

(.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:
2. from sqlalchemy.exc import SQLAlchemyError
3. from werkzeug.exceptions import HTTPException
4. from flask_restful import Resource, marshal_with, abort, fields, reqparse
5. from app.extensions import db
6. from app.models.models import UserModel, CommunicationModel, UmbrellaModel, PaymentModel,
   BlockModel, ZoneModel
7.
8. # Argument parsers for different resources
9. umbrella_args = reqparse.RequestParser()
10. umbrella_args.add_argument('name', type=str, required=True, help='Umbrella Name is required')
11. umbrella_args.add_argument('location', type=str, required=True, help='Umbrella location is required')
12.
13. block_args = reqparse.RequestParser()
14. block_args.add_argument('name', type=str, required=True, help='Block Name is required')
15. block_args.add_argument('parent_umbrella_id', type=int, required=True, help='Parent Umbrella is
   required')
16.
17. zone_args = reqparse.RequestParser()
18. zone_args.add_argument('name', type=str, required=True, help='Zone Name is required')
19. zone_args.add_argument('parent_block_id', type=int, required=True, help='Parent Block is required')
20.
21. user_args = reqparse.RequestParser()
22. user_args.add_argument('email', type=str, required=True, help='Email is required')
23. user_args.add_argument('password', type=str, required=True, help='Password is required')
24.
25. communication_args = reqparse.RequestParser()
26. communication_args.add_argument('content', type=str, required=True, help='Content is required')
27. communication_args.add_argument('user_id', type=int, required=True, help='Author is required')
28.
29. payment_args = reqparse.RequestParser()
30. payment_args.add_argument('payer_id', type=int, required=True, help='Payer is required')
31. payment_args.add_argument('source_phone_number', type=str, required=True, help='Source phone
   number is required') # Changed to str
32. payment_args.add_argument('amount', type=float, required=True, help='Amount is required')
33.
34. # Fields for serialization
35. user_fields = {
36.     "id": fields.Integer,
37.     "full_name": fields.String,
38.     "email": fields.String,
39.     "password": fields.String,
40.     "id_number": fields.Integer,
```

```
41.     "phone_number": fields.String, # Changed to String
42.     "active": fields.Boolean,
43.     "zone_id": fields.Integer,
44.     "bank": fields.String,
45.     "acc_number": fields.String, # Changed to String
46.     "registered_at": fields.DateTime,
47.     "updated_at": fields.DateTime,
48.     "message": fields.String(attribute="author.full_name")
49. }
50.
51. communication_fields = {
52.     "id": fields.Integer,
53.     "content": fields.String,
54.     "user_id": fields.Integer,
55.     "created_at": fields.DateTime,
56.     "updated_at": fields.DateTime
57. }
58.
59. payment_fields = {
60.     "id": fields.Integer,
61.     "payer_id": fields.Integer,
62.     "amount": fields.Float,
63.     "payment_date": fields.DateTime,
64.     "mpesa_id": fields.String,
65.     "account_number": fields.String, # Changed to String
66.     "source_phone_number": fields.String, # Changed to String
67.     "transaction_status": fields.Boolean
68. }
69.
70. block_fields = {
71.     "id": fields.Integer,
72.     "name": fields.String,
73.     "umbrella_id": fields.Integer
74. }
75.
76. umbrella_fields = {
77.     "id": fields.Integer,
78.     "name": fields.String,
79.     "location": fields.String
80. }
81.
82. zone_fields = {
83.     "id": fields.Integer,
84.     "name": fields.String,
85.     "parent_block_id": fields.Integer
86. }
```

