id',None)
requested_data = request.data
```

- Usage of **singular** and **plural** form of variable naming would be a best practice

- for one or more - *Sales_order_items*
- for one item - *Product_name*

- For Constants recommended to use **Uppercase** as standard
  - E.g., CURRENT_SALES

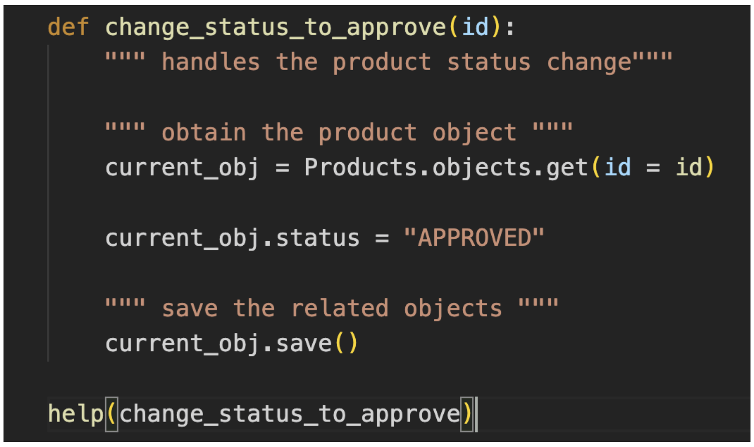
- For File naming use **snack case** as standard
  - E.g., api_permission.py

- Use **Pascal case** for utility classes
  - E.g., MakeSalesReservation() , ProductManagement()

- Use plural for **folder naming** like users/ , helpers/ , forms/

2.3 Docstring and Comments

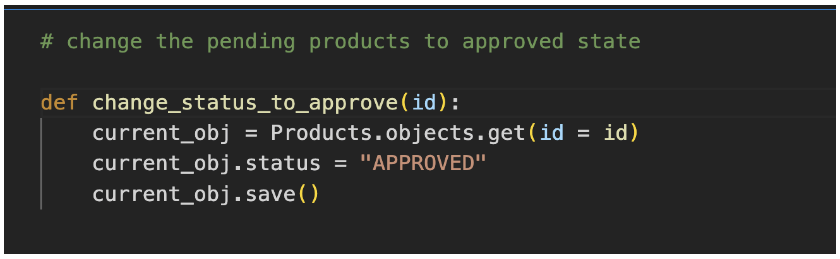
- It is recommended to add **docstring** in the current code in-order to understand the functional works of a piece of code with example as below image ,



```
def change_status_to_approve(id):
    """ handles the product status change"""

    """ obtain the product object """
    current_obj = Products.objects.get(id = id)

    current_obj.status = "APPROVED"

    """ save the related objects """
    current_obj.save()

help(change_status_to_approve)
```

- No docstring required if a function is **straightforward** and what it does .

- Add comments for explaining **why** and **what** of the code



```python
# change the pending products to approved state

def change_status_to_approve(id):
    current_obj = Products.objects.get(id = id)
    current_obj.status = "APPROVED"
    current_obj.save()
```

2.4 Documenting strategies

- Add a detailed **README.md** for
  - Installation guide
  - Technology stack
  - ER diagram
  - Authors , Maintainers and Contributors .

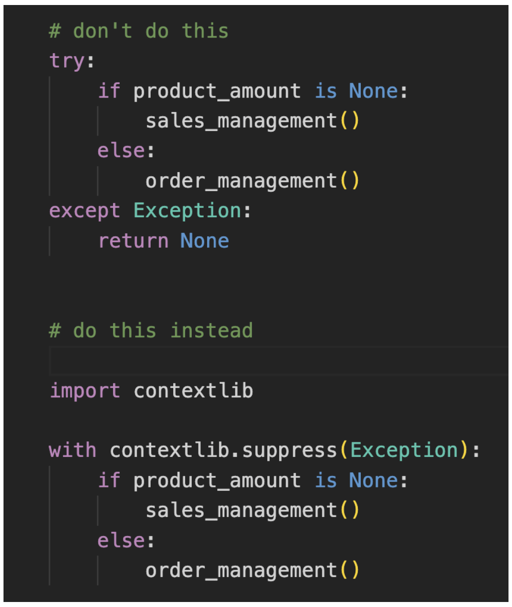
- Additional documents
  - Postman collection
  - Swagger api document
  - Developer documentation
  - Mermaid

- REF : https://readme.so/editor

3 . ERROR HANDLING

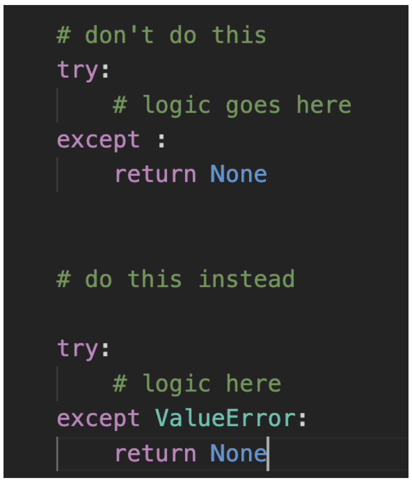
3.1 Exceptions

- As a result there is an event of adding try and except clauses in the current code base , it is better to avoid those exception clauses for smaller logic and use **contextlib** library .



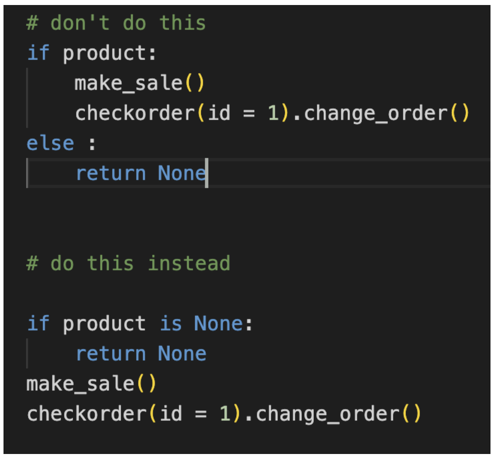
```
# don't do this
try:
    if product_amount is None:
        sales_management()
    else:
        order_management()
except Exception:
    return None


