

Лабораторная работа 8. Основы работы с планировщиками

Жунёв Андрей Александрович РИМ-150950

Цель работы:

Научиться создавать, запускать и управлять асинхронными и отложенными задачами с помощью TaskIQ.

Ход работы

1. Создание таблицы для отчетов в базе данных

Создание таблицы для отчетов необходимо для хранения сформированных отчетов по заказам. Таблица будет содержать информацию о дате отчета, идентификаторе заказа и количестве продукции в заказе. Это позволит хранить историю отчетов и предоставлять доступ к ним через REST API.

1.1. Создание модели для таблицы отчетов

```
125
126     class Report(Base):
127         __tablename__ = "reports"
128
129         id: Mapped[int] = mapped_column(
130             primary_key=True,
131             autoincrement=True,
132         )
133         report_at: Mapped[date] = mapped_column(nullable=False, index=True)
134         order_id: Mapped[int] = mapped_column(
135             ForeignKey("orders.id"), nullable=False, index=True
136         )
137         count_product: Mapped[int] = mapped_column(nullable=False)
138         created_at: Mapped[datetime] = mapped_column(
139             nullable=False, default=datetime.now
140         )
141
142         # Связь с заказом
143         order = relationship("Order", back_populates="reports")
144
145         __table_args__ = (
146             Index("idx_report_at_order_id", "report_at", "order_id"),
147         )
148
```

1.2. Создание миграции для таблицы отчетов

```
=> [app] resolving provenance for metadata file
[+] Running 9/9
✓ lab2-rabbitmq_worker      Built
✓ lab2-app                   Built
✓ Network lab2_pg_network_lab2   Created
✓ Container redis            Healthy
✓ Container rabbitmq_lab2    Started
✓ Container db_postgres_lab2  Healthy
✓ Container app_lab3          Started
✓ Container rabbitmq_worker_lab2 Started
✓ Container pgadmin4_lab2    Started
(lab2) > lab2 git:(main) ✘ cd app
(lab2) > app git:(main) ✘ alembic revision --autogenerate -m "add reports table"
INFO [alembic.runtime.migration] Context impl PostgresqlImpl.
INFO [alembic.runtime.migration] Will assume transactional DDL.
INFO [alembic.autogenerate.compare] Detected added table 'reports'
INFO [alembic.autogenerate.compare] Detected added index 'idx_report_at_order_id' on ('report_at', 'order_id')
INFO [alembic.autogenerate.compare] Detected added index 'ix_reports_order_id' on ('order_id',)
INFO [alembic.autogenerate.compare] Detected added index 'ix_reports_report_at' on ('report_at',)
INFO [alembic.ddl.postgresql] Detected sequence named 'addresses_id_seq' as owned by integer column 'addresses(id)', assuming SERIAL and omitting
INFO [alembic.ddl.postgresql] Detected sequence named 'order_items_id_seq' as owned by integer column 'order_items(id)', assuming SERIAL and omitting
Generating /Users/2madeira/DEV/first_sem/App_Dev_sem_1/lab2/app/migrations/versions/bf507bdc6f82_add_reports_table.py ... done
(lab2) > app git:(main) ✘ █
```

#K to generate command

```

1     """add reports table
2
3     Revision ID: bf507bdc6f82
4     Revises: 9e6b96ed2170
5     Create Date: 2025-12-16 16:24:03.961616
6
7     .....
8     from typing import Sequence, Union
9
10    from alembic import op
11    import sqlalchemy as sa
12
13
14    # revision identifiers, used by Alembic.
15    revision: str = 'bf507bdc6f82'
16    down_revision: Union[str, Sequence[str], None] = '9e6b96ed2170'
17    branch_labels: Union[str, Sequence[str], None] = None
18    depends_on: Union[str, Sequence[str], None] = None
19
20
21    def upgrade() -> None:
22        """Upgrade schema."""
23        # ### commands auto generated by Alembic - please adjust! ###
24        op.create_table('reports',
25            sa.Column('id', sa.Integer(), autoincrement=True, nullable=False),
26            sa.Column('report_at', sa.Date(), nullable=False),
27            sa.Column('order_id', sa.Integer(), nullable=False),
28            sa.Column('count_product', sa.Integer(), nullable=False),
29            sa.Column('created_at', sa.DateTime(), nullable=False),
30            sa.ForeignKeyConstraint(['order_id'], ['orders.id'], ),
31            sa.PrimaryKeyConstraint('id')
32        )
33        op.create_index('idx_report_at_order_id', 'reports', ['report_at', 'order_id'], unique=False)
34        op.create_index(op.f('ix_reports_order_id'), 'reports', ['order_id'], unique=False)
35        op.create_index(op.f('ix_reports_report_at'), 'reports', ['report_at'], unique=False)
36        # ### end Alembic commands ###
37
38
39    def downgrade() -> None:
40        """Downgrade schema."""
41        # ### commands auto generated by Alembic - please adjust! ###
42        op.drop_index(op.f('ix_reports_report_at'), table_name='reports')
43        op.drop_index(op.f('ix_reports_order_id'), table_name='reports')
44        op.drop_index('idx_report_at_order_id', table_name='reports')
45        op.drop_table('reports')
46        # ### end Alembic commands ###
47

```

(lab2) → app git:(main) ✘ alembic upgrade head
INFO [alembic.runtime.migration] Context impl PostgresqlImpl.
INFO [alembic.runtime.migration] Will assume transactional DDL.
INFO [alembic.runtime.migration] Running upgrade 9e6b96ed2170 -> bf507bdc6f82, add reports table

2. Установка зависимостей для работы с TaskIQ

Установка зависимостей необходима для интеграции TaskIQ в приложение. TaskIQ предоставляет удобный интерфейс для создания планировщиков задач с поддержкой cron-выражений и интеграции с RabbitMQ. Это позволит автоматически формировать отчеты по расписанию и отправлять их в очередь сообщений.

2.1. Добавление зависимостей в pyproject.toml

```

19      "uvicorn>=0.38.0",
20      "taskiq>=0.11.0",
21      "taskiq-faststream>=0.1.0",
22      "pytest>=8.0.0",
23      "pytest-asyncio>=0.23.0",

```

```
(lab2) → lab2 git:(main) ✘ uv sync
Resolved 91 packages in 1.61s
Prepared 9 packages in 381ms
Installed 13 packages in 25ms
+ aiohappyeyeballs==2.6.1
+ aiohttp==3.13.2
+ aiosignal==1.4.0
+ attrs==25.4.0
+ frozenlist==1.8.0
+ importlib-metadata==8.7.0
+ izulu==0.50.0
+ pycron==3.2.0
+ pytz==2025.2
+ taskiq==0.11.20
+ taskiq-dependencies==1.5.7
+ taskiq-faststream==0.3.2
+ zipp==3.23.0
○ (lab2) → lab2 git:(main) ✘
```

3. Создание модуля планировщика задач

Создание модуля планировщика необходимо для централизованного управления задачами, которые должны выполняться по расписанию. TaskIQ позволяет использовать cron-выражения для гибкого планирования задач и интегрируется с существующей инфраструктурой RabbitMQ.

