```
(.tabpay) ➜  tabpay git:(restructured) tree
.
├── README.md
├── api.txt
├── app
│   ├── __init__.py
│   ├── __pycache__
│   │   ├── __init__.cpython-312.pyc
│   │   ├── config.cpython-312.pyc
│   │   ├── extensions.cpython-312.pyc
│   │   └── routes.cpython-312.pyc
│   ├── api
│   │   ├── __init__.py
│   │   ├── __pycache__
│   │   │   ├── __init__.cpython-312.pyc
│   │   │   └── api.cpython-312.pyc
│   │   └── api.py
│   ├── config.py
│   ├── extensions.py
│   ├── forms
│   │   ├── __init__.py
│   │   ├── __pycache__
│   │   │   ├── __init__.cpython-312.pyc
│   │   │   └── main.cpython-312.pyc
│   │   ├── auth.py
│   │   └── main.py
│   ├── models
│   │   ├── __init__.py
│   │   ├── __pycache__
│   │   │   ├── __init__.cpython-312.pyc
│   │   │   └── models.cpython-312.pyc
│   │   └── models.py
│   ├── routes
│   │   ├── __init__.py
│   │   ├── auth.py
│   │   └── main.py
│   ├── static
│   │   ├── images
│   │   │   ├── image.jpg
│   │   │   ├── index_image(1).png
│   │   │   ├── index_image(2).png
│   │   │   ├── index_image(3).png
│   │   │   ├── index_image(4).png
│   │   │   ├── logo.png
│   │   │   └── profile_image.png
│   │   └── tabpay_css
│   │       ├── auth.css
│   │       ├── dashboard.css
│   │       └── index.css
│   ├── templates
│   │   ├── base.html
│   │   ├── block_reports.html
```

```
|   |    ├── forgot_password.html
|   |    ├── host.html
|   |    ├── index.html
|   |    ├── manage_contribution.html
|   |    ├── security
|   |   |    ├── _menu.html
|   |   |    ├── _messages.html
|   |   |    ├── base copy.html
|   |   |    ├── base.html
|   |   |    ├── change_email.html
|   |   |    ├── change_password.html
|   |   |    ├── email
|   |   |   |    ├── change_email_instructions.html
|   |   |   |    ├── change_email_instructions.txt
|   |   |   |    ├── change_notice.html
|   |   |   |    ├── change_notice.txt
|   |   |   |    ├── confirmation_instructions.html
|   |   |   |    ├── confirmation_instructions.txt
|   |   |   |    ├── login_instructions.html
|   |   |   |    ├── login_instructions.txt
|   |   |   |    ├── reset_instructions.html
|   |   |   |    ├── reset_instructions.txt
|   |   |   |    ├── reset_notice.html
|   |   |   |    ├── reset_notice.txt
|   |   |   |    ├── two_factor_instructions.html
|   |   |   |    ├── two_factor_instructions.txt
|   |   |   |    ├── two_factor_rescue.html
|   |   |   |    ├── two_factor_rescue.txt
|   |   |   |    ├── us_instructions.html
|   |   |   |    ├── us_instructions.txt
|   |   |   |    ├── welcome.html
|   |   |   |    ├── welcome.txt
|   |   |   |    ├── welcome_existing.html
|   |   |   |    ├── welcome_existing.txt
|   |   |   |    ├── welcome_existing_username.html
|   |   |   |   └── welcome_existing_username.txt
|   |   |    ├── forgot_password.html
|   |   |    ├── login_user.html
|   |   |    ├── register_user.html
|   |   |    ├── reset_password.html
|   |   |   └── send_confirmation.html
|   |    ├── settings.html
|   |   └── statistics.html
|   └── utils
|       └── __init__.py
 ├── instance
|   └── tabpay.db
 ├── requirements.txt
 ├── run.py
└── test
   └── __init__.py
```

File contents:

1. app/api/api.py:

```
from sqlalchemy.exc import SQLAlchemyError
from werkzeug.exceptions import HTTPException
from flask_restful import Resource, marshal_with, abort, fields, reqparse
from app.extensions import db
from app.models.models import UserModel, CommunicationModel, UmbrellaModel, PaymentModel, BlockModel, ZoneModel

# Argument parsers for different resources
umbrella_args = reqparse.RequestParser()
umbrella_args.add_argument('name', type=str, required=True, help='Umbrella Name is required')
umbrella_args.add_argument('location', type=str, required=True, help='Umbrella location is required')

block_args = reqparse.RequestParser()
block_args.add_argument('name', type=str, required=True, help='Block Name is required')
block_args.add_argument('parent_umbrella_id', type=int, required=True, help='Parent Umbrella is required')

zone_args = reqparse.RequestParser()
zone_args.add_argument('name', type=str, required=True, help='Zone Name is required')
zone_args.add_argument('parent_block_id', type=int, required=True, help='Parent Block is required')

user_args = reqparse.RequestParser()
user_args.add_argument('email', type=str, required=True, help='Email is required')
user_args.add_argument('password', type=str, required=True, help='Password is required')

communication_args = reqparse.RequestParser()
communication_args.add_argument('content', type=str, required=True, help='Content is required')
communication_args.add_argument('user_id', type=int, required=True, help='Author is required')

payment_args = reqparse.RequestParser()
payment_args.add_argument('payer_id', type=int, required=True, help='Payer is required')
payment_args.add_argument('source_phone_number', type=str, required=True, help='Source phone number is required')  # Changed to str
payment_args.add_argument('amount', type=float, required=True, help='Amount is required')

# Fields for serialization
user_fields = {
    "id": fields.Integer,
    "full_name": fields.String,
    "email": fields.String,
    "password": fields.String,
    "id_number": fields.Integer,
    "phone_number": fields.String,  # Changed to String
    "active": fields.Boolean,
    "zone_id": fields.Integer,
    "bank": fields.String,
    "acc_number": fields.String,  # Changed to String
    "registered_at": fields.DateTime,
    "updated_at": fields.DateTime,
    "message": fields.String(attribute="author.full_name")
}

communication_fields = {
    "id": fields.Integer,
    "content": fields.String,
    "user_id": fields.Integer,
    "created_at": fields.DateTime,
    "updated_at": fields.DateTime
}

payment_fields = {
    "id": fields.Integer,
    "payer_id": fields.Integer,
    "amount": fields.Float,
    "payment_date": fields.DateTime,
    "mpesa_id": fields.String,
    "account_number": fields.String,  # Changed to String
    "source_phone_number": fields.String,  # Changed to String
    "transaction_status": fields.Boolean
}

block_fields = {
    "id": fields.Integer,
    "name": fields.String,
    "umbrella_id": fields.Integer
}

umbrella_fields = {
    "id": fields.Integer,
    "name": fields.String,
    "location": fields.String
}

zone_fields = {
    "id": fields.Integer,
    "name": fields.String,
    "parent_block_id": fields.Integer
}
```

```python

class Users(Resource):
    @marshal_with(user_fields)
    def get(self):
        try:
            users = UserModel.query.all()
            return users, 200

        except SQLAlchemyError as e:
            db.session.rollback()
            error_message = {"error": "Database error occurred", "details": str(e)}
            abort(500, message=error_message)

        except HTTPException as e:
            error_message = {"error": "HTTP error occurred", "details": str(e)}
            abort(e.code, message=error_message)

        except Exception as e:
            error_message = {"error": "Unexpected error occurred", "details": str(e)}
            abort(500, message=error_message)

        finally:
            db.session.close()

    @marshal_with(user_fields)
    def post(self):
        try:
            args = user_args.parse_args()
            existing_user = UserModel.query.filter_by(email=args['email']).first()

            if existing_user:
                error_message = {"error": "User already exists"}
                abort(409, message=error_message)

            new_user = UserModel(**args)
            db.session.add(new_user)
            db.session.commit()
            return new_user, 201

        except SQLAlchemyError as e:
            db.session.rollback()
            error_message = {"error": "Database error occurred", "details": str(e)}
            abort(500, message=error_message)

        except HTTPException as e:
            error_message = {"error": "HTTP error occurred", "details": str(e)}
            abort(e.code, message=error_message)

        except Exception as e:
            error_message = {"error": "Unexpected error occurred", "details": str(e)}
            abort(500, message=error_message)

        finally:
            db.session.close()

class User(Resource):
    @marshal_with(user_fields)
    def get(self, id):
        try:
            user = UserModel.query.get_or_404(id)
            return user, 200

        except SQLAlchemyError as e:
            error_message = {"error": "Database error occurred", "details": str(e)}
            abort(500, message=error_message)

        except HTTPException as e:
            error_message = {"error": "HTTP error occurred", "details": str(e)}
            abort(e.code, message=error_message)

        except Exception as e:
            error_message = {"error": "Unexpected error occurred", "details": str(e)}
            abort(500, message=error_message)

        finally:
            db.session.close()

    @marshal_with(user_fields)
    def patch(self, id):
        try:
            args = user_args.parse_args()
            existing_user = UserModel.query.get_or_404(id)

            if existing_user:
                for key, value in args.items():
                    setattr(existing_user, key, value)
                db.session.commit()
                return existing_user, 200

            abort(404, message={"error": "User not found"})

        except SQLAlchemyError as e:
            db.session.rollback()
            error_message = {"error": "Database error occurred", "details": str(e)}
            abort(500, message=error_message)

        except HTTPException as e:
            error_message = {"error": "HTTP error occurred", "details": str(e)}
            abort(e.code, message=error_message)

        except Exception as e:
            error_message = {"error": "Unexpected error occurred", "details": str(e)}
            abort(500, message=error_message)

        finally:
            db.session.close()

    @marshal_with(user_fields)
    def delete(self, id):
        try:
            existing_user = UserModel.query.get_or_404(id)

            if existing_user:
                db.session.delete(existing_user)
                db.session.commit()
                users = UserModel.query.all()
                return users, 200

            abort(404, message={"error": "User not found"})

        except SQLAlchemyError as e:
            db.session.rollback()
            error_message = {"error": "Database error occurred", "details": str(e)}
            abort(500, message=error_message)

        except HTTPException as e:
            error_message = {"error": "HTTP error occurred", "details": str(e)}
            abort(e.code, message=error_message)

        except Exception as e:
            error_message = {"error": "Unexpected error occurred", "details": str(e)}
            abort(500, message=error_message)

        finally:
            db.session.close()