# do this instead

import contextlib

with contextlib.suppress(Exception):
    if product_amount is None:
        sales_management()
    else:
        order_management()
```

- Use exception types in exception clauses like **ValueError** , **TypeError** ,



```
# don't do this
try:
    # logic goes here
except :
    return None


# do this instead

try:
    # logic here
except ValueError:
    return None
```

- For improved readability use **guard classes** for safe exiting .

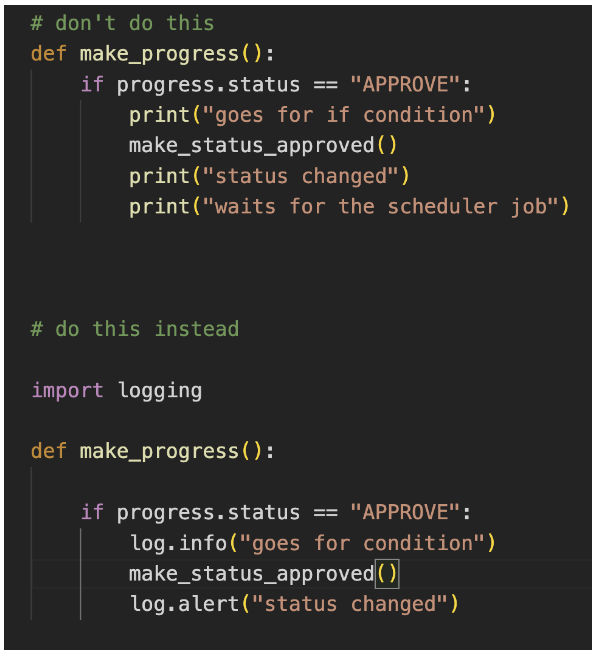


```
# don't do this
if product:
    make_sale()
    checkorder(id = 1).change_order()
else :
    return None


# do this instead

if product is None:
    return None
make_sale()
checkorder(id = 1).change_order()
```

- REF : Link_1 , Link_2

3.2 Logging

- It is recommended to use a logger for printing the process status for better logging and debugging .



```
# don't do this
def make_progress():
    if progress.status == "APPROVE":
        print("goes for if condition")
        make_status_approved()
        print("status changed")
        print("waits for the scheduler job")



# do this instead

import logging

def make_progress():

    if progress.status == "APPROVE":
        log.info("goes for condition")
        make_status_approved()
        log.alert("status changed")
```

- For event based or daily logging use django's inbuilt logger
    - REF : django_logging

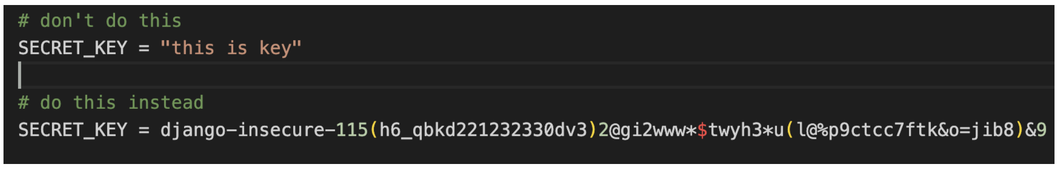
4 . SECURITY IMPROVEMENTS

4.1 Validation

- As a result in the code review process , i could see there is vital loophole in the form validations and **input sanitization** in both client and server side
- **Expected Validations** are well used in the current code base on the server side but requires more fine tuning .
- Chances for **XSS attacks** from client side .
- Tips :
  - Add Content security Policy (**CSP**)
  - Better Input validation and HTML escaping
  - REF : [aviod_xss](#) , [django_security](#)

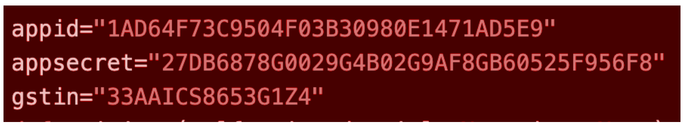
4.2 Credentials Exposure

- **Security Key** in django is the doorway and crucial component for application security , i could see the security key is kept as below.



