

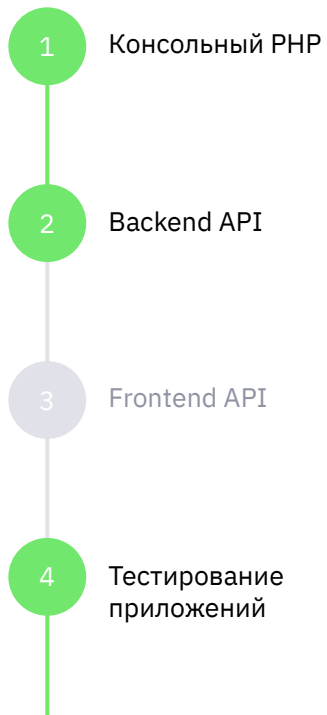
Очереди в РНР

Урок 7











План курса



WELCOME



Что будет на уроке сегодня

-  Викторина, которая построена на основании реальных вопросов, которые задают на собеседовании
-  Имитация работы выполнения заданий от тимлида
-  Опыт постановки ТЗ от тимлида
-  Опыт работы с обменниками RabbitMQ
-  Опыт вынесения функционала в очереди
-  Опыт работы с очередями в PHP



Викторина

Построена на основании реальных вопросов,
которые задают на собеседовании



Преподаватель демонстрирует вопросы викторины, зачитывает их, а студенты пишут ответы в чате

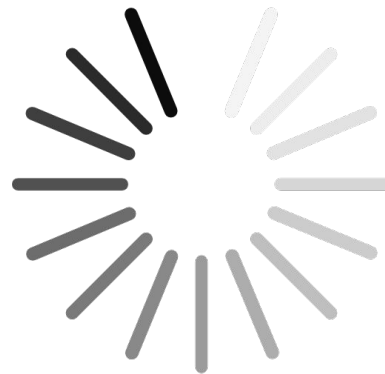


1. Что такое очереди?

1. Структура данных, представляющая упорядоченный набор элементов, где новые элементы добавляются в конец очереди, а элементы удаляются из начала очереди
2. Центральный распределительный центр для маршрутизации сообщений (хаб) между производителями и потребителями в RabbitMQ
3. Инструмент, который передает сообщения от отправителя к получателю, храня их в очереди

Дополнительный вопрос:

В каких случаях используются очереди?





2. Что такое очереди сообщений?

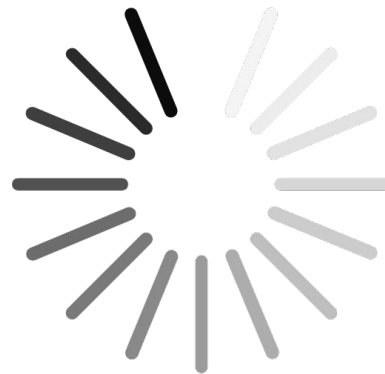
1. Способ обмена сообщениями между сервисами, который работает асинхронно

2. Является принципом, а не конкретной технологией или стандартом

3. Приложение, которое принимает и обрабатывает сообщение

Дополнительный вопрос:

Как они работают очереди?



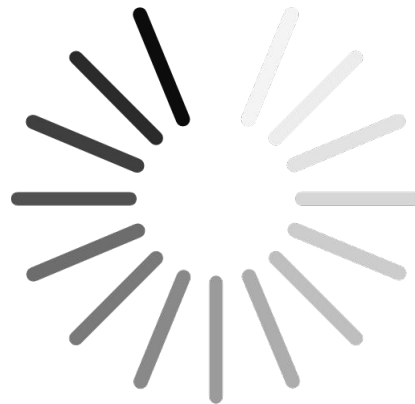


3. В каких случаях очереди сообщений неэффективны?