```

```python
class Umbrellas(Resource):
    @marshal_with(umbrella_fields)
    def get(self):
        try:
            umbrellas = UmbrellaModel.query.all()
            return umbrellas, 200

        except SQLAlchemyError as e:
            db.session.rollback()
            error_message = {"error": "Database error occurred", "details": str(e)}
            abort(500, message=error_message)

        except HTTPException as e:
            error_message = {"error": "HTTP error occurred", "details": str(e)}
            abort(e.code, message=error_message)

        except Exception as e:
            error_message = {"error": "Unexpected error occurred", "details": str(e)}
            abort(500, message=error_message)

        finally:
            db.session.close()

    @marshal_with(umbrella_fields)
    def post(self):
        try:
            args = umbrella_args.parse_args()
            new_umbrella = UmbrellaModel(**args)
            db.session.add(new_umbrella)
            db.session.commit()
            return new_umbrella, 201

        except SQLAlchemyError as e:
            db.session.rollback()
            error_message = {"error": "Database error occurred", "details": str(e)}
            abort(500, message=error_message)

        except HTTPException as e:
            error_message = {"error": "HTTP error occurred", "details": str(e)}
            abort(e.code, message=error_message)

        except Exception as e:
            error_message = {"error": "Unexpected error occurred", "details": str(e)}
            abort(500, message=error_message)

        finally:
```

```python
        db.session.close()

class Umbrella(Resource):
    @marshal_with(umbrella_fields)
    def get(self, id):
        try:
            umbrella = UmbrellaModel.query.get_or_404(id)
            return umbrella, 200

        except SQLAlchemyError as e:
            error_message = {"error": "Database error occurred", "details": str(e)}
            abort(500, message=error_message)

        except HTTPException as e:
            error_message = {"error": "HTTP error occurred", "details": str(e)}
            abort(e.code, message=error_message)

        except Exception as e:
            error_message = {"error": "Unexpected error occurred", "details": str(e)}
            abort(500, message=error_message)

        finally:
            db.session.close()

    @marshal_with(umbrella_fields)
    def patch(self, id):
        try:
            args = umbrella_args.parse_args()
            umbrella = UmbrellaModel.query.get_or_404(id)

            for key, value in args.items():
                setattr(umbrella, key, value)
            db.session.commit()
            return umbrella, 200

        except SQLAlchemyError as e:
            db.session.rollback()
            error_message = {"error": "Database error occurred", "details": str(e)}
            abort(500, message=error_message)

        except HTTPException as e:
            error_message = {"error": "HTTP error occurred", "details": str(e)}
            abort(e.code, message=error_message)

        except Exception as e:
            error_message = {"error": "Unexpected error occurred", "details": str(e)}
            abort(500, message=error_message)

        finally:
            db.session.close()

    @marshal_with(umbrella_fields)
    def delete(self, id):
        try:
            umbrella = UmbrellaModel.query.get_or_404(id)
            db.session.delete(umbrella)
            db.session.commit()
            umbrellas = UmbrellaModel.query.all()
            return umbrellas, 200

        except SQLAlchemyError as e:
            db.session.rollback()
            error_message = {"error": "Database error occurred", "details": str(e)}
            abort(500, message=error_message)

        except HTTPException as e:
            error_message = {"error": "HTTP error occurred", "details": str(e)}
            abort(e.code, message=error_message)

        except Exception as e:
            error_message = {"error": "Unexpected error occurred", "details": str(e)}
            abort(500, message=error_message)

        finally:
            db.session.close()



class Communications(Resource):
    @marshal_with(communication_fields)
    def get(self):
        try:
            communications = CommunicationModel.query.all()
            return communications, 200

        except SQLAlchemyError as e:
            db.session.rollback()
            error_message = {"error": "Database error occurred", "details": str(e)}
            abort(500, message=error_message)
```

```python

        except HTTPException as e:
            error_message = {"error": "HTTP error occurred", "details": str(e)}
            abort(e.code, message=error_message)

        except Exception as e:
            error_message = {"error": "Unexpected error occurred", "details": str(e)}
            abort(500, message=error_message)

        finally:
            db.session.close()

    @marshal_with(communication_fields)
    def post(self):
        try:
            args = communication_args.parse_args()
            new_communication = CommunicationModel(**args)
            db.session.add(new_communication)
            db.session.commit()
            return new_communication, 201

        except SQLAlchemyError as e:
            db.session.rollback()
            error_message = {"error": "Database error occurred", "details": str(e)}
            abort(500, message=error_message)

        except HTTPException as e:
            error_message = {"error": "HTTP error occurred", "details": str(e)}
            abort(e.code, message=error_message)

        except Exception as e:
            error_message = {"error": "Unexpected error occurred", "details": str(e)}
            abort(500, message=error_message)

        finally:
            db.session.close()

class Communication(Resource):
    @marshal_with(communication_fields)
    def get(self, id):
        try:
            communication = CommunicationModel.query.get_or_404(id)
            return communication, 200

        except SQLAlchemyError as e:
            error_message = {"error": "Database error occurred", "details": str(e)}
            abort(500, message=error_message)

        except HTTPException as e:
            error_message = {"error": "HTTP error occurred", "details": str(e)}
            abort(e.code, message=error_message)

        except Exception as e:
            error_message = {"error": "Unexpected error occurred", "details": str(e)}
            abort(500, message=error_message)

        finally:
            db.session.close()

    @marshal_with(communication_fields)
    def patch(self, id):
        try:
            args = communication_args.parse_args()
            communication = CommunicationModel.query.get_or_404(id)

            for key, value in args.items():
                setattr(communication, key, value)
            db.session.commit()
            return communication, 200

        except SQLAlchemyError as e:
            db.session.rollback()
            error_message = {"error": "Database error occurred", "details": str(e)}
            abort(500, message=error_message)

        except HTTPException as e:
            error_message = {"error": "HTTP error occurred", "details": str(e)}
            abort(e.code, message=error_message)

        except Exception as e:
            error_message = {"error": "Unexpected error occurred", "details": str(e)}
            abort(500, message=error_message)

        finally:
            db.session.close()

    @marshal_with(communication_fields)
    def delete(self, id):
        try:
            communication = CommunicationModel.query.get_or_404(id)
            db.session.delete(communication)
            db.session.commit()
            communications = CommunicationModel.query.all()
            return communications, 200

        except SQLAlchemyError as e:
            db.session.rollback()
            error_message = {"error": "Database error occurred", "details": str(e)}
            abort(500, message=error_message)

        except HTTPException as e:
            error_message = {"error": "HTTP error occurred", "details": str(e)}
            abort(e.code, message=error_message)

        except Exception as e:
            error_message = {"error": "Unexpected error occurred", "details": str(e)}
            abort(500, message=error_message)

        finally:
            db.session.close()



class Payments(Resource):
    @marshal_with(payment_fields)
    def get(self):
        try:
            payments = PaymentModel.query.all()
            return payments, 200

        except SQLAlchemyError as e:
            db.session.rollback()
            error_message = {"error": "Database error occurred", "details": str(e)}
            abort(500, message=error_message)

        except HTTPException as e:
            error_message = {"error": "HTTP error occurred", "details": str(e)}
            abort(e.code, message=error_message)

        except Exception as e:
            error_message = {"error": "Unexpected error occurred", "details": str(e)}
            abort(500, message=error_message)

        finally:
            db.session.close()

    @marshal_with(payment_fields)
    def post(self):
        try:
            args = payment_args.parse_args()
            new_payment = PaymentModel(**args)
            db.session.add(new_payment)
            db.session.commit()
            return new_payment, 201

        except SQLAlchemyError as e:
            db.session.rollback()
            error_message = {"error": "Database error occurred", "details": str(e)}
            abort(500, message=error_message)

        except HTTPException as e:
            error_message = {"error": "HTTP error occurred", "details": str(e)}
            abort(e.code, message=error_message)