```
# don't do this
SECRET_KEY = "this is key"

# do this instead
SECRET_KEY = django-insecure-115(h6_qbkd221232330dv3)2@gi2www*$twyh3*u(l@%p9ctcc7ftk&o=jib8)&9
```

- Exposure of sensitive information and credentials are exposed in folder name **gst/**

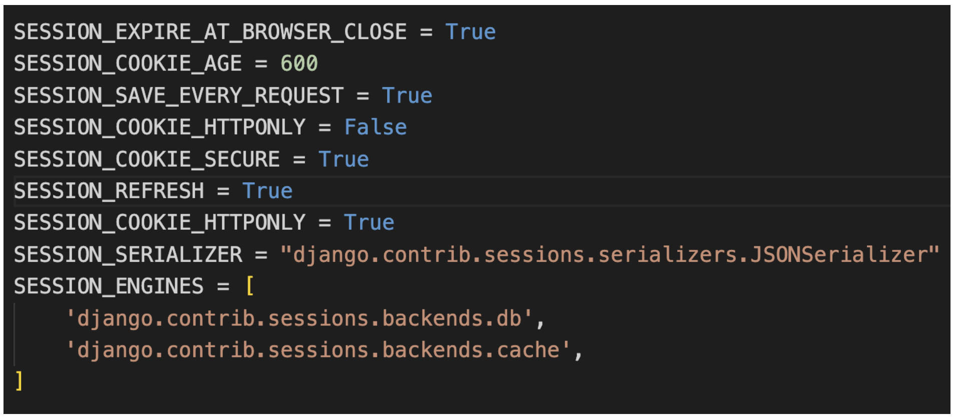


```
appid="1AD64F73C9504F03B30980E1471AD5E9"
appsecret="27DB6878G0029G4B02G9AF8GB60525F956F8"
gstin="33AAICS8653G1Z4"
```

- It is recommended to **encrypt** and **decrypt** those information like GST , passwords and tokens .

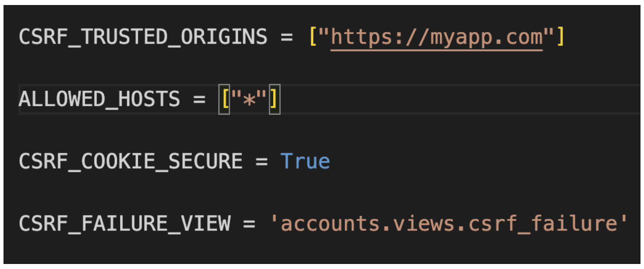
4.3 Session and Request Management

- Adding a custom session **middleware** or utilising the django session for better session uptime and cookie safing as below E.g., .



```
SESSION_EXPIRE_AT_BROWSER_CLOSE = True
SESSION_COOKIE_AGE = 600
SESSION_SAVE_EVERY_REQUEST = True
SESSION_COOKIE_HTTPONLY = False
SESSION_COOKIE_SECURE = True
SESSION_REFRESH = True
SESSION_COOKIE_HTTPONLY = True
SESSION_SERIALIZER = "django.contrib.sessions.serializers.JSONSerializer"
SESSION_ENGINES = [
    'django.contrib.sessions.backends.db',
    'django.contrib.sessions.backends.cache',
]
```

- Additional **CSRF** validation are recommended as below example



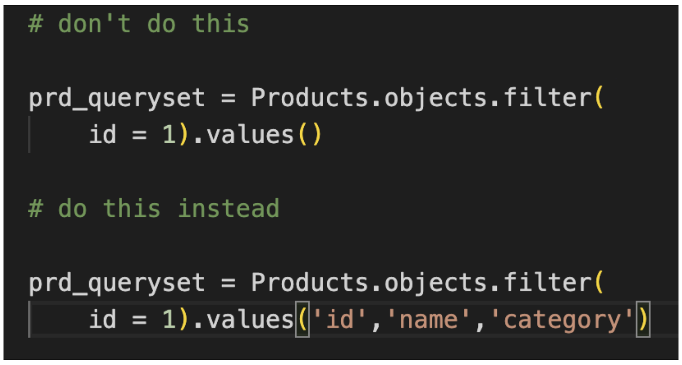
```
CSRF_TRUSTED_ORIGINS = ["https://myapp.com"]

ALLOWED_HOSTS = ["*"]

