Краткое описание системы PayHub

PayHub (Система) является системой управления заказами (СУЗ).
Система обладает следующим функционалом:

1. Обеспечивает хранение и выдачу посредством API информации о товарах и услугах, пользователях, клиентах, организациях, местах продаж и т.п.
2. Система обеспечивает возможность формирования заказов из вышеуказанных товаров или услуг, выставлени счетов и их оплату при помощи внешней подсистемы процессинга, хранение информации о доставке заказа клиенту.
3. Система обеспечивает возможность ведения программ лояльности с автоматическим зачислением на соответствующий счёт заложенных сумм после оплаты заказа.
4. Система поддерживает возможность подключения шаблонов сайтов с хранением параметров данных сайтов в привязке к местам продаж.
5. Система поддерживает возможность ведения организвационной структуры, включающей в себя организации, их пользователей (с разграничение доступа по ролям), места продаж и группы мест. Также поддерживаются возможность формирования HTTP ссылок на данные места продаж и QR кодов на основании данных ссылок.
6. Система обеспечивает базовые возможности по формированию PDF документов, в частности счетов, договоров, закрывающих документов на базе шаблонов. Также возможно формирование документов после завершения проведения заказа (например, сертификатов) с их автоматической рассылкой клиентам.
7. Система обеспечивает возможность централизованного управления несколькими инстансами Системы.
8. Система обеспечивает возможность формирования групп и коллекций товаров или услуг, а также разделения их по типам.
9. Система поддерживает возможность трансляции изменения по товарам или заказам между Системой и внешними СУЗ.
10. Для связи с внешними системами Система содержит HTTPS API с использованием RSA подписей для аутентификации сообщений.

Примеры кода

```
class BaseEntity(Base):
    __abstract__ = True

    @classmethod
    def reg_api_field(cls, column):
        try:
            cls.registered_properties.add(column.__name__)
        except AttributeError:
            cls.registered_properties = {column.__name__}
        return column

    id = mapped_column(Integer, primary_key=True, sort_order=-10)

    created_at = mapped_column(DateTime, default=datetime.datetime.now(), sort_order=-6)
    updated_at = mapped_column(DateTime, default=datetime.datetime.now(), sort_order=-5)

    def get_fields(self, *, fields_names=None, **kvargs):
        fields_info = {'id': str(self.ext_id if hasattr(self, 'ext_id') else self.id)}

        if fields_names is not None:
```

```
            for field_name in fields_names:
                if field_name.endswith('_id') and hasattr(self, field_name.removesuffix('_id')):
                    field_value = getattr(self, field_name.removesuffix('_id'))
                    if isinstance(field_value, Base):
                        try:
                            formatted_field_value = str(field_value.ext_id)
                        except:
                            formatted_field_value = str(field_value.id)
                        fields_info[field_name] = formatted_field_value
                else:
                    try:
                        fields_info[field_name] = getattr(self, f'get_{field_name}')(**kvargs)
                    except AttributeError as e:
                        try:
                            field_value = getattr(self, field_name)
                            if field_value is not None:
                                formatted_field_value = self.format_field_value(field_value)

                                fields_info[field_name] = formatted_field_value
                        except AttributeError:
                            pass

        return fields_info


[...]


class DocumentTemplate(ExternallyAccessible, Stateful, Systemic, Entity):
    __tablename__ = 'documents_templates'
    __table_args__ = {'schema': 'payhub'}

    name = mapped_column(String(128))
    description = mapped_column(Text())

    document_type_id = mapped_column(Integer, ForeignKey(DocumentType.id))
    document_type = relationship(DocumentType)

    organization_id = mapped_column(Integer, ForeignKey('payhub.organizations.id'))
    organization = relationship('Organization', foreign_keys=[organization_id])

    locale_id = mapped_column(Integer, ForeignKey('payhub.locales.id'))
    locale = relationship('Locale')

    template_definition = mapped_column(Text())

    parameters_definitions = mapped_column(JSONB)


[...]


class SystemsClustersTypesApiEndpoint(BaseApiEndpoint):
    def __init__(self, db):
        super().__init__(db)

        self.entity_class = SystemsClusterType

        self.order_variants = {
            'name': SystemsClusterType.name
        }

        self.view_roles = ['superadmin']

[...]

@app.route('/<template>', 'POST')
@app.route('/<template>/<action>', 'POST')
def route(template, action, db):
```

```
    try:
        if api_endpoints.get(template):
            import importlib
            module = importlib.import_module('api')
            class_ = getattr(module, api_endpoints.get(template))
            endpoint_instance = class_(db)

            raw_data, _ = retrieve_request_data()
            endpoint_instance.set_request_data(json.loads(raw_data))

            method = getattr(endpoint_instance, f'process_{action}' if action else
'process_request')

            if not method:
                response.status = 404
                return {'res': 'error', "message": "The action is not found."}
            else:
                return api_core.request_handler(method, db)
        else:
            proc_module = import_module('api.' + template)

            if proc_module.enabled:
                if action is None:
                    return api_core.request_handler(proc_module.process_request, db)
                else:
                    return api_core.request_handler(getattr(proc_module, 'process_' + action), db)
            else:
                response.status = 404
                return {'res': 'error', "message": "The module is not enabled."}

    except AttributeError as ex:
        api_core.get_logger('ph').exception("Failed to load module")
        response.status = 404
        return {'res': 'error', "message": str(ex)}

    except Exception as ex:
        api_core.get_logger('ph').exception(ex)
        response.status = 500
        return {"res": "error", "message": "Internal server error"}


[...]


elif file_ext == '.xls':
    wb = xlrd.open_workbook(file_contents=file_content)
    ws = wb.sheet_by_index(0)
    for row_num in range(ws.nrows):
        row = ws.row_values(row_num)
        if len(row) == 0:
            continue
        self.logger.info(row)
        if rows_to_skip > 0:
            rows_to_skip -= 1
        else:
            product_added, product_updated = self.process_product_row(row,
current_products_articles,
                                                                      organization, db)


[…]


def process_order(self, order: Order):
    if order.client_order.client:
        org_client = (self.session.query(OrganizationClient)
                      .filter(OrganizationClient.organization == order.organization)
                      .filter(OrganizationClient.client == order.client_order.client).first())
```

```
if org_client:
    if len(org_client.loyalty_cards) == 0:
        self.create_loyalty_card(org_client)

    for loyalty_card in org_client.loyalty_cards:
        if loyalty_card.type_id == LCT_LOYALTY_CARD and loyalty_card.state_id == OS_ACTIVE:
            if self.process_card(loyalty_card, order.client_order, order.total_amount):
                break
```