        except Exception as e:
            error_message = {"error": "Unexpected error occurred", "details": str(e)}
            abort(500, message=error_message)

        finally:
            db.session.close()

class Payment(Resource):
    @marshal_with(payment_fields)
    def get(self, id):
        try:
            payment = PaymentModel.query.get_or_404(id)
            return payment, 200

        except SQLAlchemyError as e:
            error_message = {"error": "Database error occurred", "details": str(e)}
            abort(500, message=error_message)

        except HTTPException as e:
            error_message = {"error": "HTTP error occurred", "details": str(e)}
            abort(e.code, message=error_message)

        except Exception as e:
            error_message = {"error": "Unexpected error occurred", "details": str(e)}
            abort(500, message=error_message)

        finally:
            db.session.close()

    @marshal_with(payment_fields)
```

```python
def patch(self, id):
    try:
        args = payment_args.parse_args()
        payment = PaymentModel.query.get_or_404(id)

        for key, value in args.items():
            setattr(payment, key, value)
        db.session.commit()
        return payment, 200

    except SQLAlchemyError as e:
        db.session.rollback()
        error_message = {"error": "Database error occurred", "details": str(e)}
        abort(500, message=error_message)

    except HTTPException as e:
        error_message = {"error": "HTTP error occurred", "details": str(e)}
        abort(e.code, message=error_message)

    except Exception as e:
        error_message = {"error": "Unexpected error occurred", "details": str(e)}
        abort(500, message=error_message)

    finally:
        db.session.close()

@marshal_with(payment_fields)
def delete(self, id):
    try:
        payment = PaymentModel.query.get_or_404(id)
        db.session.delete(payment)
        db.session.commit()
        payments = PaymentModel.query.all()
        return payments, 200

    except SQLAlchemyError as e:
        db.session.rollback()
        error_message = {"error": "Database error occurred", "details": str(e)}
        abort(500, message=error_message)

    except HTTPException as e:
        error_message = {"error": "HTTP error occurred", "details": str(e)}
        abort(e.code, message=error_message)

    except Exception as e:
        error_message = {"error": "Unexpected error occurred", "details": str(e)}
```

```python
            abort(500, message=error_message)

        finally:
            db.session.close()


class Blocks(Resource):
    @marshal_with(block_fields)
    def get(self):
        try:
            blocks = BlockModel.query.all()
            return blocks, 200

        except SQLAlchemyError as e:
            db.session.rollback()
            error_message = {"error": "Database error occurred", "details": str(e)}
            abort(500, message=error_message)

        except HTTPException as e:
            error_message = {"error": "HTTP error occurred", "details": str(e)}
            abort(e.code, message=error_message)

        except Exception as e:
            error_message = {"error": "Unexpected error occurred", "details": str(e)}
            abort(500, message=error_message)

        finally:
            db.session.close()

    @marshal_with(block_fields)
    def post(self):
        try:
            args = block_args.parse_args()
            new_block = BlockModel(**args)
            db.session.add(new_block)
            db.session.commit()
            return new_block, 201

        except SQLAlchemyError as e:
            db.session.rollback()
            error_message = {"error": "Database error occurred", "details": str(e)}
            abort(500, message=error_message)

        except HTTPException as e:
            error_message = {"error": "HTTP error occurred", "details": str(e)}
            abort(e.code, message=error_message)

        except Exception as e:
            error_message = {"error": "Unexpected error occurred", "details": str(e)}
            abort(500, message=error_message)

        finally:
            db.session.close()

class Block(Resource):
    @marshal_with(block_fields)
    def get(self, id):
        try:
            block = BlockModel.query.get_or_404(id)
            return block, 200

        except SQLAlchemyError as e:
            error_message = {"error": "Database error occurred", "details": str(e)}
            abort(500, message=error_message)

        except HTTPException as e:
            error_message = {"error": "HTTP error occurred", "details": str(e)}
            abort(e.code, message=error_message)

        except Exception as e:
            error_message = {"error": "Unexpected error occurred", "details": str(e)}
            abort(500, message=error_message)

        finally:
            db.session.close()

    @marshal_with(block_fields)
    def patch(self, id):
        try:
            args = block_args.parse_args()
            block = BlockModel.query.get_or_404(id)

            for key, value in args.items():
                setattr(block, key, value)
            db.session.commit()
            return block, 200

        except SQLAlchemyError as e:
            db.session.rollback()
            error_message = {"error": "Database error occurred", "details": str(e)}
            abort(500, message=error_message)

        except HTTPException as e:
            error_message = {"error": "HTTP error occurred", "details": str(e)}
            abort(e.code, message=error_message)

        except Exception as e:
            error_message = {"error": "Unexpected error occurred", "details": str(e)}
            abort(500, message=error_message)

        finally:
            db.session.close()

    @marshal_with(block_fields)
    def delete(self, id):
        try:
            block = BlockModel.query.get_or_404(id)
            db.session.delete(block)
            db.session.commit()
            blocks = BlockModel.query.all()
            return blocks, 200

        except SQLAlchemyError as e:
            db.session.rollback()
            error_message = {"error": "Database error occurred", "details": str(e)}
            abort(500, message=error_message)

        except HTTPException as e:
            error_message = {"error": "HTTP error occurred", "details": str(e)}
            abort(e.code, message=error_message)

        except Exception as e:
            error_message = {"error": "Unexpected error occurred", "details": str(e)}
            abort(500, message=error_message)

        finally:
            db.session.close()


class Zones(Resource):
    @marshal_with(zone_fields)
    def get(self):
        try:
            zones = ZoneModel.query.all()
```

```python
            return zones, 200

        except SQLAlchemyError as e:
            db.session.rollback()
            error_message = {"error": "Database error occurred", "details": str(e)}
            abort(500, message=error_message)

        except HTTPException as e:
            error_message = {"error": "HTTP error occurred", "details": str(e)}
            abort(e.code, message=error_message)

        except Exception as e:
            error_message = {"error": "Unexpected error occurred", "details": str(e)}
            abort(500, message=error_message)

        finally:
            db.session.close()

    @marshal_with(zone_fields)
    def post(self):
        try:
            args = zone_args.parse_args()
            new_zone = ZoneModel(**args)
            db.session.add(new_zone)
            db.session.commit()
            return new_zone, 201

        except SQLAlchemyError as e:
            db.session.rollback()
            error_message = {"error": "Database error occurred", "details": str(e)}
            abort(500, message=error_message)

        except HTTPException as e:
            error_message = {"error": "HTTP error occurred", "details": str(e)}
            abort(e.code, message=error_message)

        except Exception as e:
            error_message = {"error": "Unexpected error occurred", "details": str(e)}
            abort(500, message=error_message)

        finally:
            db.session.close()

class Zone(Resource):
    @marshal_with(zone_fields)
    def get(self, id):
        try:
            zone = ZoneModel.query.get_or_404(id)
            return zone, 200

        except SQLAlchemyError as e:
            error_message = {"error": "Database error occurred", "details": str(e)}
            abort(500, message=error_message)

        except HTTPException as e:
            error_message = {"error": "HTTP error occurred", "details": str(e)}
            abort(e.code, message=error_message)

        except Exception as e:
            error_message = {"error": "Unexpected error occurred", "details": str(e)}
            abort(500, message=error_message)

        finally:
            db.session.close()

    @marshal_with(zone_fields)
    def patch(self, id):
        try:
            args = zone_args.parse_args()
            zone = ZoneModel.query.get_or_404(id)

            for key, value in args.items():
                setattr(zone, key, value)
            db.session.commit()
            return zone, 200

        except SQLAlchemyError as e:
            db.session.rollback()
            error_message = {"error": "Database error occurred", "details": str(e)}
            abort(500, message=error_message)

        except HTTPException as e:
            error_message = {"error": "HTTP error occurred", "details": str(e)}
            abort(e.code, message=error_message)

        except Exception as e:
            error_message = {"error": "Unexpected error occurred", "details": str(e)}
            abort(500, message=error_message)

        finally:
            db.session.close()