CSRF_COOKIE_SECURE = True

CSRF_FAILURE_VIEW = 'accounts.views.csrf_failure'
```

- In addition **HSTS , SSL/TLS , CORS** policy , **Proxy** configuration can be added based on the server or serverless deployment model .
- Use penetration testing tools to check the **site health** .
- REF : pen_tesing , django_checkup
- Run **python manage.py check** for security and vulnerabilities checking .

5 . PERFORMANCE AND OPTIMIZATION

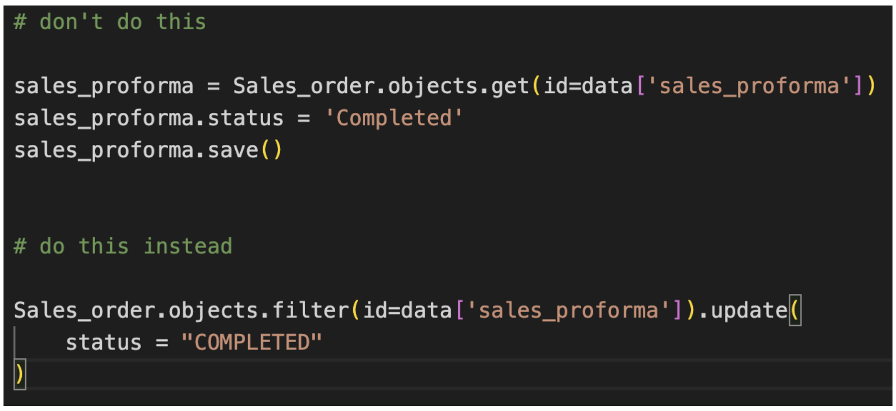
5.1 Query Optimization

- The current code base has better optimised queries and solves the N + 1 queries .
- Tips :
    - *Get what you need :* Instead of retrieving all the fields , get the required fields from the DB



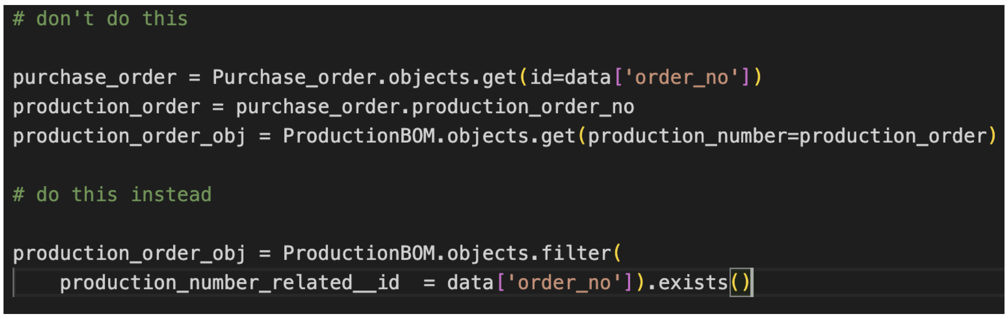
```
# don't do this

prd_queryset = Products.objects.filter(
    id = 1).values()

# do this instead

prd_queryset = Products.objects.filter(
    id = 1).values('id','name','category')
```

-
    - *Don't multiply :* calling the instance object multiple times calls the query multiple times .



```
# don't do this

sales_proforma = Sales_order.objects.get(id=data['sales_proforma'])
sales_proforma.status = 'Completed'
sales_proforma.save()


# do this instead

Sales_order.objects.filter(id=data['sales_proforma']).update(
    status = "COMPLETED"
)
```

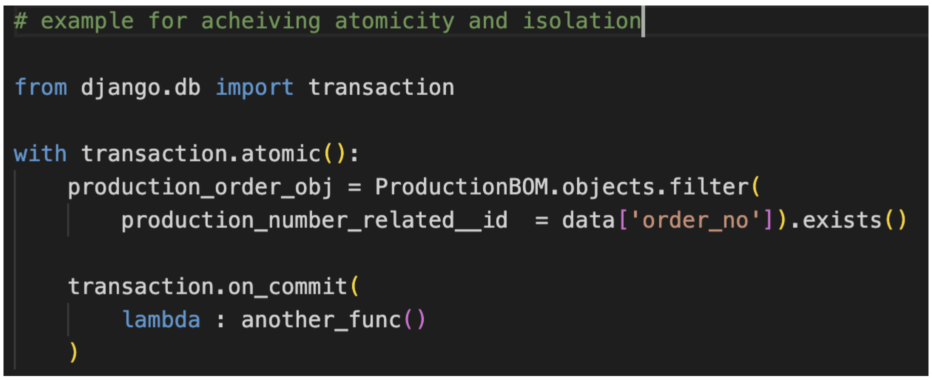
- Forward and reverse **relationships** aren't used in the codebase .
    - E.g.,



```
# don't do this

purchase_order = Purchase_order.objects.get(id=data['order_no'])
production_order = purchase_order.production_order_no
production_order_obj = ProductionBOM.objects.get(production_number=production_order)

# do this instead

production_order_obj = ProductionBOM.objects.filter(
    production_number_related__id  = data['order_no']).exists()
```

- No **ACID** properties fulfilled in the code base .

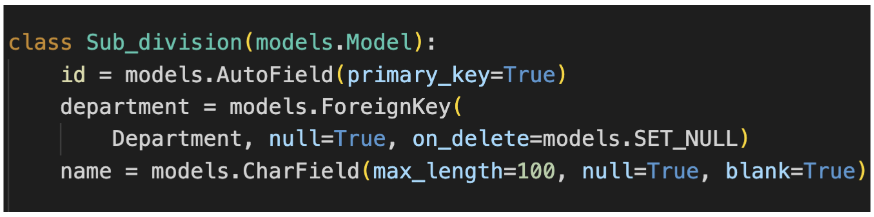
- There is a high chance of database **concurrency** problems .