```
87.
88. class Users(Resource):
89.     @marshal_with(user_fields)
90.     def get(self):
91.         try:
92.             users = UserModel.query.all()
93.             return users, 200
94.
95.         except SQLAlchemyError as e:
96.             db.session.rollback()
97.             error_message = {"error": "Database error occurred", "details": str(e)}
98.             abort(500, message=error_message)
99.
100.        except HTTPException as e:
101.            error_message = {"error": "HTTP error occurred", "details": str(e)}
102.            abort(e.code, message=error_message)
103.
104.        except Exception as e:
105.            error_message = {"error": "Unexpected error occurred", "details": str(e)}
106.            abort(500, message=error_message)
107.
108.        finally:
109.            db.session.close()
110.
111.    @marshal_with(user_fields)
112.    def post(self):
113.        try:
114.            args = user_args.parse_args()
115.            existing_user = UserModel.query.filter_by(email=args['email']).first()
116.
117.            if existing_user:
118.                error_message = {"error": "User already exists"}
119.                abort(409, message=error_message)
120.
121.            new_user = UserModel(**args)
122.            db.session.add(new_user)
123.            db.session.commit()
124.            return new_user, 201
125.
126.        except SQLAlchemyError as e:
127.            db.session.rollback()
128.            error_message = {"error": "Database error occurred", "details": str(e)}
129.            abort(500, message=error_message)
130.
131.        except HTTPException as e:
132.            error_message = {"error": "HTTP error occurred", "details": str(e)}
```

```
133.     abort(e.code, message=error_message)
134.
135. except Exception as e:
136.     error_message = {"error": "Unexpected error occurred", "details": str(e)}
137.     abort(500, message=error_message)
138.
139. finally:
140.     db.session.close()
141.
142. class User(Resource):
143.     @marshal_with(user_fields)
144.     def get(self, id):
145.         try:
146.             user = UserModel.query.get_or_404(id)
147.             return user, 200
148.
149.         except SQLAlchemyError as e:
150.             error_message = {"error": "Database error occurred", "details": str(e)}
151.             abort(500, message=error_message)
152.
153.         except HTTPException as e:
154.             error_message = {"error": "HTTP error occurred", "details": str(e)}
155.             abort(e.code, message=error_message)
156.
157.     except Exception as e:
158.         error_message = {"error": "Unexpected error occurred", "details": str(e)}
159.         abort(500, message=error_message)
160.
161.     finally:
162.         db.session.close()
163.
164.     @marshal_with(user_fields)
165.     def patch(self, id):
166.         try:
167.             args = user_args.parse_args()
168.             existing_user = UserModel.query.get_or_404(id)
169.
170.             if existing_user:
171.                 for key, value in args.items():
172.                     setattr(existing_user, key, value)
173.                 db.session.commit()
174.                 return existing_user, 200
175.
176.             abort(404, message={"error": "User not found"})
177.
178.         except SQLAlchemyError as e:
```

```
179.         db.session.rollback()
180.         error_message = {"error": "Database error occurred", "details": str(e)}
181.         abort(500, message=error_message)
182.
183.     except HTTPException as e:
184.         error_message = {"error": "HTTP error occurred", "details": str(e)}
185.         abort(e.code, message=error_message)
186.
187.     except Exception as e:
188.         error_message = {"error": "Unexpected error occurred", "details": str(e)}
189.         abort(500, message=error_message)
190.
191.     finally:
192.         db.session.close()
193.
194. @marshal_with(user_fields)
195. def delete(self, id):
196.     try:
197.         existing_user = UserModel.query.get_or_404(id)
198.
199.         if existing_user:
200.             db.session.delete(existing_user)
201.             db.session.commit()
202.             users = UserModel.query.all()
203.             return users, 200
204.
205.         abort(404, message={"error": "User not found"})
206.
207.     except SQLAlchemyError as e:
208.         db.session.rollback()
209.         error_message = {"error": "Database error occurred", "details": str(e)}
210.         abort(500, message=error_message)
211.
212.     except HTTPException as e:
213.         error_message = {"error": "HTTP error occurred", "details": str(e)}
214.         abort(e.code, message=error_message)
215.
216.     except Exception as e:
217.         error_message = {"error": "Unexpected error occurred", "details": str(e)}
218.         abort(500, message=error_message)
219.
220.     finally:
221.         db.session.close()
222.
```

