1 слайд. Приветствие:

* Добрый день, дамы и господа. Тема моей презентации - …

2 слайд. Структура презентации:

* Итак, о чём сегодня пойдёт речь.
* Я вкратце расскажу про django-celery-beat и для чего он предназначен. Затем мы рассмотрим пример с реальной проблемой бизнеса. В конце мы подведём итоги и вы зададите вопросы. Вопросы по ходу презентации допускаются
* Цель демонстрации - познакомить вас с концептом шаблона проектирования, осветить теоретические аспекты шаблона и показать, как его можно реализовать на практике.
* Примечание: последний раз я заглядывал в Celery около года назад, поэтому детали реализации уходят на второй план. В своей презентации я сделаю упор на основные используемые механики, о которых стоит знать при реализации шаблона на Django.

3 слайд. Демо celery-beat:

* CeleryBeat.settings: INSTALLED\_APPS. Такой пакет называется scheduler (или шедулер по-русски)
* Расписания на примере админки:
  + Кронтабы - формат данных, предназначенный для установки дефолтных расписаний
  + Астрономические события - можно привязаться к астрономическому событию и запускать таски, к примеру, на рассвете в указанной локации
  + Таймстампы - всё очевидно
  + Интервалы - всё очевидно. Создаём расписание, которое будет отрабатывать каждые N периодов
  + Подключение зарегистрированных задач по созданным расписаниям

Это всё цветочки, нам же интересны ягодки.

4 слайд. Проблема бизнеса:

* У бизнеса есть два готовых сервиса - один полумёртвый, другой сырой. Полумёртвый сервис приносит деньги, сырой ест ресурсы с перспективой крупного расширения. Оба решают идентичную задачу - обеспечивают торговлю на разрабатываемой площадке.
* Бизнес хочет, чтобы в сервисах был одинаковый набор товаров, что невозможно по техническим причинам. Тем временем у пользователей-продавцов пригорает на обеих площадках, потому что цены и остатки разъезжаются, что влияет на прибыли каждого из стейкхолдеров.
* Задача - обеспечить мониторинг расхождений, чтобы поддержка могла объяснить, почему что-то где-то не обновилось

5 слайд. Техническая проблема:

* Структуры БД двух сервисов существенно различались, что исключало сравнение объектов напрямую
* Рекомендация проверить все товары на лету завела меня в тупик. Обработка была на чистых SQL-запросах через ORM, которые потом преобразовывались в словари, что сразу показало на ключевой недостаток обработки на лету - длительное время обработки и большие потери времени на запросы к базе.
* Так как оба сервиса находились в разработке, постоянно менялся состав атрибутов. При изменениях состава атрибутов офферов нужно было менять логику обработки, что делало код неподдерживаемым
* Итого, первое решение получилось неподдерживаемым

6 слайд. Связанность систем. Зацепление:

* Пройдёмся по определениям
* Связанность - мера монолитности программного компонента. Чем более монолитен компонент, тем его проще понять, переиспользовать и поддерживать. Очевидно, что, если компонент описывает что-то одно, то в этом проще разобраться. Высокая связанность внутри одного компонента есть хорошо.
* Зацепление - мера независимости программного компонента. Низкое зацепление компонентов говорит о хорошо спроектированной и структурированной разрабатываемой системе - компоненты достаточно обособлены и не влияют друг на друга. В противном случае система превращается в лабиринт Минотавра, реализуя шаблон “Большой ком грязи”.
* Если говорить в обозначениях SOLID, то:
  + Согласно Single Responsibility Principle, у компонента должна быть одна и только одна причина для изменения. Если два компонента системы организованы таким образом, что изменение в одном порождает изменение в другом, то два таких компонента называются связанными.
  + Согласно Open-Closed Principle, система должна быть открыта для расширения и закрыта для модификации. Это, с одной стороны, значит, что логику хорошо реализованного модуля трудно и бессмысленно менять - она уже железная и ей негде обвалиться. С другой стороны, любой желающий должен суметь свободно расширить существующую логику.
* Главный недостаток первого решения заключается именно в том, что оно пыталось связать две изолированных системы

7 слайд. Шаблон “Посредник”:

* Один из известных шаблонов проектирования от банды четырёх. Классифицируется как поведенческий, потому что влияет на взаимодействие компонентов друг с другом
* Позволяет инкапсулировать логику взаимодействия компонентов, не привязывая их напрямую. Это уменьшает созависимость компонентов
* Общий подход к реализации прост - создать компонент-посредник, на который будет делегирована логика взаимодействия, пока взаимодействующие компоненты играют роль контейнеров
* Итак, шаблон поможет нам сконцентрировать логику обработки в одном месте, не привязываясь к реализациям внешних баз.

8 слайд. Реализация в виде микросервиса:

* Архитектура проекта - MVC с асинхронными контроллерами. БД подключаются через ORM, транспорт обеспечивается Celery, результаты выводятся на UI.
* Поддержка множества БД в проекте Django:
  + Django позволяет поддерживать множество соединений с БД в одном проекте. Реализуется достаточно тривиально
  + Модели подключаемых баз.
    - Чтобы полностью исключить возможность изменений подключаемых БД, прописываем managed=False класса Meta. Это скажет Django, что база контролируется внешними силами
    - Чтобы пробросить подключаемые модели в админку, копипастим из документации класс админки, наследуем от него админки для М2 и RM и регистрируем модели
  + Модели посредника
    - Так как необходимо сравнивать атрибуты одних и тех же сущностей с разной реализацией, удобно создать абстрактную модель, которая будет включать общие для реализаций поля.
    - Прописываем абстрактную модель, от которой наследуем конкретные несостыковки. Делим модель результатов на 2 подрезультата, т.к. несостыковки могут быть по одну сторону посредника. Дописываем дополнительные поля, уникальные для реализаций. Собираем результат в общую модель посредника
  + Маршрутизаторы баз данных
    - В текущем виде для каждого запроса на внешнюю базу необходимо прописывать using. Чтобы не запинаться об это, пропишем маршрутизаторы баз данных. В таком виде запросы к моделям подключаемых баз будут автоматически перенаправляться на нужную базу
* Инкапсуляция логики обработки
  + Объявляем асинхронный контроллер. Он будет собирать офферы с обеих баз, проводить сравнение офферов и записывать результаты анализа в базу посредника.
  + Подключаем контроллер к расписанию
  + Profit

9 слайд. Итоги:

* django-celery-beat позволяет запускать задачи по разнообразным расписаниям и не ограничивает разработчика в выборе инструмента
* Шаблон “Посредник” уменьшает зацепление между компонентами, не влияя на связанность логики внутри компонентов или же на связанность компонентов друг с другом. Это позволяет крутить компоненты по типу шарнирной связи - шарнир стоит, как есть, а компоненты двигаются свободно