```python
# example for acheiving atomicity and isolation

from django.db import transaction

with transaction.atomic():
    production_order_obj = ProductionBOM.objects.filter(
        production_number_related__id  = data['order_no']).exists()

    transaction.on_commit(
        lambda : another_func()
    )
```

5.2 Schemas and Indexing

- Never ever use **underscore** in schema definition



```python
class Sub_division(models.Model):
    id = models.AutoField(primary_key=True)
    department = models.ForeignKey(
        Department, null=True, on_delete=models.SET_NULL)
    name = models.CharField(max_length=100, null=True, blank=True)
```

- Add **indexing** for schemas for faster access .
- Add defined **string representation** for human readability .

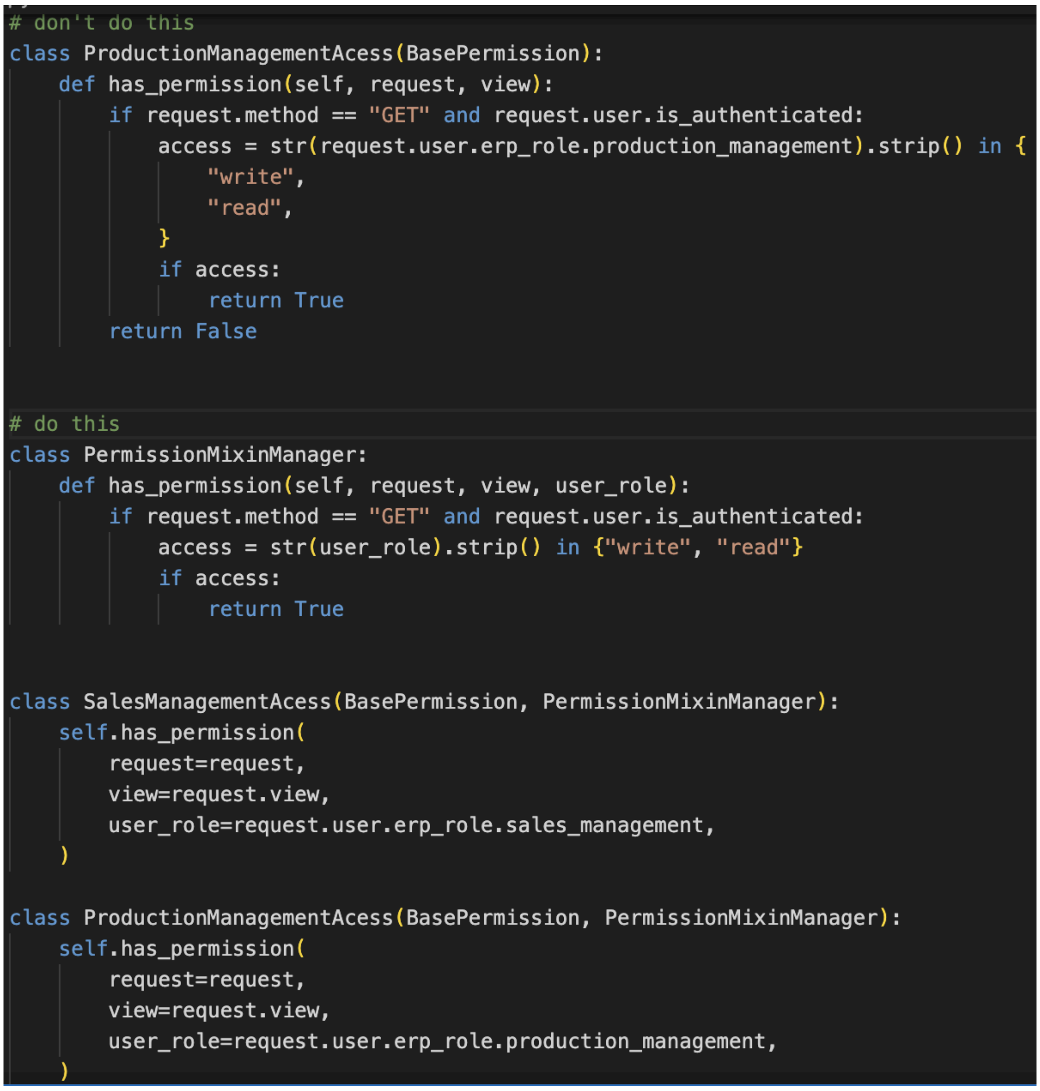
5.3 Performance tuning

- Implement **mem-cache** or redis cache mechanism for frequently used queries or objects .
- Use **CDN** to cache assets and to serve static data .
- Use **django silk** for profiling and query fetch time .
- Add **pagination** for large queryset and table data .
- Add rate limiting and **throttling** if required .
- Enable **Gzip** technique for data compression .
- Enable Database connection age and pooling if required .
- REF : Link_1 , Link_2 , profiler

6 . REUSABILITY AND REFACTORING

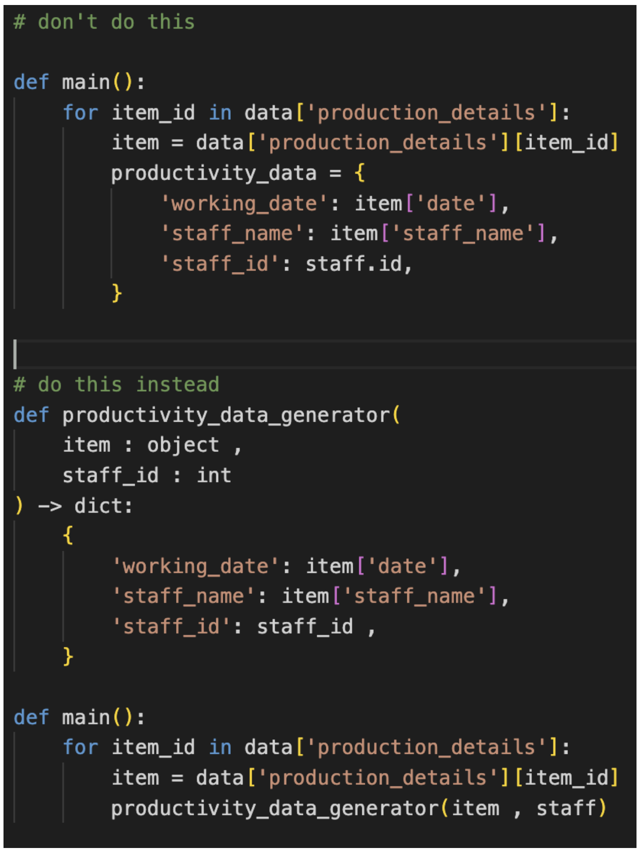
6.1 Modularity and Reusability

- In the current code most of the functional logic’s aren’t reused well .
- There are some flaws with the code design and some of them are **spaghetti** code . E.g.,



