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. # Argument parsers for different resources
8. umbrella\_args = reqparse.RequestParser()
9. umbrella\_args.add\_argument('name', type=str, required=True, help='Umbrella Name is required')
10. umbrella\_args.add\_argument('location', type=str, required=True, help='Umbrella location is required')
11. block\_args = reqparse.RequestParser()
12. block\_args.add\_argument('name', type=str, required=True, help='Block Name is required')
13. block\_args.add\_argument('parent\_umbrella\_id', type=int, required=True, help='Parent Umbrella is required')
14. zone\_args = reqparse.RequestParser()
15. zone\_args.add\_argument('name', type=str, required=True, help='Zone Name is required')
16. zone\_args.add\_argument('parent\_block\_id', type=int, required=True, help='Parent Block is required')
17. user\_args = reqparse.RequestParser()
18. user\_args.add\_argument('email', type=str, required=True, help='Email is required')
19. user\_args.add\_argument('password', type=str, required=True, help='Password is required')
20. communication\_args = reqparse.RequestParser()
21. communication\_args.add\_argument('content', type=str, required=True, help='Content is required')
22. communication\_args.add\_argument('user\_id', type=int, required=True, help='Author is required')
23. payment\_args = reqparse.RequestParser()
24. payment\_args.add\_argument('payer\_id', type=int, required=True, help='Payer is required')
25. payment\_args.add\_argument('source\_phone\_number', type=str, required=True, help='Source phone number is required') # Changed to str
26. payment\_args.add\_argument('amount', type=float, required=True, help='Amount is required')
27. # Fields for serialization
28. user\_fields = {
29. "id": fields.Integer,
30. "full\_name": fields.String,
31. "email": fields.String,
32. "password": fields.String,
33. "id\_number": fields.Integer,
34. "phone\_number": fields.String, # Changed to String
35. "active": fields.Boolean,
36. "zone\_id": fields.Integer,
37. "bank": fields.String,
38. "acc\_number": fields.String, # Changed to String
39. "registered\_at": fields.DateTime,
40. "updated\_at": fields.DateTime,
41. "message": fields.String(attribute="author.full\_name")
42. }
43. communication\_fields = {
44. "id": fields.Integer,
45. "content": fields.String,
46. "user\_id": fields.Integer,
47. "created\_at": fields.DateTime,
48. "updated\_at": fields.DateTime
49. }
50. payment\_fields = {
51. "id": fields.Integer,
52. "payer\_id": fields.Integer,
53. "amount": fields.Float,
54. "payment\_date": fields.DateTime,
55. "mpesa\_id": fields.String,
56. "account\_number": fields.String, # Changed to String
57. "source\_phone\_number": fields.String, # Changed to String
58. "transaction\_status": fields.Boolean
59. }
60. block\_fields = {
61. "id": fields.Integer,
62. "name": fields.String,
63. "umbrella\_id": fields.Integer
64. }
65. umbrella\_fields = {
66. "id": fields.Integer,
67. "name": fields.String,
68. "location": fields.String
69. }
70. zone\_fields = {
71. "id": fields.Integer,
72. "name": fields.String,
73. "parent\_block\_id": fields.Integer
74. }
75. class Users(Resource):
76. @marshal\_with(user\_fields)
77. def get(self):
78. try:
79. users = UserModel.query.all()
80. return users, 200
82. except SQLAlchemyError as e:
83. db.session.rollback()
84. error\_message = {"error": "Database error occurred", "details": str(e)}
85. abort(500, message=error\_message)
87. except HTTPException as e:
88. error\_message = {"error": "HTTP error occurred", "details": str(e)}
89. abort(e.code, message=error\_message)
91. except Exception as e:
92. error\_message = {"error": "Unexpected error occurred", "details": str(e)}
93. abort(500, message=error\_message)
95. finally:
96. db.session.close()
97. @marshal\_with(user\_fields)
98. def post(self):
99. try:
100. args = user\_args.parse\_args()
101. existing\_user = UserModel.query.filter\_by(email=args['email']).first()
103. if existing\_user:
104. error\_message = {"error": "User already exists"}
105. abort(409, message=error\_message)
107. new\_user = UserModel(\*\*args)
108. db.session.add(new\_user)
109. db.session.commit()
110. return new\_user, 201
112. except SQLAlchemyError as e:
113. db.session.rollback()
114. error\_message = {"error": "Database error occurred", "details": str(e)}
115. abort(500, message=error\_message)
117. except HTTPException as e:
118. error\_message = {"error": "HTTP error occurred", "details": str(e)}
119. abort(e.code, message=error\_message)
121. except Exception as e:
122. error\_message = {"error": "Unexpected error occurred", "details": str(e)}
123. abort(500, message=error\_message)
125. finally:
126. db.session.close()
127. class User(Resource):
128. @marshal\_with(user\_fields)
129. def get(self, id):
130. try:
131. user = UserModel.query.get\_or\_404(id)
132. return user, 200
134. except SQLAlchemyError as e:
135. error\_message = {"error": "Database error occurred", "details": str(e)}
136. abort(500, message=error\_message)
138. except HTTPException as e:
139. error\_message = {"error": "HTTP error occurred", "details": str(e)}
140. abort(e.code, message=error\_message)
142. except Exception as e:
143. error\_message = {"error": "Unexpected error occurred", "details": str(e)}
144. abort(500, message=error\_message)
146. finally:
147. db.session.close()
148. @marshal\_with(user\_fields)
149. def patch(self, id):
150. try:
151. args = user\_args.parse\_args()
152. existing\_user = UserModel.query.get\_or\_404(id)
154. if existing\_user:
155. for key, value in args.items():
156. setattr(existing\_user, key, value)
157. db.session.commit()
158. return existing\_user, 200
160. abort(404, message={"error": "User not found"})
162. except SQLAlchemyError as e:
163. db.session.rollback()
164. error\_message = {"error": "Database error occurred", "details": str(e)}
165. abort(500, message=error\_message)
167. except HTTPException as e:
168. error\_message = {"error": "HTTP error occurred", "details": str(e)}
169. abort(e.code, message=error\_message)
171. except Exception as e:
172. error\_message = {"error": "Unexpected error occurred", "details": str(e)}
173. abort(500, message=error\_message)
175. finally:
176. db.session.close()
177. @marshal\_with(user\_fields)
178. def delete(self, id):
179. try:
180. existing\_user = UserModel.query.get\_or\_404(id)
182. if existing\_user:
183. db.session.delete(existing\_user)
184. db.session.commit()
185. users = UserModel.query.all()
186. return users, 200
188. abort(404, message={"error": "User not found"})
190. except SQLAlchemyError as e:
191. db.session.rollback()
192. error\_message = {"error": "Database error occurred", "details": str(e)}
193. abort(500, message=error\_message)
195. except HTTPException as e:
196. error\_message = {"error": "HTTP error occurred", "details": str(e)}
197. abort(e.code, message=error\_message)
199. except Exception as e:
200. error\_message = {"error": "Unexpected error occurred", "details": str(e)}
201. abort(500, message=error\_message)
203. finally:
204. db.session.close()
206. class Umbrellas(Resource):
207. @marshal\_with(umbrella\_fields)
208. def get(self):
209. try:
210. umbrellas = UmbrellaModel.query.all()
211. return umbrellas, 200
213. except SQLAlchemyError as e:
214. db.session.rollback()
215. error\_message = {"error": "Database error occurred", "details": str(e)}
216. abort(500, message=error\_message)
218. except HTTPException as e:
219. error\_message = {"error": "HTTP error occurred", "details": str(e)}
220. abort(e.code, message=error\_message)
222. except Exception as e:
223. error\_message = {"error": "Unexpected error occurred", "details": str(e)}
224. abort(500, message=error\_message)
226. finally:
227. db.session.close()
228. @marshal\_with(umbrella\_fields)
229. def post(self):
230. try:
231. args = umbrella\_args.parse\_args()
232. new\_umbrella = UmbrellaModel(\*\*args)
233. db.session.add(new\_umbrella)
234. db.session.commit()
235. return new\_umbrella, 201
237. except SQLAlchemyError as e:
238. db.session.rollback()
239. error\_message = {"error": "Database error occurred", "details": str(e)}
240. abort(500, message=error\_message)
242. except HTTPException as e:
243. error\_message = {"error": "HTTP error occurred", "details": str(e)}
244. abort(e.code, message=error\_message)
246. except Exception as e:
247. error\_message = {"error": "Unexpected error occurred", "details": str(e)}