3.1. Создание модуля для планировщика

```
+ zipp==3.23.0
● (lab2) → lab2 git:(main) ✘ uv sync
Resolved 92 packages in 382ms
Prepared 1 package in 185ms
Installed 1 package in 3ms
+ taskiq-aio-pika==0.5.0
○ (lab2) → lab2 git:(main) ✘
```

pyproject.toml M

scheduler.py U X

lab2 > app > scheduler.py > ...

```
1 """Модуль для настройки планировщика задач TaskIQ."""
2
3 import logging
4 import os
5 from datetime import date
6
7 from taskiq import TaskiqDepends
8 from taskiq_aio_pika import AioPikaBroker
9
10 # Настройка логирования
11 logging.basicConfig(level=logging.INFO)
12 logger = logging.getLogger(__name__)
13
14
15 def get_rabbitmq_url() -> str:
16     """
17         Получить URL подключения к RabbitMQ из переменных окружения.
18
19     Returns:
20         str: URL для подключения к RabbitMQ в формате amqp://user:password@host:port/vhost
21     """
22     host = os.getenv("RABBITMQ_HOST", "localhost")
23     port = os.getenv("RABBITMQ_PORT", "5672")
24     vhost = os.getenv("RABBITMQ_VHOST", "local")
25     user = os.getenv("RABBITMQ_USER", "guest")
26     password = os.getenv("RABBITMQ_PASSWORD", "guest")
27
28     # Убираем начальный слэш из vhost, если он есть
29     vhost = vhost.lstrip("/")
30
31     return f"amqp://{user}:{password}@{host}:{port}/{vhost}"
32
33
34 # Инициализация брокера TaskIQ
35 rabbitmq_url = get_rabbitmq_url()
36 logger.info("Initializing TaskIQ broker with RabbitMQ: %s", rabbitmq_url)
37
38 broker = AioPikaBroker(rabbitmq_url)
39
40
```

3.2. Настройка брокера для TaskIQ

```

+ taskiq-aio-pika==0.5.0
● (lab2) → lab2 git:(main) ✘ uv sync
Resolved 93 packages in 972ms
Prepared 2 packages in 374ms
Uninstalled 1 package in 14ms
Installed 2 packages in 2ms
- redis==7.1.0
+ redis==6.4.0
+ taskiq-redis==1.1.2
○ (lab2) → lab2 git:(main) ✘

```

```

34
35 def get_redis_url() -> str:
36     """
37         Получить URL подключения к Redis из переменных окружения.
38
39     Returns:
40         str: URL для подключения к Redis в формате redis://host:port/db
41     """
42     host = os.getenv("REDIS_HOST", "localhost")
43     port = int(os.getenv("REDIS_PORT", "6379"))
44     db = int(os.getenv("REDIS_DB", "0"))
45
46     return f"redis://{host}:{port}/{db}"
47
48
49 # Инициализация брокера TaskIQ
50 rabbitmq_url = get_rabbitmq_url()
51 logger.info("Initializing TaskIQ broker with RabbitMQ: %s", rabbitmq_url)
52
53 broker = AioPikaBroker(rabbitmq_url)
54
55 # Инициализация источников расписаний
56 redis_url = get_redis_url()
57 logger.info("Initializing Redis schedule source: %s", redis_url)
58
59 # RedisScheduleSource для динамических расписаний
60 redis_schedule_source = RedisScheduleSource(redis_url)
61
62 # LabelScheduleSource для статических расписаний из декораторов задач
63 label_schedule_source = LabelScheduleSource(broker)
64
65 # Создание планировщика с обоими источниками расписаний
66 scheduler = TaskiqScheduler(
67     broker=broker,
68     sources=[redis_schedule_source, label_schedule_source],
69 )
70

```

4. Создание репозитория для работы с отчетами

Создание репозитория необходимо для абстракции доступа к данным отчетов. Репозиторий обеспечивает единообразный интерфейс для работы с таблицей отчетов и следует паттерну Repository, используемому в проекте.

4.1. Создание репозитория отчетов

The screenshot shows the VS Code interface with the following details:

- Project Structure:** The left sidebar shows the project structure under `APP_DEV_SEM_1`. It includes `lab1`, `lab2`, `app`, `repositories`, and `schemas` directories. Inside `repositories`, there are files: `__init__.py`, `order_repository.py`, `product_repository.py`, `report_repository.py` (which is currently selected), and `user_repository.py`.
- Code Editor:** The right pane displays the `report_repository.py` file content. The code defines a `ReportRepository` class with an `async def create_report` method.
- Status Bar:** The status bar at the bottom shows the current file paths: `pyproject.toml M`, `scheduler.py U`, and `report_repository.py U X`.

```
1 """Репозиторий для работы с отчетами."""
2
3 from datetime import date
4
5 from sqlalchemy import select
6 from sqlalchemy.ext.asyncio import AsyncSession
7
8 from app.models import Report
9
10
11 class ReportRepository:
12     """Репозиторий для CRUD операций с отчетами."""
13
14     async def create_report(
15         self,
16         session: AsyncSession,
17         report_at: date,
18         order_id: int,
19         count_product: int,
20     ) -> Report:
21         """
22             Создать запись отчета.
23
24         Args:
```

4.2. Добавление методов в репозиторий заказов

The screenshot shows the `order_repository.py` file in the code editor. The `get_orders_by_date` method is implemented as follows:

```
194
195     async def get_orders_by_date(
196         self, session: AsyncSession, order_date: date
197     ) -> list[Order]:
198
199         """
200             Получить все заказы, созданные в указанную дату.
201
202         Args:
203             session: Асинхронная сессия базы данных
204             order_date: Дата для получения заказов
205
206         Returns:
207             Список заказов с загруженными items, созданных в указанную дату
208
209             # Создаем диапазон времени для указанной даты
210             start_datetime = datetime.combine(order_date, datetime.min.time())
211             end_datetime = datetime.combine(order_date, datetime.max.time())
212
213             stmt = (
214                 select(Order)
215                 .where(Order.created_at >= start_datetime)
216                 .where(Order.created_at <= end_datetime)
217                 .options(selectinload(Order.items))
218                 .order_by(Order.created_at.desc())
219             )
220
221             result = await session.execute(stmt)
222             return list[Order](result.scalars().all())
```

5. Создание сервиса для работы с отчетами

Создание сервиса необходимо для бизнес-логики работы с отчетами. Сервис инкапсулирует сложную логику формирования и обработки отчетов, обеспечивая разделение ответственности между слоями приложения.