```
223. class Umbrellas(Resource):
224.     @marshal_with(umbrella_fields)
225.     def get(self):
226.         try:
227.             umbrellas = UmbrellaModel.query.all()
228.             return umbrellas, 200
229.
230.         except SQLAlchemyError as e:
231.             db.session.rollback()
232.             error_message = {"error": "Database error occurred", "details": str(e)}
233.             abort(500, message=error_message)
234.
235.         except HTTPException as e:
236.             error_message = {"error": "HTTP error occurred", "details": str(e)}
237.             abort(e.code, message=error_message)
238.
239.         except Exception as e:
240.             error_message = {"error": "Unexpected error occurred", "details": str(e)}
241.             abort(500, message=error_message)
242.
243.         finally:
244.             db.session.close()
245.
246.     @marshal_with(umbrella_fields)
247.     def post(self):
248.         try:
249.             args = umbrella_args.parse_args()
250.             new_umbrella = UmbrellaModel(**args)
251.             db.session.add(new_umbrella)
252.             db.session.commit()
253.             return new_umbrella, 201
254.
255.         except SQLAlchemyError as e:
256.             db.session.rollback()
257.             error_message = {"error": "Database error occurred", "details": str(e)}
258.             abort(500, message=error_message)
259.
260.         except HTTPException as e:
261.             error_message = {"error": "HTTP error occurred", "details": str(e)}
262.             abort(e.code, message=error_message)
263.
264.         except Exception as e:
265.             error_message = {"error": "Unexpected error occurred", "details": str(e)}
266.             abort(500, message=error_message)
267.
268.         finally:
```



```
269.     db.session.close()
270.
271. class Umbrella(Resource):
272.     @marshal_with(umbrella_fields)
273.     def get(self, id):
274.         try:
275.             umbrella = UmbrellaModel.query.get_or_404(id)
276.             return umbrella, 200
277.
278.         except SQLAlchemyError as e:
279.             error_message = {"error": "Database error occurred", "details": str(e)}
280.             abort(500, message=error_message)
281.
282.         except HTTPException as e:
283.             error_message = {"error": "HTTP error occurred", "details": str(e)}
284.             abort(e.code, message=error_message)
285.
286.         except Exception as e:
287.             error_message = {"error": "Unexpected error occurred", "details": str(e)}
288.             abort(500, message=error_message)
289.
290.         finally:
291.             db.session.close()
292.
293.     @marshal_with(umbrella_fields)
294.     def patch(self, id):
295.         try:
296.             args = umbrella_args.parse_args()
297.             umbrella = UmbrellaModel.query.get_or_404(id)
298.
299.             for key, value in args.items():
300.                 setattr(umbrella, key, value)
301.             db.session.commit()
302.             return umbrella, 200
303.
304.         except SQLAlchemyError as e:
305.             db.session.rollback()
306.             error_message = {"error": "Database error occurred", "details": str(e)}
307.             abort(500, message=error_message)
308.
309.         except HTTPException as e:
310.             error_message = {"error": "HTTP error occurred", "details": str(e)}
311.             abort(e.code, message=error_message)
312.
313.         except Exception as e:
314.             error_message = {"error": "Unexpected error occurred", "details": str(e)}
```

```
315.         abort(500, message=error_message)
316.
317.     finally:
318.         db.session.close()
319.
320. @marshal_with(umbrella_fields)
321. def delete(self, id):
322.     try:
323.         umbrella = UmbrellaModel.query.get_or_404(id)
324.         db.session.delete(umbrella)
325.         db.session.commit()
326.         umbrellas = UmbrellaModel.query.all()
327.         return umbrellas, 200
328.
329.     except SQLAlchemyError as e:
330.         db.session.rollback()
331.         error_message = {"error": "Database error occurred", "details": str(e)}
332.         abort(500, message=error_message)
333.
334.     except HTTPException as e:
335.         error_message = {"error": "HTTP error occurred", "details": str(e)}
336.         abort(e.code, message=error_message)
337.
338.     except Exception as e:
339.         error_message = {"error": "Unexpected error occurred", "details": str(e)}
340.         abort(500, message=error_message)
341.
342.     finally:
343.         db.session.close()
344.
345. class Communications(Resource):
346.     @marshal_with(communication_fields)
347.     def get(self):
348.         try:
349.             communications = CommunicationModel.query.all()
350.             return communications, 200
351.
352.         except SQLAlchemyError as e:
353.             db.session.rollback()
354.             error_message = {"error": "Database error occurred", "details": str(e)}
355.             abort(500, message=error_message)
```

