Django Celery Workers Project Structure

Django with Celery for background task processing. Includes Redis/RabbitMQ configuration patterns.

#django #python #celery #async #backend

PNG PDF Copy Prompt

Project Directory

```
myproject/
    manage.py
  config/
    __init__.py
    celery.py  Celery app conf…
  settings/
      __init__.py
      base.py
      local.py
      production.py
    urls.py
    wsgi.py
    asgi.py
  apps/
    users/
      __init__.py
      admin.py
      apps.py
      models.py
      views.py
      urls.py
      tasks.py  User-related as…
    migrations/
        __init__.py
    notifications/  Email, SMS, push
      __init__.py
      apps.py
      tasks.py  Notification de…
      services.py  Email/SMS provi…
    templates/
      notifications/
        email/
    reports/  Heavy computati…
      __init__.py
      apps.py
      models.py
      views.py
      tasks.py  Report generati…
      generators.py  Report logic
    migrations/
        __init__.py
    __init__.py
  taskqueue/  Celery infrastr…
    __init__.py
    base.py  Base task class…
    schedules.py  Beat schedule d…
    routing.py  Queue routing r…
    signals.py  Task success/fa…
    monitoring.py  Task metrics he…
  templates/
    base.html
  requirements/
    base.txt
    local.txt
    production.txt
  docker-compose.yml  Django + Redis …
  .env.example
  .gitignore
  pytest.ini
```

Why This Structure?

Tasks live in each app's `tasks.py` for discoverability. The `taskqueue/` folder centralizes Celery infrastructure: base task classes, schedules, routing, and monitoring. This keeps app code focused on business logic while Celery config stays organized.

Key Directories

`config/celery.py` - Celery app instantiation and autodiscover

`apps/{app}/tasks.py` - App-specific async tasks

`taskqueue/schedules.py` - Celery Beat periodic schedules

`taskqueue/routing.py` - Queue routing by task type

Task Organization

`notifications/tasks.py` - Emails, SMS, push—async delivery

`reports/tasks.py` - Heavy computation, PDF generation

`users/tasks.py` - Onboarding flows, data sync

Getting Started

1. `pip install celery redis`
2. Configure `CELERY_BROKER_URL` in settings
3. Import celery app in `config/__init__.py`
4. Run: `celery -A config worker -l info`
5. For scheduled tasks: `celery -A config beat -l info`

Key Celery Settings

- `CELERY_BROKER_URL = "redis://localhost:6379/0"`
- `CELERY_RESULT_BACKEND = "redis://localhost:6379/0"`
- `CELERY_TASK_ALWAYS_EAGER = True` `# For testing`
- `CELERY_TASK_ROUTES = "taskqueue.routing.route_task"`
- `CELERY_BEAT_SCHEDULE = schedules.CELERYBEAT_SCHEDULE`

Docker Services

`django` - Web server (Gunicorn)

`celery_worker` - Task consumer (-Q default,high)

`celery_beat` - Periodic task scheduler

`redis` - Message broker and result backend

`flower` - Optional: Celery monitoring UI

Best Practices

- Pass IDs, not objects—refetch in task
- Use `bind=True` for access to `self.retry()`
- Set reasonable `time_limit` and `soft_time_limit`
- Use `countdown` / `eta` for delayed execution
- Group related tasks with chains/groups

Trade-offs

`Debugging` - Async failures harder to trace than sync

`Testing` - Need `CELERY_TASK_ALWAYS_EAGER` or mocking

`Infrastructure` - Requires Redis/RabbitMQ and worker processes