```

```
@marshal_with(zone_fields)
def delete(self, id):
    try:
        zone = ZoneModel.query.get_or_404(id)
        db.session.delete(zone)
        db.session.commit()
        zones = ZoneModel.query.all()
        return zones, 200

    except SQLAlchemyError as e:
        db.session.rollback()
        error_message = {"error": "Database error occurred", "details": str(e)}
        abort(500, message=error_message)

    except HTTPException as e:
        error_message = {"error": "HTTP error occurred", "details": str(e)}
        abort(e.code, message=error_message)

    except Exception as e:
        error_message = {"error": "Unexpected error occurred", "details": str(e)}
        abort(500, message=error_message)

    finally:
        db.session.close()

```

2. app/forms/auth.py

```
from flask_security.forms import RegisterForm
from wtforms import StringField,IntegerField
from wtforms.validators import DataRequired, Length

class ExtendedRegisterForm(RegisterForm):
    full_name = StringField('Please enter your Full Names', validators=[DataRequired(), Length(min=4,
max=20)],render_kw={'placeholder':'Jiara Martins'})
    id_number = IntegerField('ID No:', validators=[DataRequired()],render_kw={'placeholder':'xxxxxxxx'})
```

3. app/forms/main.py

```
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SelectField, IntegerField, SubmitField
from wtforms.validators import DataRequired, Length, ValidationError,EqualTo
from app.models.models import UserModel

class AddMemberForm(FlaskForm):
    full_name = StringField('Member Full Name',validators=[DataRequired(),
Length(max=100,min=10)],render_kw={'placeholder':'Patrick Cheruiyot'})
    id_number = IntegerField('Member ID
Number',validators=[DataRequired()],render_kw={'placeholder':'xxxxxxxx'})
    phone_number = IntegerField('Phone
Number',validators=[DataRequired()],render_kw={'placeholder':'0798543234'})
    member_zone = SelectField('Member Zone', choices=[('Zone 1', 'Zone 1'), ('Zone 2', 'Zone
2')],validators=[DataRequired()])
    bank = SelectField('Select Bank', choices=[('Equity', 'Equity'), ('DTB', 'DTB')],validators=[DataRequired()])
    acc_number = IntegerField('Bank Account
Number',validators=[DataRequired()],render_kw={'placeholder':'xxxxxx'})
    submit = SubmitField('SAVE')

    def validate_id_number(self,id_number):
        user = UserModel.query.filter_by(id_number=id_number.data).first()
        if user:
            raise ValidationError('Member ID already exists')

    def validate_phone_number(self, phone_number):
        user = UserModel.query.filter_by(phone_number=phone_number.data).first()
        if user:
            raise ValidationError('Member phone number already exists')




class ProfileForm(FlaskForm):
    full_name = StringField('Update Your Full Names',validators=[DataRequired(),
Length(max=100,min=10)],render_kw={'placeholder':'Patrick Cheruiyot'})
    id_number = IntegerField('Update Your ID',validators=[DataRequired()],render_kw={'placeholder':'xxxxxxxx'})
    password = PasswordField('Password',validators=[DataRequired(),
Length(max=100,min=6)],render_kw={'placeholder':'******'})
    confirm_password = PasswordField('Confirm Password',validators=[DataRequired(),
Length(max=100,min=6),EqualTo('password',message="Passwords do not
match!")],render_kw={'placeholder':'******'})
    submit = SubmitField('SUBMIT')

    def validate_id_number(self,id_number):
        user = UserModel.query.filter_by(id_number=id_number.form.data).first()
        if user:
            raise ValidationError('Member ID already exists')



class AddCommitteForm(FlaskForm):
    full_name = StringField('Committee Full Name',validators=[DataRequired(),
Length(max=100,min=10)],render_kw={'placeholder':'Patrick Cheruiyot'})
    id_number = IntegerField('Their ID Number',validators=[DataRequired()],render_kw={'placeholder':'xxxxxxxx'})
    role = SelectField('Role', choices=[('Chairman', 'Chairman'), ('Secretary',
'Secretary')],validators=[DataRequired()])
    phone_number = IntegerField('Phone
Number',validators=[DataRequired()],render_kw={'placeholder':'0798543234'})
    submit = SubmitField('SUBMIT')

    def validate_id_number(self,id_number):
        user = UserModel.query.filter_by(id_number=id_number.form.data).first()
        if user:
            raise ValidationError('Member ID already exists')

    def validate_phone_number(self, phone_number):
        user = UserModel.query.filter_by(phone_number=phone_number.data).first()
        if user:
            raise ValidationError('Member phone number already exists')



class UmbrellaForm(FlaskForm):
    umbrella_name = StringField('Umbrella Name',validators=[DataRequired(),
Length(max=100,min=4)],render_kw={'placeholder':'Nyangores'})
    location = StringField('Location',validators=[DataRequired(),
Length(max=100,min=4)],render_kw={'placeholder':'xxxxxxx'})
    submit = SubmitField('SUBMIT')


class BlockForm(FlaskForm):
    block_name = StringField('Block Name',validators=[DataRequired(),
Length(max=100,min=4)],render_kw={'placeholder':'Block 5'})
    parent_umbrella = SelectField('Parent Umbrella', choices=[('Nyangores', 'Nyangores'), ('Meja',
'Meja')],validators=[DataRequired()])
    submit = SubmitField('SUBMIT')



class ZoneForm(FlaskForm):
    zone_name = StringField('Zone Name',validators=[DataRequired(),
Length(max=100,min=4)],render_kw={'placeholder':'Meja Estate zone'})
```

```
    parent_block =  SelectField('Parent Block', choices=[('Block 1', 'Block 1'), ('Block 2', 'Block 
2')],validators=[DataRequired()])
    submit = SubmitField('SUBMIT')
```