```
356.
357.     except HTTPException as e:
358.         error_message = {"error": "HTTP error occurred", "details": str(e)}
359.         abort(e.code, message=error_message)
360.
361.     except Exception as e:
362.         error_message = {"error": "Unexpected error occurred", "details": str(e)}
363.         abort(500, message=error_message)
364.
365.     finally:
366.         db.session.close()
367.
368. @marshal_with(communication_fields)
369. def post(self):
370.     try:
371.         args = communication_args.parse_args()
372.         new_communication = CommunicationModel(**args)
373.         db.session.add(new_communication)
374.         db.session.commit()
375.         return new_communication, 201
376.
377.     except SQLAlchemyError as e:
378.         db.session.rollback()
379.         error_message = {"error": "Database error occurred", "details": str(e)}
380.         abort(500, message=error_message)
381.
382.     except HTTPException as e:
383.         error_message = {"error": "HTTP error occurred", "details": str(e)}
384.         abort(e.code, message=error_message)
385.
386.     except Exception as e:
387.         error_message = {"error": "Unexpected error occurred", "details": str(e)}
388.         abort(500, message=error_message)
389.
390.     finally:
391.         db.session.close()
392.
393. class Communication(Resource):
394.     @marshal_with(communication_fields)
395.     def get(self, id):
396.         try:
397.             communication = CommunicationModel.query.get_or_404(id)
398.             return communication, 200
399.
400.         except SQLAlchemyError as e:
401.             error_message = {"error": "Database error occurred", "details": str(e)}
```

```
402.         abort(500, message=error_message)
403.
404.     except HTTPException as e:
405.         error_message = {"error": "HTTP error occurred", "details": str(e)}
406.         abort(e.code, message=error_message)
407.
408.     except Exception as e:
409.         error_message = {"error": "Unexpected error occurred", "details": str(e)}
410.         abort(500, message=error_message)
411.
412.     finally:
413.         db.session.close()
414.
415. @marshal_with(communication_fields)
416. def patch(self, id):
417.     try:
418.         args = communication_args.parse_args()
419.         communication = CommunicationModel.query.get_or_404(id)
420.
421.         for key, value in args.items():
422.             setattr(communication, key, value)
423.         db.session.commit()
424.         return communication, 200
425.
426.     except SQLAlchemyError as e:
427.         db.session.rollback()
428.         error_message = {"error": "Database error occurred", "details": str(e)}
429.         abort(500, message=error_message)
430.
431.     except HTTPException as e:
432.         error_message = {"error": "HTTP error occurred", "details": str(e)}
433.         abort(e.code, message=error_message)
434.
435.     except Exception as e:
436.         error_message = {"error": "Unexpected error occurred", "details": str(e)}
437.         abort(500, message=error_message)
438.
439.     finally:
440.         db.session.close()
441.
442. @marshal_with(communication_fields)
443. def delete(self, id):
444.     try:
445.         communication = CommunicationModel.query.get_or_404(id)
446.         db.session.delete(communication)
447.         db.session.commit()
```

```
448.     communications = CommunicationModel.query.all()
449.     return communications, 200
450.
451. except SQLAlchemyError as e:
452.     db.session.rollback()
453.     error_message = {"error": "Database error occurred", "details": str(e)}
454.     abort(500, message=error_message)
455.
456. except HTTPException as e:
457.     error_message = {"error": "HTTP error occurred", "details": str(e)}
458.     abort(e.code, message=error_message)
459.
460. except Exception as e:
461.     error_message = {"error": "Unexpected error occurred", "details": str(e)}
462.     abort(500, message=error_message)
463.
464. finally:
465.     db.session.close()
466.
467. class Payments(Resource):
468.     @marshal_with(payment_fields)
469.     def get(self):
470.         try:
471.             payments = PaymentModel.query.all()
472.             return payments, 200
473.
474.         except SQLAlchemyError as e:
475.             db.session.rollback()
476.             error_message = {"error": "Database error occurred", "details": str(e)}
477.             abort(500, message=error_message)
478.
479.         except HTTPException as e:
480.             error_message = {"error": "HTTP error occurred", "details": str(e)}
481.             abort(e.code, message=error_message)
482.
483.         except Exception as e:
484.             error_message = {"error": "Unexpected error occurred", "details": str(e)}
485.             abort(500, message=error_message)
486.
487.         finally:
488.             db.session.close()
489.
490.     @marshal_with(payment_fields)
491.     def post(self):
```