5.1. Создание сервиса отчетов

The screenshot shows the VS Code interface with the following details:

- Project Structure:** The left sidebar displays the project structure under `APP_DEV_SEM_1`. It includes `lab1`, `lab2`, `app` (containing `migrations`, `repositories`, and several repository files), `services` (containing `order_service.py`, `product_service.py`, `report_service.py`, and `user_service.py`), and various configuration and utility files like `alembic.ini`, `database.py`, `dependencies.py`, `exceptions.py`, `models.py`, `push_data.py`, and `push_products_orders.py`.
- Code Editor:** The main editor tab is `report_service.py`, which contains the following Python code:

```
1  """Сервис для бизнес-логики работы с отчетами."""
2
3  from datetime import date
4
5  from sqlalchemy.ext.asyncio import AsyncSession
6
7  from app.repositories.order_repository import OrderRepository
8  from app.repositories.report_repository import ReportRepository
9
10
11 class ReportService:
12     """Сервис для бизнес-логики работы с отчетами."""
13
14     def __init__(self,
15                  order_repository: OrderRepository,
16                  report_repository: ReportRepository,
17                 ):
18         """
19             Инициализация сервиса.
20
21             Args:
22                 order_repository: Репозиторий для работы с заказами (Dependency Injection)
23                 report_repository: Репозиторий для работы с отчетами (Dependency Injection)
24
25         self.order_repository = order_repository
26         self.report_repository = report_repository
27
28     async def generate_report(
29                     self, session: AsyncSession, report_date: date
30                 ) -> list:
31         """
32
33         Сформировать отчет за указанную дату.
34
35         Получает все заказы за указанную дату, подсчитывает количество продукции
36         в каждом заказе и создает записи отчетов.
37
38         Args:
39             session: Асинхронная сессия базы данных
40             report_date: Дата для формирования отчета
41
```

5.2. Создание задачи для формирования отчета

The screenshot shows the VS Code interface with the project structure on the left and the code editor on the right.

Project Structure:

- APP_DEV_SEM_1
 - .vscode
 - lab1
 - lab2
 - __pycache__
 - .pytest_cache
 - .venv
 - .vscode
 - app
 - __pycache__
 - cache
 - controllers
 - migrations
 - repositories
 - __pycache__
 - __init__.py
 - order_repository.py
 - product_repository.py
 - report_repository.py
 - user_repository.py
 - schemas
 - services
 - __pycache__
 - __init__.py
 - order_service.py
 - product_service.py
 - report_service.py
 - user_service.py
 - alembic.ini
 - database.py
 - dependencies.py
 - exceptions.py
 - models.py
 - push_data.py
 - push_products_orders.py
 - queries.py
 - rabbitmq_consumer.py
 - redis_client.py
 - scheduler.py
 - db_init
 - htmlcov
 - tests

Code Editor (scheduler.py):

```
lab2 > app > scheduler.py > ...
75     broker=broker,
76     sources=[redis_schedule_source, label_schedule_source],
77 )
78
79
80 # Провайдер сессии БД для TaskIQ
81 async def provide_db_session_for_taskiq() -> AsyncSession:
82     """
83     Провайдер сессии базы данных для задач TaskIQ.
84
85     Yields:
86         AsyncSession: Асинхронная сессия базы данных
87     """
88     async with async_session_factory() as session:
89         try:
90             yield session
91         finally:
92             await session.close()
93
94
95 # Задача для формирования отчета
96 @broker.task(
97     schedule=[
98         {
99             "cron": "0 0 * * *", # Каждый день в полночь
100            "cron_offset": None,
101            "args": [],
102            "kwargs": {}
103        }
104    ]
105 )
106 async def my_scheduled_task(
107     report_date: date | None = None,
108     db_session: AsyncSession = TaskiqDepends(provide_db_session_for_taskiq),
109     report_service: ReportService = TaskiqDepends(provide_report_service),
110 ) -> None:
111     """
112     Задача для формирования отчета по заказам.
113
114     Использует ReportService для формирования отчетов за указанную дату
115     и отправляет сообщение в RabbitMQ с информацией о созданных отчетах.
116
117     Args:
118         report_date: Дата для формирования отчета (по умолчанию текущая дата)
119         db_session: Сессия базы данных (внедряется через DI)
120         report_service: Сервис для работы с отчетами (внедряется через DI)
121
122     if report_date is None:
123         report_date = date.today()
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
617
618
619
619
620
621
622
623
624
625
625
626
627
627
628
629
629
630
631
631
632
633
633
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1
```

pyproject.toml M

scheduler.py U

dependencies.py M X

lab2 > app > dependencies.py > ...

```
131     ~/DEV/first_sem/App_Dev_sem_1/lab2/app · Contains emphasized items )
131     | return OrderService(order_repository, product_repository)
132
133     | #L to chat, #K to generate
134     | async def provide_report_repository(
135     |     db_session: AsyncSession,
136     ) -> ReportRepository:
137     |     """
138     |     Провайдер репозитория отчетов.
139
140     |     Args:
141     |         db_session: Сессия базы данных (внедряется через DI)
142
143     |     Returns:
144     |         ReportRepository: Экземпляр репозитория отчетов
145     |         """
146     |     return ReportRepository()
147
148
149     | async def provide_report_service(
150     |     order_repository: OrderRepository,
151     |     report_repository: ReportRepository,
152     ) -> ReportService:
153     |     """
154     |     Провайдер сервиса отчетов.
155
156     |     Args:
157     |         order_repository: Репозиторий заказов (внедряется через DI)
158     |         report_repository: Репозиторий отчетов (внедряется через DI)
159
160     |     Returns:
161     |         ReportService: Экземпляр сервиса отчетов
162     |         """
163     |     return ReportService(order_repository, report_repository)
164
```

5.3. Настройка зависимостей для задач

The screenshot shows a code editor with two tabs: 'pyproject.toml' and 'scheduler.py'. The 'scheduler.py' tab is active and displays Python code for dependency injection using Taskiq. The code defines three providers: 'provide_order_repository_for_taskiq', 'provide_report_repository_for_taskiq', and 'provide_report_service_for_taskiq'. Each provider takes a database session and returns a repository or service. The 'provide_report_service_for_taskiq' provider also depends on the other two providers.

```
94
95
96     async def provide_order_repository_for_taskiq(
97         db_session: AsyncSession = TaskiqDepends(provide_db_session_for_taskiq),
98     ) -> OrderRepository:
99         """
100             Провайдер репозитория заказов для задач TaskIQ.
101
102             Args:
103                 db_session: Сессия базы данных (внедряется через DI)
104
105             Returns:
106                 OrderRepository: Экземпляр репозитория заказов
107         """
108
109         from app.repositories.order_repository import OrderRepository
110
111         return OrderRepository()
112
113
114     async def provide_report_repository_for_taskiq(
115         db_session: AsyncSession = TaskiqDepends(provide_db_session_for_taskiq),
116     ) -> ReportRepository:
117         """
118             Провайдер репозитория отчетов для задач TaskIQ.
119
120             Args:
121                 db_session: Сессия базы данных (внедряется через DI)
122
123             Returns:
124                 ReportRepository: Экземпляр репозитория отчетов
125         """
126
127         from app.repositories.report_repository import ReportRepository
128
129
130     async def provide_report_service_for_taskiq(
131         order_repository: OrderRepository = TaskiqDepends(
132             provide_order_repository_for_taskiq
133         ),
134         report_repository: ReportRepository = TaskiqDepends(
135             provide_report_repository_for_taskiq
136         ),
137     ) -> ReportService:
138         """
139             Провайдер сервиса отчетов для задач TaskIQ.
140
141             Args:
142                 order_repository: Репозиторий заказов (внедряется через DI)
143                 report_repository: Репозиторий отчетов (внедряется через DI)
144         """
```

6. Создание схемы для отчетов

Создание схемы необходимо для валидации данных при работе с отчетами через REST API. Схемы обеспечивают типизацию и валидацию входных и выходных данных.