4.app/models/models.py

```
from flask_security import UserMixin, RoleMixin, SQLAlchemyUserDatastore
from flask_security.utils import hash_password
from ..extensions import db
import uuid

# Association table for many-to-many relationship between User and Block
member_blocks = db.Table('member_blocks',
    db.Column('user_id', db.Integer, db.ForeignKey('users.id'), primary_key=True),
    db.Column('block_id', db.Integer, db.ForeignKey('blocks.id'), primary_key=True)
)

# Association table for many-to-many relationship between User and Role
roles_users = db.Table('roles_users',
    db.Column('user_id', db.Integer, db.ForeignKey('users.id'), primary_key=True),
    db.Column('role_id', db.Integer, db.ForeignKey('roles.id'), primary_key=True)
)

class Role(db.Model, RoleMixin):
    __tablename__ = 'roles'

    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(80), unique=True, nullable=False)
    description = db.Column(db.String(255), nullable=True)



    def __repr__(self):
        return f"<Role {self.name}>"

class UserModel(db.Model, UserMixin):
    __tablename__ = 'users'

    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(255), unique=True, nullable=True)  # Email may be null for non-login members
    password = db.Column(db.String(255), nullable=True)  # Auto-generated password can be nullable
    full_name = db.Column(db.String(255))
    id_number = db.Column(db.Integer, index=True,unique=True)
    phone_number = db.Column(db.String(80), unique=True, index=True)
    active = db.Column(db.Boolean, default=True)
```

```python
    bank = db.Column(db.String(50))
    acc_number = db.Column(db.String(50))
    registered_at = db.Column(db.DateTime, default=db.func.current_timestamp())
    updated_at = db.Column(db.DateTime, default=db.func.current_timestamp(),
onupdate=db.func.current_timestamp())
    fs_uniquifier = db.Column(db.String(64), unique=True, nullable=False, default=lambda: str(uuid.uuid4()))
    zone = db.Column(db.String(100))
    confirmed_at = db.Column(db.DateTime)
    webauth = db.relationship('WebAuth', backref='user', uselist=False)


    # Relationships
    roles = db.relationship('Role', secondary=roles_users, backref=db.backref('users', lazy='dynamic'))
    messages = db.relationship('CommunicationModel', backref='author', lazy=True)
    payments = db.relationship('PaymentModel', backref='payer', lazy=True)

    # Many-to-many relationship with blocks
    block_memberships = db.relationship('BlockModel', secondary=member_blocks, backref=db.backref('users',
lazy=True))

    # Password auto-generation method
    def generate_auto_password(self):
        import random, string
        password = ''.join(random.choices(string.ascii_letters + string.digits, k=8))
        self.password = hash_password(password)
        return password

    def __repr__(self):
        return f"<Member {self.full_name}>"

class WebAuth(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('users.id'))
    auth_token = db.Column(db.String(255), unique=True, nullable=False)

class UmbrellaModel(db.Model):
    __tablename__ = 'umbrellas'

    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(255), nullable=False, unique=True)
    location = db.Column(db.String(255), nullable=False)
    created_by = db.Column(db.Integer, db.ForeignKey('users.id'))
    blocks = db.relationship('BlockModel', backref='parent_umbrella', lazy=True)

    def __repr__(self):
        return f"<Umbrella {self.name}>"
```

```python
class BlockModel(db.Model):
    __tablename__ = 'blocks'

    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(255), nullable=False)
    parent_umbrella_id = db.Column(db.Integer, db.ForeignKey('umbrellas.id'), nullable=False)
    zones = db.relationship('ZoneModel', backref='parent_block', lazy=True)
    payments = db.relationship('PaymentModel', backref='block_payments', lazy=True)
    created_by = db.Column(db.Integer, db.ForeignKey('users.id'))



class ZoneModel(db.Model):
    __tablename__ = 'zones'

    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(20), nullable=False)
    parent_block_id = db.Column(db.Integer, db.ForeignKey("blocks.id"), nullable=False)
    created_by = db.Column(db.Integer, db.ForeignKey('users.id'))


    def __repr__(self):
        return f"<Zone {self.name}>"

class PaymentModel(db.Model):
    __tablename__ = 'payments'

    id = db.Column(db.Integer, primary_key=True)
    mpesa_id = db.Column(db.String(255), nullable=False)
    account_number = db.Column(db.String(80), nullable=False)
    source_phone_number = db.Column(db.String(80), nullable=False)
    amount = db.Column(db.Integer, nullable=False)
    payment_date = db.Column(db.DateTime, default=db.func.current_timestamp())
    transaction_status = db.Column(db.Boolean, default=False)

    # Payment association with a specific block
    block_id = db.Column(db.Integer, db.ForeignKey('blocks.id'), nullable=False)

    # Payment association with a specific user (payer)
    payer_id = db.Column(db.Integer, db.ForeignKey('users.id'), nullable=False)

    def __repr__(self):
        return f"<Payment {self.amount} by Member {self.payer_id}>"

    @classmethod
    def get_contributions_by_member(cls, user_id):
        """Get all contributions made by a specific member."""
        return cls.query.filter_by(payer_id=user_id).all()

    @classmethod
    def get_contributions_by_block(cls, block_id):
        """Get all contributions for a specific block."""
        return cls.query.filter_by(block_id=block_id).all()



class CommunicationModel(db.Model):
    __tablename__ = 'communications'

    id = db.Column(db.Integer, primary_key=True)
    content = db.Column(db.String(255), nullable=False)
    created_at = db.Column(db.DateTime, default=db.func.current_timestamp())
    member_id = db.Column(db.Integer, db.ForeignKey('users.id'), nullable=False)

    def __repr__(self):
        return f"<Message from Member {self.member_id}>"

# Setup Flask-Security
user_datastore = SQLAlchemyUserDatastore(db, UserModel, Role)
```

5. app/routes/main.py

```
from app import create_app as app
from flask import render_template,redirect,url_for,request,flash
from flask_security import  roles_required, current_user
from flask_security.utils import hash_password
from app.extensions import db
from app.models.models import UserModel,UmbrellaModel, BlockModel,ZoneModel,user_datastore




# Profile route
@app.route('/settings/profile', methods=['GET', 'POST'])
@roles_required('Admin')
def settings_profile():
    if request.method == 'POST':
        # Update user profile logic here
        full_name = request.form.get('fullName')
        id_number = request.form.get('id_number')
        new_password = request.form.get('newPassword')
        confirm_password = request.form.get('confirmPassword')
        # Ensure passwords match and apply other validations
        if new_password == confirm_password:
            current_user.full_name = full_name
            current_user.id_number = id_number
            if new_password:
                current_user.password = hash_password(new_password)
            db.session.commit()
            flash('Profile updated successfully!')
        else:
            flash('Passwords do not match!')

        return redirect(url_for('settings_profile'))
    return render_template('settings/profile.html')
```