```
492.     try:
493.         args = payment_args.parse_args()
494.         new_payment = PaymentModel(**args)
495.         db.session.add(new_payment)
496.         db.session.commit()
497.         return new_payment, 201
498.
499.     except SQLAlchemyError as e:
500.         db.session.rollback()
501.         error_message = {"error": "Database error occurred", "details": str(e)}
502.         abort(500, message=error_message)
503.
504.     except HTTPException as e:
505.         error_message = {"error": "HTTP error occurred", "details": str(e)}
506.         abort(e.code, message=error_message)
507.
508.     except Exception as e:
509.         error_message = {"error": "Unexpected error occurred", "details": str(e)}
510.         abort(500, message=error_message)
511.
512.     finally:
513.         db.session.close()
514.
515. class Payment(Resource):
516.     @marshal_with(payment_fields)
517.     def get(self, id):
518.         try:
519.             payment = PaymentModel.query.get_or_404(id)
520.             return payment, 200
521.
522.         except SQLAlchemyError as e:
523.             error_message = {"error": "Database error occurred", "details": str(e)}
524.             abort(500, message=error_message)
525.
526.         except HTTPException as e:
527.             error_message = {"error": "HTTP error occurred", "details": str(e)}
528.             abort(e.code, message=error_message)
529.
530.         except Exception as e:
531.             error_message = {"error": "Unexpected error occurred", "details": str(e)}
532.             abort(500, message=error_message)
533.
534.         finally:
535.             db.session.close()
536.
537.     @marshal_with(payment_fields)
```

```
538. def patch(self, id):
539.     try:
540.         args = payment_args.parse_args()
541.         payment = PaymentModel.query.get_or_404(id)
542.
543.         for key, value in args.items():
544.             setattr(payment, key, value)
545.         db.session.commit()
546.         return payment, 200
547.
548.     except SQLAlchemyError as e:
549.         db.session.rollback()
550.         error_message = {"error": "Database error occurred", "details": str(e)}
551.         abort(500, message=error_message)
552.
553.     except HTTPException as e:
554.         error_message = {"error": "HTTP error occurred", "details": str(e)}
555.         abort(e.code, message=error_message)
556.
557.     except Exception as e:
558.         error_message = {"error": "Unexpected error occurred", "details": str(e)}
559.         abort(500, message=error_message)
560.
561.     finally:
562.         db.session.close()
563.
564. @marshal_with(payment_fields)
565. def delete(self, id):
566.     try:
567.         payment = PaymentModel.query.get_or_404(id)
568.         db.session.delete(payment)
569.         db.session.commit()
570.         payments = PaymentModel.query.all()
571.         return payments, 200
572.
573.     except SQLAlchemyError as e:
574.         db.session.rollback()
575.         error_message = {"error": "Database error occurred", "details": str(e)}
576.         abort(500, message=error_message)
577.
578.     except HTTPException as e:
579.         error_message = {"error": "HTTP error occurred", "details": str(e)}
580.         abort(e.code, message=error_message)
581.
582.     except Exception as e:
583.         error_message = {"error": "Unexpected error occurred", "details": str(e)}
```