Создание схемы отчетов

__init__.py

The screenshot shows a code editor with several tabs at the top: 'pyproject.toml M', 'scheduler.py U', 'report_schema.py U', and '__init__.py M X'. The current file is '__init__.py'. The code in the editor is as follows:

```
1  """Схемы Pydantic для валидации данных."""
2
3  from app.schemas.order_schema import (
4      OrderCreate,
5      OrderItemCreate,
6      OrderItemResponse,
7      OrderListResponse,
8      OrderResponse,
9      OrderUpdate,
10     OrderUpdateMessage,
11 )
12 from app.schemas.product_schema import (
13     ProductCreate,
14     ProductResponse,
15     ProductUpdate,
16     ProductUpdateMessage,
17 )
18 from app.schemas.report_schema import (
19     ReportCreate,
20     ReportDateRequest,
21     ReportResponse,
22 )
23 from app.schemas.user_schema import (
24     UserCreate,
25     UserListResponse,
26     UserResponse,
27     UserUpdate,
28 )
29
30 __all__ = [
31     "UserCreate",
32     "UserUpdate",
33     "UserResponse",
34     "UserListResponse",
35     "ProductCreate",
36     "ProductResponse",
37     "ProductUpdate",
38     "ProductUpdateMessage",
39     "OrderCreate",
40     "OrderItemCreate",
41     "OrderItemResponse",
42     "OrderResponse",
43     "OrderListResponse",
44     "OrderUpdate",
45     "OrderUpdateMessage",
46     "ReportCreate",
47     "ReportDateRequest",
48     "ReportResponse",
49 ]
50
```

pyproject.toml M

scheduler.py U

report_schema.py U X

lab2 > app > schemas > report_schema.py > ...

```
1  """Схемы для работы с отчетами."""
2
3  from datetime import date, datetime
4
5  from pydantic import BaseModel, ConfigDict, Field
6
7
8  class ReportResponse(BaseModel):
9      """Схема для ответа API с данными отчета."""
10
11     id: int = Field(..., gt=0, description="Уникальный идентификатор отчета")
12     report_at: date = Field(..., description="Дата отчета")
13     order_id: int = Field(..., gt=0, description="ID заказа")
14     count_product: int = Field(..., ge=0, description="Количество продукции в заказе")
15     created_at: datetime = Field(..., description="Дата и время создания отчета")
16
17     model_config = ConfigDict(from_attributes=True)
18
19
20 class ReportDateRequest(BaseModel):
21     """Схема для запроса отчета по дате."""
22
23     date: date = Field(..., description="Дата для получения отчета")
24
25
26 class ReportCreate(BaseModel):
27     """Схема для создания отчета (опционально, если нужен ручной ввод)."""
28
29     report_at: date = Field(..., description="Дата отчета")
30     order_id: int = Field(..., gt=0, description="ID заказа")
31     count_product: int = Field(..., ge=0, description="Количество продукции в заказе")
32
33
```

7. Создание контроллера для REST API эндпоинта /report

Создание контроллера необходимо для предоставления доступа к отчетам через REST API. Эндпоинт `/report` позволит получать отчеты за указанную дату, что обеспечит удобный доступ к данным для клиентских приложений.

7.1. Создание контроллера отчетов

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows the project structure under `APP_DEV_SEM_1`. Key folders include `lab1`, `lab2`, `app`, `controllers`, `schemas`, and `services`. Inside `app/controllers`, there are files like `order_controller.py`, `product_controller.py`, `report_controller.py`, and `user_controller.py`.
- Editor:** The active file is `report_controller.py`. The code defines a `ReportController` class that inherits from `Controller`. It has a single endpoint `@get("/")` named `get_report` which returns a list of `ReportResponse` objects. The code uses `litestar` and `litestar_params` libraries.
- Status Bar:** Shows the current file path: `pyproject.toml M` and the file name `report_controller.py U`.

```
1     """Контроллер для управления отчетами."""
2
3     from datetime import date
4
5     from litestar import Controller, get
6     from litestar.params import Parameter
7
8     from /DEV/first_sem/App_Dev_sem_1/lab2.pytest_cache import AsyncSession
9
10    from app.schemas.report_schema import ReportResponse
11    from app.services.report_service import ReportService
12
13    class ReportController(Controller):
14        """Контроллер для управления отчетами."""
15
16        path = "/report"
17
18        @get("/")
19        async def get_report(
20            self,
21            report_service: ReportService,
22            db_session: AsyncSession,
23            date: date | None = Parameter(
24                default=None, description="Дата для получения отчета (по умолчанию текущая дата)"
25            ),
26        ) -> list[ReportResponse]:
27            """
28            Получить отчеты за указанную дату.
29
30            Args:
31                report_service: Сервис для работы с отчетами
32                db_session: Сессия базы данных
33                date: Дата для получения отчета (опционально, по умолчанию текущая дата)
34
35            Returns:
36                list[ReportResponse]: Список отчетов за указанную дату
37
38            if date is None:
39                date = date.today()
40
41            reports = await report_service.get_report_by_date(db_session, date)
42            return [ReportResponse.model_validate(report) for report in reports]
43
44
```

7.2. Регистрация контроллера в приложении

```
pyproject.toml M report_controller.py U main.py M X

lab2 > 🐍 main.py > ...

1 import os
2 from litestar import Litestar
3 from litestar.di import Provide
4 from litestar.openapi import OpenAPIConfig
5
6 from app.controllers.order_controller import OrderController
7 from app.controllers.product_controller import ProductController
8 from app.controllers.report_controller import ReportController
9 from app.controllers.user_controller import UserController
10 from app.dependencies import (
11     provide_db_session,
12     provide_order_repository,
13     provide_order_service,
14     provide_product_repository,
15     provide_product_service,
16     provide_redis_client,
17     provide_report_repository,
18     provide_report_service,
19     provide_user_repository,
20     provide_user_service,
21 )
22
23
24 app = Litestar(
25     route_handlers=[
26         UserController,
27         ProductController,
28         OrderController,
29         ReportController,
```

8. Запуск планировщика TaskIQ

Запуск планировщика необходим для активации автоматического выполнения задач по расписанию. Планировщик будет отслеживать cron-выражения и запускать задачи в указанное время, отправляя их в очередь RabbitMQ для обработки.

8.1. Настройка команды для запуска планировщика

```
235 # Экспорт для использования в CLI
236 # Объект scheduler используется командой: taskiq scheduler app.scheduler:scheduler
237 # Все задачи зарегистрированы в брокере через декоратор @broker.task
238
239
```