6. app/__init__.py

```
from flask import Flask
from flask_restful import Api
from app import config
from flask_restful import Api
from app.api.api import
Users,User,Communications,Communication,Payments,Payment,Blocks,Block,Umbrellas,Umbrella,Zones,Zone
from flask_security import  SQLAlchemyUserDatastore
from app.models.models import UserModel, Role
from app.extensions import security, db, mail
# from app.forms.auth import ExtendedRegisterForm

def create_app():
    app = Flask(__name__)
    app.config.from_object(config)

    # Initialize extensions
    db.init_app(app)
    mail.init_app(app)

    # Setup Flask-Security
    user_datastore = SQLAlchemyUserDatastore(db, UserModel, Role)
    security.init_app(app, user_datastore)

    # Initialize Flask-RESTful API
    api = Api(app)
    api.add_resource(Users, '/api/v1/users/')
    api.add_resource(User, '/api/v1/users/<int:id>/')
    api.add_resource(Communications, '/api/v1/communications/')
    api.add_resource(Communication, '/api/v1/communications/<int:id>/')
    api.add_resource(Payments, '/api/v1/payments/')
    api.add_resource(Payment, '/api/v1/payments/<int:id>/')
    api.add_resource(Blocks, '/api/v1/blocks/')
    api.add_resource(Block, '/api/v1/blocks/<int:id>/')
    api.add_resource(Umbrellas, '/api/v1/umbrellas/')
    api.add_resource(Umbrella, '/api/v1/umbrellas/<int:id>/')
    api.add_resource(Zones, '/api/v1/zones/')
    api.add_resource(Zone, '/api/v1/zones/<int:id>/')


    return app
```

7. app/config.py

```
import os
import secrets
# from .forms.auth import ExtendedRegisterForm

SECRET_KEY =  secrets.token_hex(16)
# SQLALCHEMY_DATABASE_URI = 'postgresql://captain:captain@localhost:5432/tabpay'
SQLALCHEMY_DATABASE_URI = 'sqlite:///tabpay.db'
SQLALCHEMY_TRACK_MODIFICATIONS = False
SECURITY_PASSWORD_SALT = '201343284857125688191020663358661879047'
SECURITY_REGISTERABLE = True
# SECURITY_REGISTER_FORM = ExtendedRegisterForm
SECURITY_POST_LOGIN_VIEW = '/statistics'
SECURITY_POST_LOGOUT_VIEW = '/'
SECURITY_POST_REGISTER_VIEW = '/login'
SECURITY_CONFIRMABLE = True
SECURITY_RECOVERABLE = True


# Cookie settings
REMEMBER_COOKIE_SAMESITE = 'strict' #server side
SESSION_COOKIE_SAMESITE = 'strict' # client side



# Configuration for Gmail's SMTP server
MAIL_SERVER = 'smtp.gmail.com'
MAIL_PORT = 587
MAIL_USERNAME = 'enockbett427@gmail.com'
MAIL_PASSWORD = 'ypsh pumk lluj hkeu'
MAIL_USE_TLS = True
MAIL_DEFAULT_SENDER = 'enockbett427@gmail.com'

SECURITY_CHANGE_EMAIL = True
```

8. extensions.py

```
from flask_sqlalchemy import SQLAlchemy
from flask_security import Security
# from flask import Blueprint
from flask_mailman import Mail

db = SQLAlchemy()
security = Security()
mail = Mail()

main_blueprint = Blueprint('main', __name__,template_folder='templates',static_folder='static')
auth_blueprint = Blueprint('auth',__name__,template_folder='templates',static_folder='static' )