```
584.         abort(500, message=error_message)
585.
586.     finally:
587.         db.session.close()
588.
589. class Blocks(Resource):
590.     @marshal_with(block_fields)
591.     def get(self):
592.         try:
593.             blocks = BlockModel.query.all()
594.             return blocks, 200
595.
596.         except SQLAlchemyError as e:
597.             db.session.rollback()
598.             error_message = {"error": "Database error occurred", "details": str(e)}
599.             abort(500, message=error_message)
600.
601.         except HTTPException as e:
602.             error_message = {"error": "HTTP error occurred", "details": str(e)}
603.             abort(e.code, message=error_message)
604.
605.         except Exception as e:
606.             error_message = {"error": "Unexpected error occurred", "details": str(e)}
607.             abort(500, message=error_message)
608.
609.     finally:
610.         db.session.close()
611.
612.     @marshal_with(block_fields)
613.     def post(self):
614.         try:
615.             args = block_args.parse_args()
616.             new_block = BlockModel(**args)
617.             db.session.add(new_block)
618.             db.session.commit()
619.             return new_block, 201
620.
621.         except SQLAlchemyError as e:
622.             db.session.rollback()
623.             error_message = {"error": "Database error occurred", "details": str(e)}
624.             abort(500, message=error_message)
625.
626.         except HTTPException as e:
```



```
627.     error_message = {"error": "HTTP error occurred", "details": str(e)}
628.     abort(e.code, message=error_message)
629.
630. except Exception as e:
631.     error_message = {"error": "Unexpected error occurred", "details": str(e)}
632.     abort(500, message=error_message)
633.
634. finally:
635.     db.session.close()
636.
637. class Block(Resource):
638.     @marshal_with(block_fields)
639.     def get(self, id):
640.         try:
641.             block = BlockModel.query.get_or_404(id)
642.             return block, 200
643.
644.         except SQLAlchemyError as e:
645.             error_message = {"error": "Database error occurred", "details": str(e)}
646.             abort(500, message=error_message)
647.
648.         except HTTPException as e:
649.             error_message = {"error": "HTTP error occurred", "details": str(e)}
650.             abort(e.code, message=error_message)
651.
652.     except Exception as e:
653.         error_message = {"error": "Unexpected error occurred", "details": str(e)}
654.         abort(500, message=error_message)
655.
656.     finally:
657.         db.session.close()
658.
659.     @marshal_with(block_fields)
660.     def patch(self, id):
661.         try:
662.             args = block_args.parse_args()
663.             block = BlockModel.query.get_or_404(id)
664.
665.             for key, value in args.items():
666.                 setattr(block, key, value)
667.             db.session.commit()
668.             return block, 200
669.
670.         except SQLAlchemyError as e:
671.             db.session.rollback()
672.             error_message = {"error": "Database error occurred", "details": str(e)}
```

```
673.         abort(500, message=error_message)
674.
675.     except HTTPException as e:
676.         error_message = {"error": "HTTP error occurred", "details": str(e)}
677.         abort(e.code, message=error_message)
678.
679.     except Exception as e:
680.         error_message = {"error": "Unexpected error occurred", "details": str(e)}
681.         abort(500, message=error_message)
682.
683.     finally:
684.         db.session.close()
685.
686. @marshal_with(block_fields)
687. def delete(self, id):
688.     try:
689.         block = BlockModel.query.get_or_404(id)
690.         db.session.delete(block)
691.         db.session.commit()
692.         blocks = BlockModel.query.all()
693.         return blocks, 200
694.
695.     except SQLAlchemyError as e:
696.         db.session.rollback()
697.         error_message = {"error": "Database error occurred", "details": str(e)}
698.         abort(500, message=error_message)
699.
700.     except HTTPException as e:
701.         error_message = {"error": "HTTP error occurred", "details": str(e)}
702.         abort(e.code, message=error_message)
703.
704.     except Exception as e:
705.         error_message = {"error": "Unexpected error occurred", "details": str(e)}
706.         abort(500, message=error_message)
707.
708.     finally:
709.         db.session.close()
710.
711. class Zones(Resource):
712.     @marshal_with(zone_fields)
713.     def get(self):
714.         try:
715.             zones = ZoneModel.query.all()
```