8.2. Запуск планировщика

```
Problems Output Debug Console Terminal Ports

(lab2) > lab2 git:(main) ✘ uv sync
Resolved 93 packages in 16ms
Audited 91 packages in 13ms
(lab2) > lab2 git:(main) ✘ docker compose ps
NAME           IMAGE          COMMAND                  SERVICE      CREATED        STATUS          PORTS
app_lab3        lab2-app      "./entrypoint.sh uv ..."  app          43 minutes ago Up 43 minutes   0.0.0.0:8000->8000/tcp, [::]:8000->8000/tcp
db_postgres_lab2    postgres:17.6-bookworm "docker-entrypoint.s..." db          43 minutes ago Up 43 minutes (healthy)  0.0.0.0:5433->5432/tcp, [::]:5433->5432/tcp
pgadmin4_lab2    dpage/pgadmin4:9.8.0  "/entrypoint.sh" pgadmin      43 minutes ago Up 43 minutes   0.0.0.0:8081->80/tcp, [::]:8081->80/tcp
rabbitmq_lab2    rabbitmq:3-management "docker-entrypoint.s..." rabbitmq    43 minutes ago Up 43 minutes   0.0.0.0:5672->5672/tcp, [::]:5672->5672/tcp, 0.0.0.0:15672->15672/tcp
rabbitmq_worker_lab2    lab2-rabbitmq_worker  "./entrypoint.sh uv ..." rabbitmq_worker 43 minutes ago Up 43 minutes   8000/tcp
redis            redis:7-alpine   "docker-entrypoint.s..." redis       43 minutes ago Up 43 minutes (healthy)  0.0.0.0:6379->6379/tcp, [::]:6379->6379/tcp
(lab2) > lab2 git:(main) ✘ uv run taskiq scheduler app.scheduler:scheduler --skip-first-run
[2025-12-16 17:07:22,135] [INFO] [scheduler:<module>:60] Initializing TaskIQ broker with RabbitMQ: amqp://guest:guest@localhost:5672/local
[2025-12-16 17:07:22,135] [INFO] [[scheduler:<module>:66] Initializing Redis schedule source: redis://localhost:6379/0
[2025-12-16 17:07:22,136] [INFO] [[run:run_scheduler:272] Starting scheduler.
[2025-12-16 17:07:22,156] [INFO] [[run:run_scheduler:274] Startup completed.
[2025-12-16 17:07:22,156] [INFO] [[run:run_scheduler:284] Skipping first run. Waiting 37 seconds.
[2025-12-16 17:08:00,000] [INFO] [[run:run_scheduler:286] First run skipped. The scheduler is now running.
```

[Overview](#)[Connections](#)[Channels](#)[Exchanges](#)[Queues and Streams](#)[Admin](#)

Queues

▼ All queues (7)

Pagination

 Page of 1 - Filter: Regex ? [?](#)

Overview					Messages			Message rates			+/-
Virtual host	Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack	
local	order	classic	Args	green running	0	0	0				
local	order_update	classic	Args	green running	0	0	0				
local	product	classic	Args	green running	0	0	0				
local	product_update	classic	Args	green running	0	0	0				
local	taskiq	classic	DLX DLK	green running	0	0	0				
local	taskiq.dead_letter	classic		green running	0	0	0				
local	taskiq.delay	classic	DLX DLK	green running	0	0	0				

[▶ Add a new queue](#)
[Overview](#)[Connections](#)[Channels](#)[Exchanges](#)[Queues and Streams](#)[Admin](#)

Exchanges

▼ All exchanges (8)

Pagination

 Page of 1 - Filter: Regex ? [?](#)

Virtual host	Name	Type	Features	Message rate in	Message rate out	+/-
local	(AMQP default)	direct	D			
local	amq.direct	direct	D			
local	amq.fanout	fanout	D			
local	amq.headers	headers	D			
local	amq.match	headers	D			
local	amq.rabbitmq.trace	topic	D I			
local	amq.topic	topic	D			
local	taskiq	topic				

[▶ Add a new exchange](#)

8.3. Настройка планировщика в Docker Compose (опционально)

```
169  > Run Service
170  taskiq_worker:
171    build:
172      context: .
173      dockerfile: Dockerfile
174      container_name: taskiq_worker_lab2
175      environment:
176        - DATABASE_URL=postgresql+asyncpg://${POSTGRES_USER}:admin:${POSTGRES_PASSWORD}:admin@db:5432/${POSTGRES_DB}:lab_db2
177        - DB_HOST=${DB_HOST:-db}
178        - DB_PORT=${DB_PORT:-5432}
179        - RABBITMQ_HOST=${RABBITMQ_HOST:-rabbitmq}
180        - RABBITMQ_PORT=${RABBITMQ_PORT:-5672}
181        - RABBITMQ_VHOST=${RABBITMQ_VHOST:-local}
182        - RABBITMQ_USER=${RABBITMQ_USER:-guest}
183        - RABBITMQ_PASSWORD=${RABBITMQ_PASSWORD:-guest}
184        - REDIS_HOST=${REDIS_HOST:-redis}
185        - REDIS_PORT=${REDIS_PORT:-6379}
186        - REDIS_DB=${REDIS_DB:-0}
187        - REDIS_DECODE_RESPONSES=${REDIS_DECODE_RESPONSES:-true}
188      volumes:
189        - ./app:/app/app
190      networks:
191        - pg_network_lab2
192      depends_on:
193        db:
194          condition: service_healthy
195        rabbitmq:
196          condition: service_started
197        redis:
198          condition: service_healthy
199        scheduler:
200          condition: service_started
201      command: ["uv", "run", "taskiq", "worker", "app.scheduler:broker"]
202      restart: unless-stopped
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
886
887
888
889
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1087
1088
1089
1089
1089
1090
1091
1092
1093
1094
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1279
1280
1281
1282
1283
1284
1285
1286
1287
1287
1288
1289
1289
1289
1290
1291
1292
1293
1294
1295
1296
1297
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1387
1387
1388
1389
1389
1389
1390
1391
1392
1393
1394
1395
1396
1397
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1479
1480
1481
1482
1483
1484
1485
1486
1487
1487
1488
1489
1489
1489
1490
1491
1492
1493
1494
1495
1496
1497
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1589
1590
1591
1592
1593
1594
1595
1596
1597
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1688
1689
1689
1690
1691
1692
1693
1694
16
```

```

[1] Running 8/8
✓ lab2-scheduler      Built
✓ Container rabbitmq_lab2  Running
✓ Container redis       Healthy
✓ Container db_postgres_lab2 Healthy
✓ Container app_lab3     Running
✓ Container scheduler_lab2 Started
● (lab2) ➜ app git:(main) ✘ docker compose ps scheduler
NAME           IMAGE          COMMAND         SERVICE    CREATED        STATUS        PORTS
scheduler_lab2 lab2-scheduler ".entrypoint.sh uv ..." scheduler  12 seconds ago Up 10 seconds  8000/tcp
● (lab2) ➜ app git:(main) ✘ docker-compose logs scheduler
scheduler_lab2 | Connection to db (172.18.0.4) 5432 port [tcp/postgresql] succeeded!
scheduler_lab2 | INFO [alembic.runtime.migration] Context impl PostgresqlImpl.
scheduler_lab2 | INFO [alembic.runtime.migration] Will assume transactional DDL.
scheduler_lab2 | [2025-12-16 12:20:36,732][INFO] [scheduler:<module>:60] Initializing TaskIQ broker with RabbitMQ: amqp://guest:guest@rabbitmq:5672/local
scheduler_lab2 | [2025-12-16 12:20:36,732][INFO] [scheduler:<module>:66] Initializing Redis schedule source: redis://redis:6379/0
scheduler_lab2 | [2025-12-16 12:20:36,732][INFO] [run:run_scheduler:272] Starting scheduler.
scheduler_lab2 | [2025-12-16 12:20:36,738][INFO] [run:run_scheduler:274] Startup completed.
scheduler_lab2 | [2025-12-16 12:20:36,738][INFO] [run:run_scheduler:284] Skipping first run. Waiting 23 seconds.
○ (lab2) ➜ app git:(main) ✘

```

Планировщик корректно стартует из контейнера.