```
# don't do this
class ProductionManagementAcess(BasePermission):
    def has_permission(self, request, view):
        if request.method == "GET" and request.user.is_authenticated:
            access = str(request.user.erp_role.production_management).strip() in {
                "write",
                "read",
            }
            if access:
                return True
        return False


# do this
class PermissionMixinManager:
    def has_permission(self, request, view, user_role):
        if request.method == "GET" and request.user.is_authenticated:
            access = str(user_role).strip() in {"write", "read"}
            if access:
                return True


class SalesManagementAcess(BasePermission, PermissionMixinManager):
    self.has_permission(
        request=request,
        view=request.view,
        user_role=request.user.erp_role.sales_management,
    )

class ProductionManagementAcess(BasePermission, PermissionMixinManager):
    self.has_permission(
        request=request,
        view=request.view,
        user_role=request.user.erp_role.production_management,
    )
```

- More **fatty functions** which literally makes them less readable .



```python
# don't do this

def main():
    for item_id in data['production_details']:
        item = data['production_details'][item_id]
        productivity_data = {
            'working_date': item['date'],
            'staff_name': item['staff_name'],
            'staff_id': staff.id,
        }

# do this instead
def productivity_data_generator(
    item : object ,
    staff_id : int
) -> dict:
    {
        'working_date': item['date'],
        'staff_name': item['staff_name'],
        'staff_id': staff_id ,
    }