248. abort(500, message=error\_message)
250. finally:
251. db.session.close()
252. class Umbrella(Resource):
253. @marshal\_with(umbrella\_fields)
254. def get(self, id):
255. try:
256. umbrella = UmbrellaModel.query.get\_or\_404(id)
257. return umbrella, 200
259. except SQLAlchemyError as e:
260. error\_message = {"error": "Database error occurred", "details": str(e)}
261. abort(500, message=error\_message)
263. except HTTPException as e:
264. error\_message = {"error": "HTTP error occurred", "details": str(e)}
265. abort(e.code, message=error\_message)
267. except Exception as e:
268. error\_message = {"error": "Unexpected error occurred", "details": str(e)}
269. abort(500, message=error\_message)
271. finally:
272. db.session.close()
273. @marshal\_with(umbrella\_fields)
274. def patch(self, id):
275. try:
276. args = umbrella\_args.parse\_args()
277. umbrella = UmbrellaModel.query.get\_or\_404(id)
279. for key, value in args.items():
280. setattr(umbrella, key, value)
281. db.session.commit()
282. return umbrella, 200
284. except SQLAlchemyError as e:
285. db.session.rollback()
286. error\_message = {"error": "Database error occurred", "details": str(e)}
287. abort(500, message=error\_message)
289. except HTTPException as e:
290. error\_message = {"error": "HTTP error occurred", "details": str(e)}
291. abort(e.code, message=error\_message)
293. except Exception as e:
294. error\_message = {"error": "Unexpected error occurred", "details": str(e)}
295. abort(500, message=error\_message)
297. finally:
298. db.session.close()
299. @marshal\_with(umbrella\_fields)
300. def delete(self, id):
301. try:
302. umbrella = UmbrellaModel.query.get\_or\_404(id)
303. db.session.delete(umbrella)
304. db.session.commit()
305. umbrellas = UmbrellaModel.query.all()
306. return umbrellas, 200
308. except SQLAlchemyError as e:
309. db.session.rollback()
310. error\_message = {"error": "Database error occurred", "details": str(e)}
311. abort(500, message=error\_message)
313. except HTTPException as e:
314. error\_message = {"error": "HTTP error occurred", "details": str(e)}
315. abort(e.code, message=error\_message)
317. except Exception as e:
318. error\_message = {"error": "Unexpected error occurred", "details": str(e)}
319. abort(500, message=error\_message)
321. finally:
322. db.session.close()
324. class Communications(Resource):
325. @marshal\_with(communication\_fields)
326. def get(self):
327. try:
328. communications = CommunicationModel.query.all()
329. return communications, 200
331. except SQLAlchemyError as e:
332. db.session.rollback()
333. error\_message = {"error": "Database error occurred", "details": str(e)}
334. abort(500, message=error\_message)
336. except HTTPException as e:
337. error\_message = {"error": "HTTP error occurred", "details": str(e)}
338. abort(e.code, message=error\_message)
340. except Exception as e:
341. error\_message = {"error": "Unexpected error occurred", "details": str(e)}
342. abort(500, message=error\_message)
344. finally:
345. db.session.close()
346. @marshal\_with(communication\_fields)
347. def post(self):
348. try:
349. args = communication\_args.parse\_args()
350. new\_communication = CommunicationModel(\*\*args)
351. db.session.add(new\_communication)
352. db.session.commit()
353. return new\_communication, 201
355. except SQLAlchemyError as e:
356. db.session.rollback()
357. error\_message = {"error": "Database error occurred", "details": str(e)}
358. abort(500, message=error\_message)
360. except HTTPException as e:
361. error\_message = {"error": "HTTP error occurred", "details": str(e)}
362. abort(e.code, message=error\_message)
364. except Exception as e:
365. error\_message = {"error": "Unexpected error occurred", "details": str(e)}
366. abort(500, message=error\_message)
368. finally:
369. db.session.close()
370. class Communication(Resource):
371. @marshal\_with(communication\_fields)
372. def get(self, id):
373. try:
374. communication = CommunicationModel.query.get\_or\_404(id)
375. return communication, 200
377. except SQLAlchemyError as e:
378. error\_message = {"error": "Database error occurred", "details": str(e)}
379. abort(500, message=error\_message)
381. except HTTPException as e:
382. error\_message = {"error": "HTTP error occurred", "details": str(e)}
383. abort(e.code, message=error\_message)
385. except Exception as e:
386. error\_message = {"error": "Unexpected error occurred", "details": str(e)}
387. abort(500, message=error\_message)
389. finally:
390. db.session.close()
391. @marshal\_with(communication\_fields)
392. def patch(self, id):
393. try:
394. args = communication\_args.parse\_args()
395. communication = CommunicationModel.query.get\_or\_404(id)
397. for key, value in args.items():
398. setattr(communication, key, value)
399. db.session.commit()
400. return communication, 200
402. except SQLAlchemyError as e:
403. db.session.rollback()
404. error\_message = {"error": "Database error occurred", "details": str(e)}
405. abort(500, message=error\_message)
407. except HTTPException as e:
408. error\_message = {"error": "HTTP error occurred", "details": str(e)}
409. abort(e.code, message=error\_message)
411. except Exception as e:
412. error\_message = {"error": "Unexpected error occurred", "details": str(e)}
413. abort(500, message=error\_message)
415. finally:
416. db.session.close()
417. @marshal\_with(communication\_fields)
418. def delete(self, id):
419. try:
420. communication = CommunicationModel.query.get\_or\_404(id)
421. db.session.delete(communication)
422. db.session.commit()
423. communications = CommunicationModel.query.all()
424. return communications, 200
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)
431. except HTTPException as e:
432. error\_message = {"error": "HTTP error occurred", "details": str(e)}
433. abort(e.code, message=error\_message)
435. except Exception as e:
436. error\_message = {"error": "Unexpected error occurred", "details": str(e)}
437. abort(500, message=error\_message)
439. finally:
440. db.session.close()
442. class Payments(Resource):
443. @marshal\_with(payment\_fields)
444. def get(self):
445. try:
446. payments = PaymentModel.query.all()
447. return payments, 200
449. except SQLAlchemyError as e:
450. db.session.rollback()
451. error\_message = {"error": "Database error occurred", "details": str(e)}
452. abort(500, message=error\_message)
454. except HTTPException as e:
455. error\_message = {"error": "HTTP error occurred", "details": str(e)}
456. abort(e.code, message=error\_message)
458. except Exception as e:
459. error\_message = {"error": "Unexpected error occurred", "details": str(e)}
460. abort(500, message=error\_message)
462. finally:
463. db.session.close()
464. @marshal\_with(payment\_fields)
465. def post(self):
466. try:
467. args = payment\_args.parse\_args()
468. new\_payment = PaymentModel(\*\*args)
469. db.session.add(new\_payment)
470. db.session.commit()
471. return new\_payment, 201
473. except SQLAlchemyError as e:
474. db.session.rollback()
475. error\_message = {"error": "Database error occurred", "details": str(e)}
476. abort(500, message=error\_message)
478. except HTTPException as e:
479. error\_message = {"error": "HTTP error occurred", "details": str(e)}
480. abort(e.code, message=error\_message)
482. except Exception as e:
483. error\_message = {"error": "Unexpected error occurred", "details": str(e)}
484. abort(500, message=error\_message)
486. finally:
487. db.session.close()
488. class Payment(Resource):
489. @marshal\_with(payment\_fields)
490. def get(self, id):
491. try:
492. payment = PaymentModel.query.get\_or\_404(id)
493. return payment, 200
495. except SQLAlchemyError as e:
496. error\_message = {"error": "Database error occurred", "details": str(e)}
497. abort(500, message=error\_message)
499. except HTTPException as e:
500. error\_message = {"error": "HTTP error occurred", "details": str(e)}
501. abort(e.code, message=error\_message)
503. except Exception as e:
504. error\_message = {"error": "Unexpected error occurred", "details": str(e)}
505. abort(500, message=error\_message)
507. finally:
508. db.session.close()
509. @marshal\_with(payment\_fields)
510. def patch(self, id):
511. try:
512. args = payment\_args.parse\_args()
513. payment = PaymentModel.query.get\_or\_404(id)
515. for key, value in args.items():
516. setattr(payment, key, value)
517. db.session.commit()
518. return payment, 200
520. except SQLAlchemyError as e:
521. db.session.rollback()