9. Тестирование функциональности

Тестирование проводилось вручную, также написаны автотесты с использованием pytest. Ход тестирования описан ниже

9.1. Тестирование создания отчетов

```

153   # Задача для формирования отчета
154   @broker.task(
155     schedule=[
156       {
157         "cron": "* * * * *", # Каждый день в полночь – изменил для тестов на каждую минуту
158         "cron_offset": None,
159         "args": [],
160         "kwargs": {},
161       }
162     ]
163   )

```

создал заказы через producer.py

```

st=localhost port=5672 virtual_host=local ssl=False>, error=ConnectionClosedByClient: (200) 'Normal shutdown'
2025-12-16 17:33:11,082 - __main__ - INFO Connection to RabbitMQ closed
● (lab2) ➜ lab2 git:(main) ✘ docker exec -it db_postgres_lab2 psql -U admin -d lab_db2 -c "SELECT id, user_id, created_at FROM orders WHERE DATE(created_at) = CURRENT_DATE LIMIT 5;" + ...
+-----+
| id | user_id | created_at |
+-----+
| 5  | 1        | 2025-12-16 12:33:11.090942 |
| 6  | 1        | 2025-12-16 12:33:11.090201 |
| 4  | 1        | 2025-12-16 12:33:11.088694 |
+-----+
(3 rows)

○ (lab2) ➜ lab2 git:(main) ✘

```

The terminal shows the creation of three test orders via producer.py and their retrieval via psql. The output of psql shows three rows of data: (5, 1, 2025-12-16 12:33:11.090942), (6, 1, 2025-12-16 12:33:11.090201), and (4, 1, 2025-12-16 12:33:11.088694).

```

Problems Output Debug Console Terminal Ports + ... zsh lab2
○ (lab2) ➜ lab2 git:(main) ✘ docker-compose logs -f taskiq_worker
taskiq_worker_lab2 | Connection to db (172.18.0.4) 5432 port [tcp/postgresql] succeeded!
taskiq_worker_lab2 | INFO [alembic.runtime.migration] Context impl PostgresqlImpl.
taskiq_worker_lab2 | INFO [alembic.runtime.migration] Will assume transactional DDL.
taskiq_worker_lab2 | [2025-12-16 12:54:46,319][taskiq.worker][INFO] [MainProcess] Pid of a main process: 1
5
taskiq_worker_lab2 | [2025-12-16 12:54:46,319][taskiq.worker][INFO] [MainProcess] Starting 2 worker processes.
taskiq_worker_lab2 | [2025-12-16 12:54:46,320][taskiq.process-manager][INFO] [MainProcess] Started process worker-0 with pid 16
taskiq_worker_lab2 | [2025-12-16 12:54:46,321][taskiq.process-manager][INFO] [MainProcess] Started process worker-1 with pid 17
taskiq_worker_lab2 | [2025-12-16 12:54:46,709][app.scheduler][INFO] [worker-1] Initializing TaskIQ broker with RabbitMQ: amqp://guest:guest@rabbitmq:5672/local
taskiq_worker_lab2 | [2025-12-16 12:54:46,709][app.scheduler][INFO] [worker-1] Initializing Redis schedule source: redis://redis:6379/0
taskiq_worker_lab2 | [2025-12-16 12:54:46,712][app.scheduler][INFO] [worker-0] Initializing TaskIQ broker with RabbitMQ: amqp://guest:guest@rabbitmq:5672/local
taskiq_worker_lab2 | [2025-12-16 12:54:46,712][app.scheduler][INFO] [worker-0] Initializing Redis schedule source: redis://redis:6379/0
taskiq_worker_lab2 | [2025-12-16 12:54:46,721][taskiq.receiver.receiver][INFO] [worker-1] Listening started.
taskiq_worker_lab2 | [2025-12-16 12:54:46,722][taskiq.receiver.receiver][INFO] [worker-0] Listening started.
taskiq_worker_lab2 | [2025-12-16 12:55:00,023][taskiq.receiver.receiver][INFO] [worker-0] Executing task app.scheduler:my_scheduled_task with ID: df2f363b144714223b7ce97e1ba955d7
taskiq_worker_lab2 | [2025-12-16 12:55:00,024][app.scheduler][INFO] [worker-0] Starting report generation for date: 2025-12-16
+-----+
| id | report_at | order_id | count_product | created_at |
+-----+
12 | 2025-12-16 | 4 | 2 | 2025-12-16 12:55:00.067758
11 | 2025-12-16 | 6 | 3 | 2025-12-16 12:55:00.066955
10 | 2025-12-16 | 5 | 4 | 2025-12-16 12:55:00.064881
9  | 2025-12-16 | 4 | 2 | 2025-12-16 12:52:01.076059
+-----+
152
153   # Задача для формирования отчета
154   @broker.task(
155     schedule=[
156       {
157         "cron": "0 0 * * *", # Каждый день в полночь – возвращаем обратно после тестов
158         "cron_offset": None,
159         "args": [],
160         "kwargs": {},
161       }
162     ]
163   )

```

Задача успешно поставляет в очередь и обрабатывается.

9.2. Тестирование REST API эндпоинта

```
lab2 > [-] test_report_api.sh
1  #!/bin/bash
2
3  BASE_URL="http://localhost:8000"
4  GREEN='\033[0;32m'
5  RED='\033[0;31m'
6  YELLOW='\033[1;33m'
7  BLUE='\033[0;34m'
8  NC='\033[0m' # No Color
9
10 # Счетчики
11 TOTAL_TESTS=0
12 PASSED_TESTS=0
13 FAILED_TESTS=0
14
15 # Функция для вывода результата теста
16 test_result() {
17     local test_name="$1"
18     local status="$2"
19     local message="$3"
20
21     TOTAL_TESTS=$((TOTAL_TESTS + 1))
22
23     if [ "$status" = "PASS" ]; then
24         echo -e "${GREEN}\u25a1${NC} $test_name: $message"
25         PASSED_TESTS=$((PASSED_TESTS + 1))
26     else
27         echo -e "${RED}\u25a1${NC} $test_name: $message"
28         FAILED_TESTS=$((FAILED_TESTS + 1))
29     fi
30 }
31
32 # Функция для проверки HTTP статуса
33 check_http_status() {
34     local expected_status="$1"
35     local actual_status="$2"
36     local test_name="$3"
37
38     if [ "$actual_status" -eq "$expected_status" ]; then
39         test_result "$test_name" "PASS" "HTTP статус $actual_status (ожидался $expected_status)"
40         return 0
41     else
42         test_result "$test_name" "FAIL" "HTTP статус $actual_status (ожидался $expected_status)"
43         return 1
44     fi
45 }
```

Тесты проверяют следующее

1. Запрос без параметра (отчет за сегодня)
2. Запрос с параметром даты
3. Запрос с датой без отчетов (пустой массив)