def main():
    for item_id in data['production_details']:
        item = data['production_details'][item_id]
        productivity_data_generator(item , staff)
```

6.2 Refactoring and Organisation

- The current codebase is a **monorepo** model , both client and server side requires refactoring of code in order to achieve scalability and **reusable** functions .
- As a result there is **tight coupling** in the codebase .
- All long and **monolithic** functions need to be cleaned and separated as chunks .
- The current code base requires a code composition and organisation strategy .
- Add **better comments** for file navigation .
- Adjust utilities , common constants and organise them in a separate folder .
- Abstract **classes** for your needs , in order to avoid code smells and anti patterns .

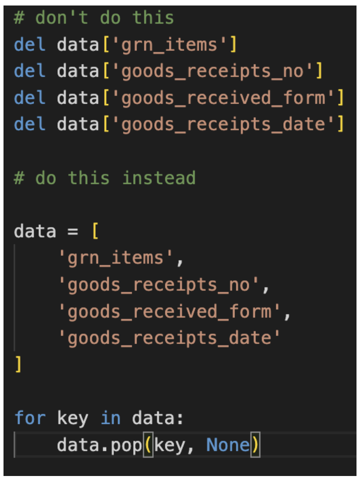
7 . AREA OF IMPROVEMENT

7.1 Design

- It is essential to improve error handling and make it more **robust** .
- More dead and unused code , remove **dead code** or tag them as unused .
- Add a **preliminary** note on each file for better explanation of what the file does .
- The code doesn't fulfil **OOAD** principles , all classes / functions are fatty .
- Use **DRY** and **KISS** principles .

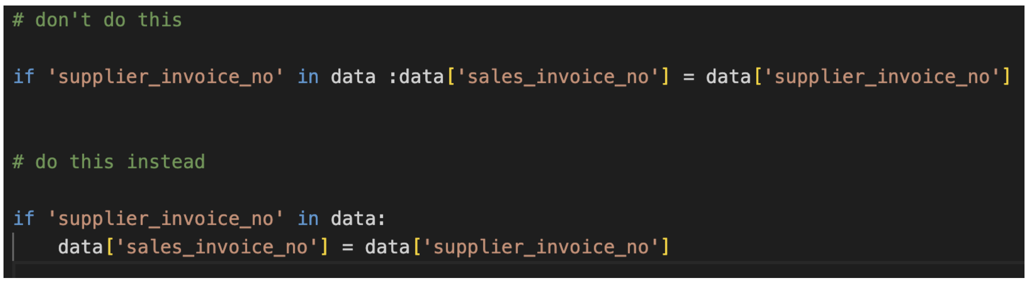
7.2 Functionality

- Use **iterations** when needed



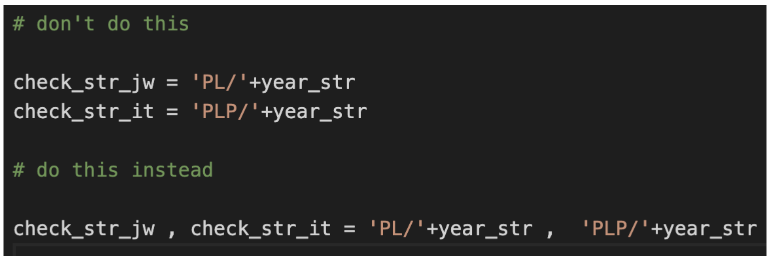
```python
# don't do this
del data['grn_items']
del data['goods_receipts_no']
del data['goods_received_form']
del data['goods_receipts_date']

# do this instead

data = [
    'grn_items',
    'goods_receipts_no',
    'goods_received_form',
    'goods_receipts_date'
]

for key in data:
    data.pop(key, None)
```

- Never use one liners , it makes them **less readable** .



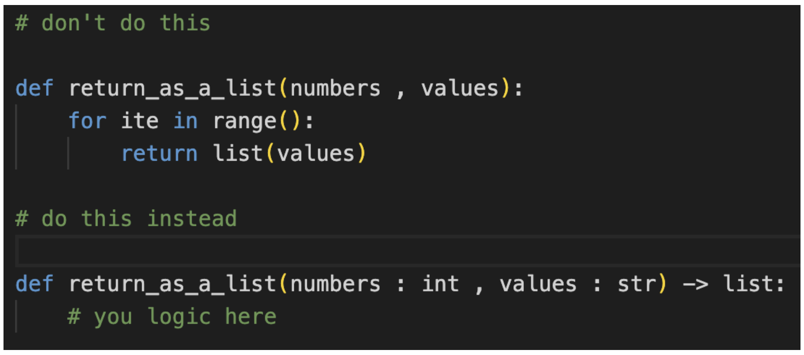
```python
# don't do this

if 'supplier_invoice_no' in data :data['sales_invoice_no'] = data['supplier_invoice_no']


# do this instead

if 'supplier_invoice_no' in data:
    data['sales_invoice_no'] = data['supplier_invoice_no']
```

- Use **one liners** for multi assignment , data unpacking , list and dict comprehensions .



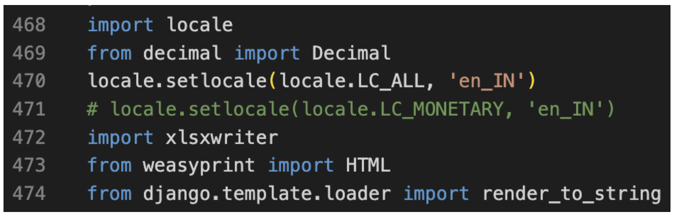
```
# don't do this

check_str_jw = 'PL/'+year_str
check_str_it = 'PLP/'+year_str

# do this instead

check_str_jw , check_str_it = 'PL/'+year_str ,  'PLP/'+year_str
```

- Add necessary **annotations** and type hints in the code base



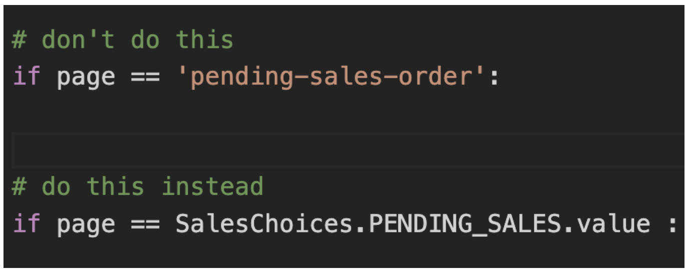
```
# don't do this

def return_as_a_list(numbers , values):
    for ite in range():
        return list(values)