```

9. run.py

```
from app import create_app
from app.extensions import db
from app.models.models import user_datastore
from flask_security.utils import hash_password
from app import create_app as app
from flask_security import  roles_required, current_user , login_required
from flask_security.utils import hash_password
from app.extensions import db
from app.models.models import UserModel,UmbrellaModel, BlockModel,ZoneModel,user_datastore
from app.forms.main import
AddMemberForm,ProfileForm,AddCommitteForm,UmbrellaForm,BlockForm,ZoneForm
from flask import render_template, flash, redirect, url_for, jsonify, request


app = create_app()

from flask import render_template,redirect,url_for




@app.route('/settings', methods=['GET'])
@roles_required('Umbrella_creator')
@login_required
def settings():

   # Instantiate all forms
   profile_form = ProfileForm()
   umbrella_form = UmbrellaForm()
   committee_form = AddCommitteForm()
   block_form = BlockForm()
   member_form = AddMemberForm()
```

```python
    zone_form = ZoneForm()


    # Render the settings page
    return render_template('settings.html', title='Dashboard | Settings',
                           profile_form=profile_form,
                           umbrella_form=umbrella_form,
                           committee_form=committee_form,
                           block_form=block_form,
                           zone_form=zone_form,
                           member_form=member_form,
                           user=current_user
                           )

# Profile Update Route
@app.route('/settings/update_profile', methods=['POST'])
def update_profile():
    profile_form = ProfileForm()
    if profile_form.validate_on_submit():
        user = UserModel.query.filter_by(id=current_user.id).first()
        if user:
            user.full_name = profile_form.full_name.data
            user.id_number = profile_form.id_number.data
            if profile_form.password.data:
                user.password = hash_password(profile_form.password.data)
            db.session.commit()
            flash('Profile updated successfully!', 'success')
        else:
            flash('User not found!', 'danger')
        return redirect(url_for('settings'))
    else:
        flash('Form validation failed', 'danger')
        return redirect(url_for('settings'))


# Committee Addition Route
@app.route('/settings/add_committee', methods=['POST'])
def add_committee():
    committee_form = AddCommitteForm()

    if committee_form.validate_on_submit():

        full_name=committee_form.full_name.data,
        id_number=committee_form.id_number.data,
        phone_number=committee_form.phone_number.data,
        roles=committee_form.role.data
        role = user_datastore.find_or_create_role(roles)



        existing_committee_member = UserModel.query.filter_by(id_number=committee_form.id_number.data).first()
        if existing_committee_member:
            print('Committee member found')
            flash('Committee member with that id exists!', 'danger')

        new_committee_member
=user_datastore.create_user(full_name=full_name,id_number=id_number,phone_number=phone_number)
        user_datastore.add_role_to_user(new_committee_member, role)
        db.session.commit()
        flash('Committee member added successfully', 'success')
    else:
        flash('Form validation failed, please check your input', 'danger')
        return redirect(url_for('settings'))



#Umbrella Creation Route
@app.route('/settings/create_umbrella', methods=['POST'])
def create_umbrella():
    umbrella_form = UmbrellaForm()
    if umbrella_form.validate_on_submit():
        umbrella = UmbrellaModel.query.filter_by(name=umbrella_form.umbrella_name.data).first()
        if umbrella:
            flash('An umbrella with that name already exists', 'danger')



        else:
            new_umbrella = UmbrellaModel(
                name=umbrella_form.umbrella_name.data,
                location=umbrella_form.location.data,
                created_by=current_user.id
            )
            db.session.add(new_umbrella)
            db.session.commit()
            flash('Umbrella created successfully!', 'success')
        return redirect(url_for('settings'))
    else:
        flash('Form validation failed', 'danger')
        return redirect(url_for('settings'))


#Block Creation Route
@app.route('/settings/create_block', methods=['POST'])
def create_block():
    block_form = BlockForm()
    if block_form.validate_on_submit():
        block = BlockModel.query.filter_by(name=block_form.block_name.data).first()
        if block:
            flash('A block with that name already exists', 'danger')
        else:
            new_block = BlockModel(
                name=block_form.block_name.data,
                parent_umbrella_id=block_form.parent_umbrella.data,
                created_by=current_user.id
            )
            db.session.add(new_block)
            db.session.commit()
            flash('Block created successfully!', 'success')
        return redirect(url_for('settings'))
    else:
        flash('Form validation failed', 'danger')
        return redirect(url_for('settings'))


#Zone Creation Route
@app.route('/settings/create_zone', methods=['POST'])
def create_zone():
    zone_form = ZoneForm()
    if zone_form.validate_on_submit():
        zone = ZoneModel.query.filter_by(name=zone_form.zone_name.data).first()
        if zone:
            flash('A zone with that name already exists', 'danger')
        else:
            new_zone = ZoneModel(
                name=zone_form.zone_name.data,
                parent_block_id=zone_form.parent_block.data,
                created_by=current_user.id
            )
            db.session.add(new_zone)
            db.session.commit()
            flash('Zone created successfully!', 'success')
        return redirect(url_for('settings'))
    else:
        flash('Form validation failed', 'danger')
        return redirect(url_for('settings'))

#Member Creation Route
@app.route('/settings/add_member', methods=['POST'])
def add_member():
    member_form = AddMemberForm()
```

```python
    if member_form.validate_on_submit():
        user = UserModel.query.filter_by(id_number=member_form.id_number.data).first()
        if user:
            flash('User with that ID already exists', 'danger')
        else:
            new_user = UserModel(
                full_name=member_form.full_name.data,
                id_number=member_form.id_number.data,
                phone_number=member_form.phone_number.data,
                zone=member_form.member_zone.data,
                bank=member_form.bank.data,
                acc_number=member_form.acc_number.data
            )
            db.session.add(new_user)
            db.session.commit()
            flash('Member added successfully', 'success')
        return redirect(url_for('settings'))
    else:
        flash('Form validation failed', 'danger')
        return redirect(url_for('settings'))




@app.route('/', methods=['GET'])
def home():
    return render_template('index.html', title='TabPay | Home')

@app.route('/statistics', methods=['GET'])
@login_required
@roles_required('Umbrella_creator')
def statistics():
    # Get total number of members
    total_members = UserModel.query.count()

    # Get total number of blocks
    total_blocks = BlockModel.query.count()

    return render_template('statistics.html', title='Dashboard | Statistics',  total_members=total_members,
        total_blocks=total_blocks, user=current_user
    )


@app.route('/manage_contribution', methods=['GET'])
def manage_contribution():

    return render_template('manage_contribution.html', title='Dashboard | Manage Contributions')
```

```
@app.route('/host', methods=['GET'])
def host():
    return render_template('host.html', title='Dashboard | Host')

@app.route('/block_reports', methods=['GET', 'POST'])
def block_reports():
    return render_template('block_reports.html', title='Dashboard | Block Reports')


@app.route('/logout')
def logout():
    return redirect(url_for('home'))


with app.app_context():
    db.create_all()

    #Create roles
    user_datastore.find_or_create_role(name='Umbrella_creator',description='Account owner')
    user_datastore.find_or_create_role(name='Chairman',description='Head of block')
    user_datastore.find_or_create_role(name='Secretary',description='block secretary')
    user_datastore.find_or_create_role(name='Member',description='Regular member')



    #Create Admin
    if not user_datastore.find_user(email='enockbett427@gmail.com'):
        hashed_password = hash_password('123456')

user_datastore.create_user(email='enockbett427@gmail.com',password=hashed_password,roles=[user_datastor
e.find_role('Umbrella_creator')])
        db.session.commit()
        print('Umbrella_creator created successfully')

    #Create Chairman
    if not user_datastore.find_user(email='captain@example.com'):
        hashed_password = hash_password('123456')

user_datastore.create_user(email='captain@example.com',password=hashed_password,roles=[user_datastore.fi
nd_role('Chairman')])
        db.session.commit()

    #Create Secretary
    if not user_datastore.find_user(email='secretary@example.com'):
        hashed_password = hash_password('123456')
```

```
user_datastore.create_user(email='secretary@example.com',password=hashed_password,roles=[user_datastore
.find_role('Secretary')])
        db.session.commit()

    #Create Members
    if not user_datastore.find_user(email='member1@example.com'):
        hashed_password = hash_password('123456')

user_datastore.create_user(email='member1@example.com',password=hashed_password,roles=[user_datastore
.find_role('Member')])
        db.session.commit()


if __name__ == "__main__":
    app.run(debug=True,port=5001)
```

requirements.txt

```
aniso8601==9.0.1
argon2-cffi==23.1.0
argon2-cffi-bindings==21.2.0
bcrypt==4.2.0
blinker==1.8.2
cffi==1.17.1
click==8.1.7
dnspython==2.6.1
email_validator==2.2.0
Flask==3.0.3
Flask-Bcrypt==1.0.1
Flask-Login==0.6.3
Flask-Mailman==1.1.1
Flask-Principal==0.4.0
Flask-RESTful==0.3.10
Flask-Security==5.5.2
Flask-SQLAlchemy==3.1.1
Flask-WTF==1.2.1
greenlet==3.1.0
idna==3.10
importlib_resources==6.4.5
itsdangerous==2.2.0
Jinja2==3.1.4
MarkupSafe==2.1.5
passlib==1.7.4
psycopg2-binary==2.9.9
pycparser==2.22
```

```
pytz==2024.2

six==1.16.0

SQLAlchemy==2.0.34

typing_extensions==4.12.2

Werkzeug==3.0.4

WTForms==3.1.2
```