4. Структуру JSON ответа

The terminal shows the output of `docker compose ps` for a multi-service Docker stack named 'lab2'. It lists services like app, db, pgadmin, rabbitmq, redis, scheduler, taskiq, and lab2-taskiq-worker. Following this, a command is run to execute a test script named `test_report_api.sh`:

```
(lab2) + lab2 git:(main) x chmod +x test_report_api.sh
(lab2) + lab2 git:(main) x ./test_report_api.sh
== Тестирование REST API эндпоинта /report ==
Проверка доступности сервиса...
x Сервис недоступен по адресу http://localhost:8000
Убедитесь, что приложение запущено: docker-compose up -d
(lab2) + lab2 git:(main) x ./test_report_api.sh
== Тестирование REST API эндпоинта /report ==

Проверка доступности сервиса...
✓ Сервис доступен (HTTP 200)

Тест 1: Запрос без параметра date (отчет за сегодня)
✓ Тест 1.1: HTTP статус: HTTP статус 200 (ожидался 200)
✓ Тест 1.2: Валидный JSON массив: Массив содержит 15 элементов
✓ Тест 1.3: Структура данных: Все обязательные поля присутствуют

Тест 2: Запрос с параметром date (сегодняшняя дата)
✓ Тест 2.1: HTTP статус: HTTP статус 200 (ожидался 200)
✓ Тест 2.2: Валидный JSON: Ответ является валидным JSON
✓ Тест 2.3: Корректность даты: Все отчеты имеют дату 2025-12-16

Тест 3: Запрос с датой без отчетов
✓ Тест 3.1: HTTP статус: HTTP статус 200 (ожидался 200)
✓ Тест 3.2: Пустой массив: Возвращен пустой массив для даты без отчетов

Тест 5: Проверка структуры JSON ответа
✓ Тест 4.1: Наличие обязательных полей: Все поля присутствуют: id report_at order_id count_product created_at
✓ Тест 4.2: Типы данных: Все типы данных корректны

Тест 5: Проверка HTTP заголовков
✓ Тест 5.1: Content-Type заголовок: Content-Type: application/json

===== ИТОГОВАЯ СТАТИСТИКА ТЕСТОВ =====
Всего тестов выполнено: 11
Успешно пройдено: 11
Произошло: 0

Процент успешности: 100%
ВСЕ ТЕСТЫ ПРОЙДЕНЫ УСПЕШНО! ✓
```

тестирование автотестами - добавлены тесты для репозитория, контроллера и сервиса работающего с отчетами

The terminal displays the results of a pytest session across multiple files and modules. The output is color-coded to indicate success (green) and warnings (yellow). A summary at the bottom shows 107 passed tests and 1 warning.

```
tests/test_repositories/test_user_repository.py::TestUserRepository::test_get_all_users_with_pagination PASSED
tests/test_repositories/test_user_repository.py::TestUserRepository::test_get_users_with_filter PASSED
tests/test_services/test_order_service.py::TestOrderService::test_create_order_success PASSED
tests/test_services/test_order_service.py::TestOrderService::test_create_order_insufficient_stock PASSED
tests/test_services/test_order_service.py::TestOrderService::test_create_order_invalid_product PASSED
tests/test_services/test_order_service.py::TestOrderService::test_create_order_invalid_user PASSED
tests/test_services/test_order_service.py::TestOrderService::test_create_order_invalid_address PASSED
tests/test_services/test_order_service.py::TestOrderService::test_create_order_address_belongs_to_different_user PASSED
tests/test_services/test_order_service.py::TestOrderService::test_get_by_id_success PASSED
tests/test_services/test_order_service.py::TestOrderService::test_update_order_success PASSED
tests/test_services/test_order_service.py::TestOrderService::test_update_order_not_found PASSED
tests/test_services/test_order_service.py::TestOrderService::test_delete_order_success PASSED
tests/test_services/test_product_service.py::TestProductService::test_get_by_id_success PASSED
tests/test_services/test_product_service.py::TestProductService::test_create_product_success PASSED
tests/test_services/test_product_service.py::TestProductService::test_create_product_invalid_price PASSED
tests/test_services/test_product_service.py::TestProductService::test_create_product_negative_stock PASSED
tests/test_services/test_product_service.py::TestProductService::test_update_product_success PASSED
tests/test_services/test_product_service.py::TestProductService::test_update_product_not_found PASSED
tests/test_services/test_product_service.py::TestProductService::test_update_product_invalid_price PASSED
tests/test_services/test_product_service.py::TestProductService::test_update_product_negative_stock PASSED
tests/test_services/test_product_service.py::TestProductService::test_delete_product_success PASSED
tests/test_services/test_product_service.py::TestProductService::test_get_by_filter PASSED
tests/test_services/test_product_service.py::TestProductService::test_count PASSED
tests/test_services/test_report_service.py::TestReportService::test_generate_report_success PASSED
tests/test_services/test_report_service.py::TestReportService::test_generate_report_no_orders PASSED
tests/test_services/test_report_service.py::TestReportService::test_get_report_by_date PASSED
tests/test_services/test_user_service.py::TestUserService::test_get_by_id_success PASSED
tests/test_services/test_user_service.py::TestUserService::test_get_by_id_not_found PASSED
tests/test_services/test_user_service.py::TestUserService::test_create_user_success PASSED
tests/test_services/test_user_service.py::TestUserService::test_create_user_duplicate_email PASSED
tests/test_services/test_user_service.py::TestUserService::test_create_user_duplicate_username PASSED
tests/test_services/test_user_service.py::TestUserService::test_update_user_success PASSED
tests/test_services/test_user_service.py::TestUserService::test_update_user_not_found PASSED
tests/test_services/test_user_service.py::TestUserService::test_update_user_duplicate_email PASSED
tests/test_services/test_user_service.py::TestUserService::test_delete_user_success PASSED
tests/test_services/test_user_service.py::TestUserService::test_get_by_filter PASSED
tests/test_services/test_user_service.py::TestUserService::test_count PASSED

===== warnings summary =====
main.py:33
  /Users/zmadeira/DEV/first_sem/App_Dev_sem_1/lab2/main.py:33: LitestarWarning: Use of a synchronous callable <function provide_redis_client at 0x10cd840> without setting sync_to_thread is discouraged since synchronous callables can block the main thread if they perform blocking operations. If the callable is guaranteed to be non-blocking, you can set sync_to_thread=False to skip this warning, or set the environment variable LITESTAR_WARN_IMPLICIT_SYNC_TO_THREAD=0 to disable warnings of this type entirely.
    "redis_client": Provide(provide_redis_client),
```

-- Docs: <https://docs.pytest.org/en/stable/how-to/capture-warnings.html>

107 passed, 1 warning in 0.79s

(lab2) + lab2 git:(main) x

10. Добавление тега

Добавление тега в репозиторий

use git push to publish your local commits

The terminal shows a series of git commands used to commit changes, add a tag, and push the code to a GitHub repository. The commit message indicates a bug fix related to imports in test fixtures.

```
nothing to commit, working tree clean
(lab2) + App_Dev_sem_1 git:(main) git add .
(lab2) + App_Dev_sem_1 git:(main) x git commit -m "bug: исправлены баги с импортами в тестах сущностей связанных с отчетами"
Black..... Passed
isort..... Passed
pylint..... Passed
[main 431d541] bug: исправлены баги с импортами в тестах сущностей связанных с отчетами
  2 files changed, 4 insertions(+), 5 deletions(-)
(lab2) + App_Dev_sem_1 git:(main) git push origin main
Enumerating objects: 110, done.
Counting objects: 100% (110/110), done.
Delta compression using up to 10 threads
Compressing objects: 100% (75/75), done.
Writing objects: 100% (75/75), 224.17 KiB | 18.68 MiB/s, done.
Total 75 (delta 34), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (34/34), completed with 27 local objects.
remote: This repository moved. Please use the new location:
remote: https://github.com/2made1ra/application_development_course.git
To https://github.com/2made1ra/App_Dev_sem_1.git
  fa7150f..431d541 main -> main
(lab2) + App_Dev_sem_1 git:(main) git tag lab_8
(lab2) + App_Dev_sem_1 git:(main) git tag -l
(lab2) + App_Dev_sem_1 git:(main) git push origin lab_8
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote: This repository moved. Please use the new location:
remote: https://github.com/2made1ra/application_development_course.git
To https://github.com/2made1ra/App_Dev_sem_1.git
 * [new tag]      lab_8 -> lab_8
(lab2) + App_Dev_sem_1 git:(main)
```

98% to generate command

Вопросы

1. Что делает Брокер (Broker) в системе TaskIQ?