# do this instead

def return_as_a_list(numbers : int , values : str) -> list:
    # you logic here
```

- Import **dependencies** always on top



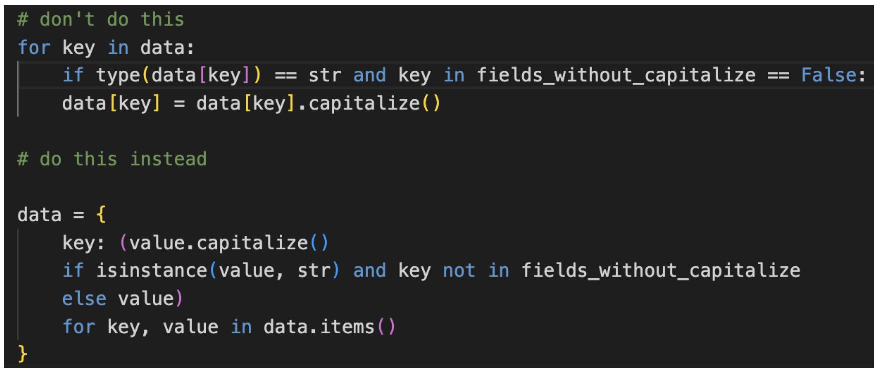
```
import locale
from decimal import Decimal
locale.setlocale(locale.LC_ALL, 'en_IN')
# locale.setlocale(locale.LC_MONETARY, 'en_IN')
import xlsxwriter
from weasyprint import HTML
from django.template.loader import render_to_string
```

- Add named **Enum** for conditions , value setters , iterations .



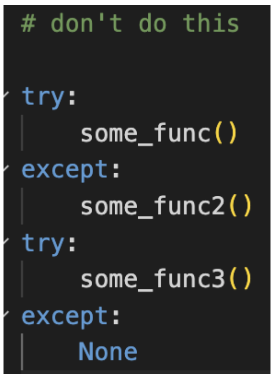
```
# don't do this
if page == 'pending-sales-order':


# do this instead
if page == SalesChoices.PENDING_SALES.value :
```

- Avoid common **iteration** mistakes



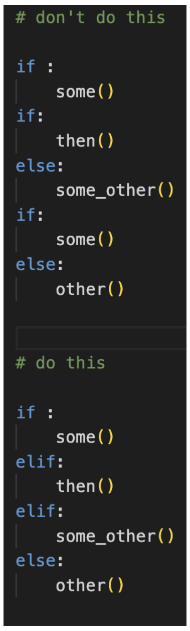
```python
# don't do this
for key in data:
    if type(data[key]) == str and key in fields_without_capitalize == False:
    data[key] = data[key].capitalize()

# do this instead

data = {
    key: (value.capitalize()
    if isinstance(value, str) and key not in fields_without_capitalize
    else value)
    for key, value in data.items()
}
```

- Use exception clauses wisely not as in below image



```python
# don't do this

try:
    some_func()
except:
    some_func2()
try:
    some_func3()
except:
    None
```

- More global values assigned , **avoid globals .**
- Arrange conditional statements based on flow scenarios.



```python
# don't do this

if :
    some()
if:
    then()
else:
    some_other()
if:
    some()
else:
    other()

# do this

if :
    some()
elif:
    then()
elif:
    some_other()
else:
    other()
```

7.3 Security Measures

- Pay attention to security practices .
- Security **audit** and application vulnerability checkup need to be taken .
- Add a **honeypot** for the admin panel and check measures for SQL injection .

7.4 Versioning

- Maintain valid version names or branch names for future code maintenance .
- Use Semantic Versioning (**SemVer**) with version tagging for better branch management .
- E.g., :
  - DEV-v1.2
  - TEST-v1.3

7.5 Management and Maintenance

- Maintain a brief **change log** to document changes in releases .
- Add a document for **rollback plan** procedures for backup and restoration .
- Provide **migration** and **installation** guidelines .
- Make file modular , split all fatty functions into chunks
- Maintain at least 300 ~ 350 lines of code (**LOC**) per file .
- There is a flaw with code structure in maintaining the **MVC** architecture .

8 . CONCLUSION

In conclusion, this code review has provided valuable insights into the quality and maintainability of the codebase. While there are strengths and positive aspects to acknowledge ,

- There are some areas where improvements can enhance the overall **code quality** .
- It is essential to prioritise the identified improvements and implement them .
- I appreciate the **dedication** and **effort** put into this codebase, but some fine tuning needs to be done .
- Don't take this code review as personally , knowledge sharing is not an insult , It is always better to **fail** and **fix** .
- Based on CEDAR all the improvements and issues are mentioned with **examples** and **actions** .
- I haven't covered code **testing** content in this code review document .

Sincerely,

Name : ANAND RAJ B
Role : SDE - L3
Date : 19-09-2023