522. error\_message = {"error": "Database error occurred", "details": str(e)}
523. abort(500, message=error\_message)
525. except HTTPException as e:
526. error\_message = {"error": "HTTP error occurred", "details": str(e)}
527. abort(e.code, message=error\_message)
529. except Exception as e:
530. error\_message = {"error": "Unexpected error occurred", "details": str(e)}
531. abort(500, message=error\_message)
533. finally:
534. db.session.close()
535. @marshal\_with(payment\_fields)
536. def delete(self, id):
537. try:
538. payment = PaymentModel.query.get\_or\_404(id)
539. db.session.delete(payment)
540. db.session.commit()
541. payments = PaymentModel.query.all()
542. return payments, 200
544. except SQLAlchemyError as e:
545. db.session.rollback()
546. error\_message = {"error": "Database error occurred", "details": str(e)}
547. abort(500, message=error\_message)
549. except HTTPException as e:
550. error\_message = {"error": "HTTP error occurred", "details": str(e)}
551. abort(e.code, message=error\_message)
553. except Exception as e:
554. error\_message = {"error": "Unexpected error occurred", "details": str(e)}
555. abort(500, message=error\_message)
557. finally:
558. db.session.close()
560. class Blocks(Resource):
561. @marshal\_with(block\_fields)
562. def get(self):
563. try:
564. blocks = BlockModel.query.all()
565. return blocks, 200
567. except SQLAlchemyError as e:
568. db.session.rollback()
569. error\_message = {"error": "Database error occurred", "details": str(e)}
570. abort(500, message=error\_message)
572. except HTTPException as e:
573. error\_message = {"error": "HTTP error occurred", "details": str(e)}
574. abort(e.code, message=error\_message)
576. except Exception as e:
577. error\_message = {"error": "Unexpected error occurred", "details": str(e)}
578. abort(500, message=error\_message)
580. finally:
581. db.session.close()
582. @marshal\_with(block\_fields)
583. def post(self):
584. try:
585. args = block\_args.parse\_args()
586. new\_block = BlockModel(\*\*args)
587. db.session.add(new\_block)
588. db.session.commit()
589. return new\_block, 201
591. except SQLAlchemyError as e:
592. db.session.rollback()
593. error\_message = {"error": "Database error occurred", "details": str(e)}
594. abort(500, message=error\_message)
596. except HTTPException as e:
597. error\_message = {"error": "HTTP error occurred", "details": str(e)}
598. abort(e.code, message=error\_message)
600. except Exception as e:
601. error\_message = {"error": "Unexpected error occurred", "details": str(e)}
602. abort(500, message=error\_message)
604. finally:
605. db.session.close()
606. class Block(Resource):
607. @marshal\_with(block\_fields)
608. def get(self, id):
609. try:
610. block = BlockModel.query.get\_or\_404(id)
611. return block, 200
613. except SQLAlchemyError as e:
614. error\_message = {"error": "Database error occurred", "details": str(e)}
615. abort(500, message=error\_message)
617. except HTTPException as e:
618. error\_message = {"error": "HTTP error occurred", "details": str(e)}
619. abort(e.code, message=error\_message)
621. except Exception as e:
622. error\_message = {"error": "Unexpected error occurred", "details": str(e)}
623. abort(500, message=error\_message)
625. finally:
626. db.session.close()
627. @marshal\_with(block\_fields)
628. def patch(self, id):
629. try:
630. args = block\_args.parse\_args()
631. block = BlockModel.query.get\_or\_404(id)
633. for key, value in args.items():
634. setattr(block, key, value)
635. db.session.commit()
636. return block, 200
638. except SQLAlchemyError as e:
639. db.session.rollback()
640. error\_message = {"error": "Database error occurred", "details": str(e)}
641. abort(500, message=error\_message)
643. except HTTPException as e:
644. error\_message = {"error": "HTTP error occurred", "details": str(e)}
645. abort(e.code, message=error\_message)
647. except Exception as e:
648. error\_message = {"error": "Unexpected error occurred", "details": str(e)}
649. abort(500, message=error\_message)
651. finally:
652. db.session.close()
653. @marshal\_with(block\_fields)
654. def delete(self, id):
655. try:
656. block = BlockModel.query.get\_or\_404(id)
657. db.session.delete(block)
658. db.session.commit()