Брокер в системе TaskIQ является критически важным компонентом, который отвечает за передачу сообщений между клиентской частью приложения и воркерами. Каждый брокер должен реализовывать абстрактный класс `AsyncBroker` из `taskiq.abc.broker`, который определяет два основных метода: `kick` (или `publish`) для отправки задач и `listen` для прослушивания новых задач. Брокер взаимодействует с очередями сообщений (например, RabbitMQ, Redis, ZeroMQ), обеспечивая надежную доставку задач от отправителя к получателю. Когда клиент вызывает метод `.kiq()` на задаче, брокер публикует сообщение в очередь, а воркеры, подключенные к тому же брокеру, получают эти сообщения через метод `listen()` и выполняют соответствующие задачи.

Брокеры также могут иметь дополнительные методы `startup()` и `shutdown()` для управления жизненным циклом соединений, а также методы `ack()` и `reject()` для подтверждения или отклонения обработки сообщений. Это позволяет реализовать различные стратегии обработки ошибок и гарантировать доставку сообщений. TaskIQ поддерживает множество типов брокеров: `InMemoryBroker` для разработки, `AioPikaBroker` для RabbitMQ, `RedisBroker` для Redis и другие, что обеспечивает гибкость в выборе инфраструктуры.

2. Что делает Планировщик (Scheduler)?

Планировщик (Scheduler) в TaskIQ отвечает за автоматический запуск задач по расписанию. Он работает с источниками расписаний (`ScheduleSources`), такими как `RedisScheduleSource`, которые хранят информацию о том, когда и какие задачи должны быть выполнены. Планировщик постоянно отслеживает эти источники и в нужное время отправляет задачи в брокер для выполнения воркерами. Задачи могут быть запланированы двумя способами: через cron-выражения (например, `"*/5 * * * *"` для выполнения каждые 5 минут) или через указание конкретного времени выполнения.

Планировщик запускается отдельным процессом с помощью команды `taskiq scheduler module:scheduler`, при этом одновременно должен работать только один экземпляр планировщика, чтобы избежать дублирования выполнения задач. Планировщик поддерживает динамическое добавление расписаний во время выполнения (например, через `RedisScheduleSource`), что позволяет гибко управлять задачами без перезапуска системы. Задачи могут быть определены с помощью декоратора `@broker.task(schedule=[...])` или добавлены программно через методы `schedule_by_cron()` и `schedule_by_time()`.

3. Проблемы с частым выполнением задач

Если задача `process_data` выполняется в среднем за 10 секунд, а настроена на выполнение по cron каждые 5 секунд, это приведет к серьезным проблемам в долгосрочной перспективе. Основная проблема заключается в том, что новые экземпляры задачи будут запускаться до завершения предыдущих, что приведет к накоплению задач в очереди и постоянному росту нагрузки на систему. Это может вызвать переполнение памяти, исчерпание ресурсов базы данных, блокировки и в конечном итоге деградацию производительности всей системы или полный отказ сервиса.

Для решения этой проблемы существуют несколько подходов. Во-первых, увеличить интервал выполнения cron до значения, превышающего среднее время выполнения задачи (например, до 15-20 секунд), чтобы гарантировать завершение предыдущей задачи перед запуском следующей. Во-вторых, реализовать механизм блокировки (`distributed lock`) на уровне задачи, чтобы предотвратить одновременное выполнение нескольких экземпляров одной и той же задачи. В-третьих, использовать механизм проверки состояния задачи перед ее запуском, чтобы убедиться, что предыдущая задача завершена. Также можно настроить мониторинг очереди задач и автоматически масштабировать количество воркеров при росте нагрузки.

4. Настройка cron для будних дней

Для запуска задачи каждый будний день в 9:00 утра в Екатеринбурге (UTC+5) необходимо использовать cron-выражение `"0 9 * * 1-5"`, где `0 9` означает 9:00, `* *` означает любой день месяца и месяц, а `1-5` означает понедельник-пятницу. Однако здесь возникает важный нюанс: если сервер работает по времени UTC, а Екатеринбург находится в часовом поясе UTC+5, то задача будет запускаться в 9:00 UTC, что соответствует 14:00 по времени Екатеринбурга. Для корректной работы необходимо использовать параметр `cron_offset` со значением `timedelta(hours=5)` или строкой `"+05:00"`, чтобы сдвинуть время выполнения на 5 часов вперед относительно UTC.

Подвох заключается в том, что без указания `cron_offset` планировщик будет использовать системное время сервера (UTC), и задача будет выполняться не в 9:00 по времени Екатеринбурга, а в 9:00 UTC. Это критично для бизнес-логики, которая зависит от локального времени. Правильная настройка будет выглядеть так: `schedule_by_cron(redis_source, "0 9 * * 1-5", cron_offset=timedelta(hours=5))` или через декоратор: `@broker.task(schedule=[{"cron": "0 9 * * 1-5", "cron_offset": timedelta(hours=5)}])`. Также важно учитывать переход на летнее/зимнее время, если это применимо к региону.

5. Масштабирование и отказоустойчивость

Если количество задач резко выросло и один воркер не справляется, для масштабирования системы на базе TaskIQ необходимо запустить дополнительные экземпляры воркеров на разных процессах или серверах, подключенных к тому же

брокеру. Все воркеры будут автоматически распределять задачи между собой, так как брокер (например, RabbitMQ или Redis) обеспечивает механизм конкурентного потребления сообщений из очереди. Систему можно горизонтально масштабировать, запуская воркеры на разных машинах или в контейнерах, при этом необходимо использовать брокеры, поддерживающие распределенную работу (не InMemoryBroker, который работает только в рамках одного процесса).

Если откажет один из запущенных воркеров, задачи, которые он обрабатывал, не будут потеряны благодаря механизму подтверждения (acknowledgment) в брокере. Неподтвержденные задачи будут возвращены в очередь и автоматически перераспределены между оставшимися воркерами. Это обеспечивает отказоустойчивость на уровне воркеров. Если откажет планировщик, уже запланированные задачи продолжат выполняться по расписанию, если используется распределенный источник расписаний (например, RedisScheduleSource), который хранит расписания независимо от процесса планировщика. Однако новые задачи не будут добавляться в расписание до восстановления планировщика. Для повышения отказоустойчивости можно запустить резервный планировщик или использовать механизм leader election для автоматического переключения на резервный экземпляр.