```
716.         return zones, 200
717.
718.     except SQLAlchemyError as e:
719.         db.session.rollback()
720.         error_message = {"error": "Database error occurred", "details": str(e)}
721.         abort(500, message=error_message)
722.
723.     except HTTPException as e:
724.         error_message = {"error": "HTTP error occurred", "details": str(e)}
725.         abort(e.code, message=error_message)
726.
727.     except Exception as e:
728.         error_message = {"error": "Unexpected error occurred", "details": str(e)}
729.         abort(500, message=error_message)
730.
731.     finally:
732.         db.session.close()
733.
734. @marshal_with(zone_fields)
735. def post(self):
736.     try:
737.         args = zone_args.parse_args()
738.         new_zone = ZoneModel(**args)
739.         db.session.add(new_zone)
740.         db.session.commit()
741.         return new_zone, 201
742.
743.     except SQLAlchemyError as e:
744.         db.session.rollback()
745.         error_message = {"error": "Database error occurred", "details": str(e)}
746.         abort(500, message=error_message)
747.
748.     except HTTPException as e:
749.         error_message = {"error": "HTTP error occurred", "details": str(e)}
750.         abort(e.code, message=error_message)
751.
752.     except Exception as e:
753.         error_message = {"error": "Unexpected error occurred", "details": str(e)}
754.         abort(500, message=error_message)
755.
756.     finally:
757.         db.session.close()
758.
759. class Zone(Resource):
760.     @marshal_with(zone_fields)
761.     def get(self, id):
```

```
762.     try:
763.         zone = ZoneModel.query.get_or_404(id)
764.         return zone, 200
765.
766.     except SQLAlchemyError as e:
767.         error_message = {"error": "Database error occurred", "details": str(e)}
768.         abort(500, message=error_message)
769.
770.     except HTTPException as e:
771.         error_message = {"error": "HTTP error occurred", "details": str(e)}
772.         abort(e.code, message=error_message)
773.
774.     except Exception as e:
775.         error_message = {"error": "Unexpected error occurred", "details": str(e)}
776.         abort(500, message=error_message)
777.
778.     finally:
779.         db.session.close()
780.
781. @marshal_with(zone_fields)
782. def patch(self, id):
783.     try:
784.         args = zone_args.parse_args()
785.         zone = ZoneModel.query.get_or_404(id)
786.
787.         for key, value in args.items():
788.             setattr(zone, key, value)
789.         db.session.commit()
790.         return zone, 200
791.
792.     except SQLAlchemyError as e:
793.         db.session.rollback()
794.         error_message = {"error": "Database error occurred", "details": str(e)}
795.         abort(500, message=error_message)
796.
797.     except HTTPException as e:
798.         error_message = {"error": "HTTP error occurred", "details": str(e)}
799.         abort(e.code, message=error_message)
800.
801.     except Exception as e:
802.         error_message = {"error": "Unexpected error occurred", "details": str(e)}
803.         abort(500, message=error_message)
804.
805.     finally:
806.         db.session.close()
807.
```

```

808. @marshal_with(zone_fields)
809. def delete(self, id):
810.     try:
811.         zone = ZoneModel.query.get_or_404(id)
812.         db.session.delete(zone)
813.         db.session.commit()
814.         zones = ZoneModel.query.all()
815.         return zones, 200
816.
817.     except SQLAlchemyError as e:
818.         db.session.rollback()
819.         error_message = {"error": "Database error occurred", "details": str(e)}
820.         abort(500, message=error_message)
821.
822.     except HTTPException as e:
823.         error_message = {"error": "HTTP error occurred", "details": str(e)}
824.         abort(e.code, message=error_message)
825.
826.     except Exception as e:
827.         error_message = {"error": "Unexpected error occurred", "details": str(e)}
828.         abort(500, message=error_message)
829.
830.     finally:
831.         db.session.close()
832.

```

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': 'xxxxxxx'})

```

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':'xxxxxxx'})
    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':'xxxxxxx'})
    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 AddCommitteeForm(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':'xxxxxxx'})
    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)

```



```

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}>"

```

```

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()

```

```
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()

```

```

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_datastore.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.find_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
psycpg2-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