659. blocks = BlockModel.query.all()
660. return blocks, 200
662. except SQLAlchemyError as e:
663. db.session.rollback()
664. error\_message = {"error": "Database error occurred", "details": str(e)}
665. abort(500, message=error\_message)
667. except HTTPException as e:
668. error\_message = {"error": "HTTP error occurred", "details": str(e)}
669. abort(e.code, message=error\_message)
671. except Exception as e:
672. error\_message = {"error": "Unexpected error occurred", "details": str(e)}
673. abort(500, message=error\_message)
675. finally:
676. db.session.close()
678. class Zones(Resource):
679. @marshal\_with(zone\_fields)
680. def get(self):
681. try:
682. zones = ZoneModel.query.all()
683. return zones, 200
685. except SQLAlchemyError as e:
686. db.session.rollback()
687. error\_message = {"error": "Database error occurred", "details": str(e)}
688. abort(500, message=error\_message)
690. except HTTPException as e:
691. error\_message = {"error": "HTTP error occurred", "details": str(e)}
692. abort(e.code, message=error\_message)
694. except Exception as e:
695. error\_message = {"error": "Unexpected error occurred", "details": str(e)}
696. abort(500, message=error\_message)
698. finally:
699. db.session.close()
700. @marshal\_with(zone\_fields)
701. def post(self):
702. try:
703. args = zone\_args.parse\_args()
704. new\_zone = ZoneModel(\*\*args)
705. db.session.add(new\_zone)
706. db.session.commit()
707. return new\_zone, 201
709. except SQLAlchemyError as e:
710. db.session.rollback()
711. error\_message = {"error": "Database error occurred", "details": str(e)}
712. abort(500, message=error\_message)
714. except HTTPException as e:
715. error\_message = {"error": "HTTP error occurred", "details": str(e)}
716. abort(e.code, message=error\_message)
718. except Exception as e:
719. error\_message = {"error": "Unexpected error occurred", "details": str(e)}
720. abort(500, message=error\_message)
722. finally:
723. db.session.close()
724. class Zone(Resource):
725. @marshal\_with(zone\_fields)
726. def get(self, id):
727. try:
728. zone = ZoneModel.query.get\_or\_404(id)
729. return zone, 200
731. except SQLAlchemyError as e:
732. error\_message = {"error": "Database error occurred", "details": str(e)}
733. abort(500, message=error\_message)
735. except HTTPException as e:
736. error\_message = {"error": "HTTP error occurred", "details": str(e)}
737. abort(e.code, message=error\_message)
739. except Exception as e:
740. error\_message = {"error": "Unexpected error occurred", "details": str(e)}
741. abort(500, message=error\_message)
743. finally:
744. db.session.close()
745. @marshal\_with(zone\_fields)
746. def patch(self, id):
747. try:
748. args = zone\_args.parse\_args()
749. zone = ZoneModel.query.get\_or\_404(id)
751. for key, value in args.items():
752. setattr(zone, key, value)
753. db.session.commit()
754. return zone, 200
756. except SQLAlchemyError as e:
757. db.session.rollback()
758. error\_message = {"error": "Database error occurred", "details": str(e)}
759. abort(500, message=error\_message)
761. except HTTPException as e:
762. error\_message = {"error": "HTTP error occurred", "details": str(e)}
763. abort(e.code, message=error\_message)
765. except Exception as e:
766. error\_message = {"error": "Unexpected error occurred", "details": str(e)}
767. abort(500, message=error\_message)
769. finally:
770. db.session.close()
771. @marshal\_with(zone\_fields)
772. def delete(self, id):
773. try:
774. zone = ZoneModel.query.get\_or\_404(id)
775. db.session.delete(zone)
776. db.session.commit()
777. zones = ZoneModel.query.all()
778. return zones, 200
780. except SQLAlchemyError as e:
781. db.session.rollback()
782. error\_message = {"error": "Database error occurred", "details": str(e)}
783. abort(500, message=error\_message)
785. except HTTPException as e:
786. error\_message = {"error": "HTTP error occurred", "details": str(e)}
787. abort(e.code, message=error\_message)
789. except Exception as e:
790. error\_message = {"error": "Unexpected error occurred", "details": str(e)}
791. abort(500, message=error\_message)
793. finally:
794. 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)

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)