1. Простая архитектура

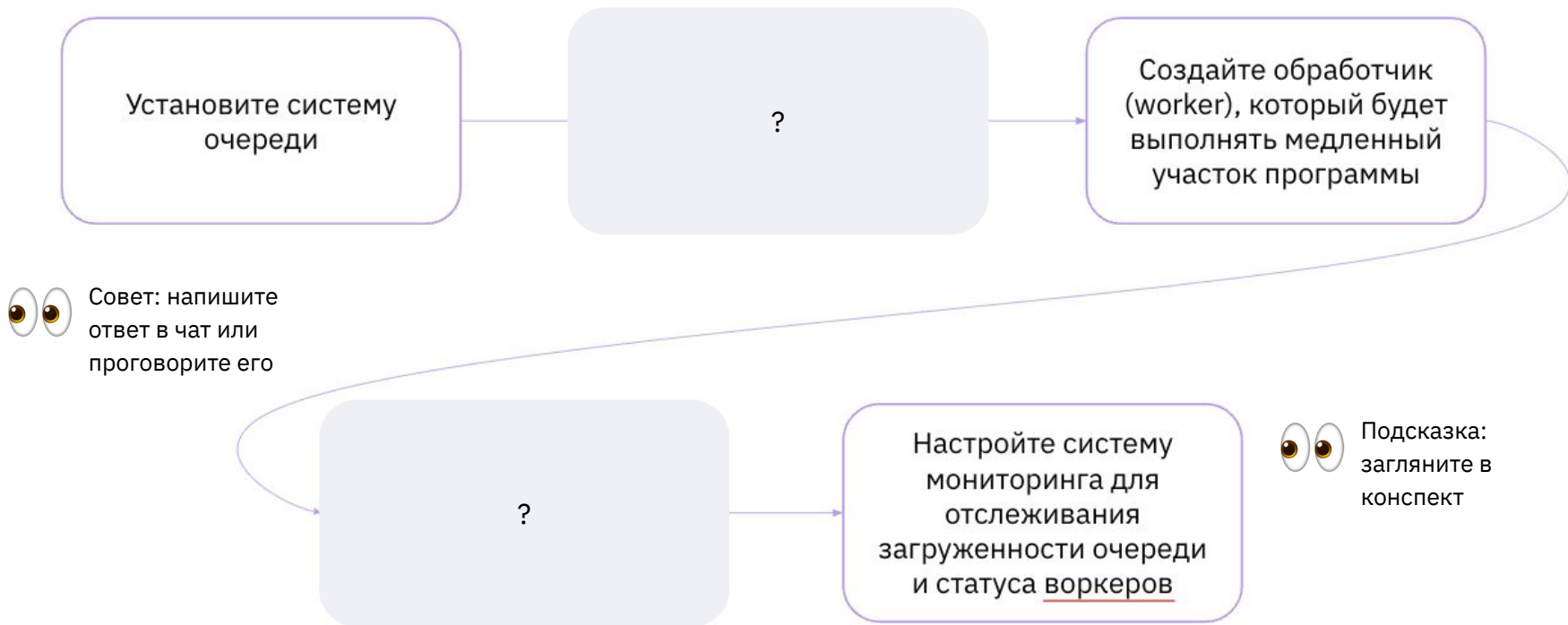
2. Монолитное приложение

3. Сложная архитектура





4. Перед вами представлена последовательность того, если бы вы выносили медленный участок кода в очереди сообщений. Посмотрите внимательно, какие элементы процесса скрыты?

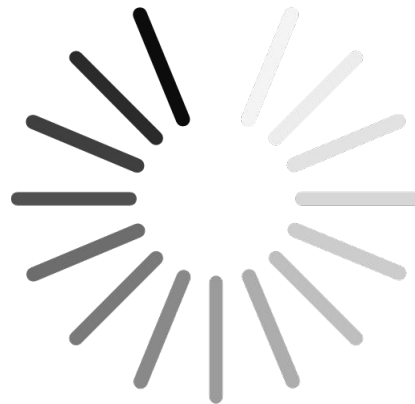




5. Если нам нужна надежность и гибкость маршрутизации, но порядок доставки сообщений не так важен, то выбираем ...

1. RabbitMQ

2. Apache Kafka

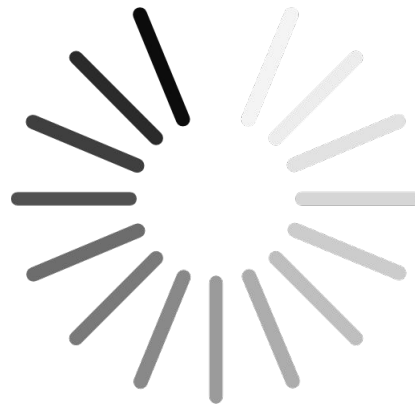




6. Если мы планируем пропускную способность до 1 миллиона сообщений в секунду, то выбираем ...

1. RabbitMQ

2. Apache Kafka





7. Опишите понятия Producer/Publisher, Consumer/Subscriber и Queue. Как они связаны?

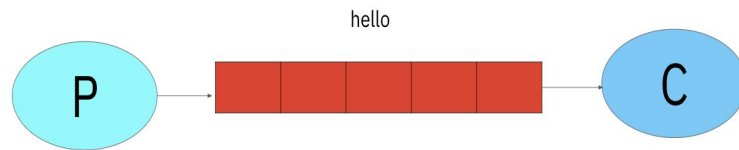
Вопрос без вариантов ответа



Подсказка: загляните в конспект



Совет: напишите ответ в чат или проговорите его



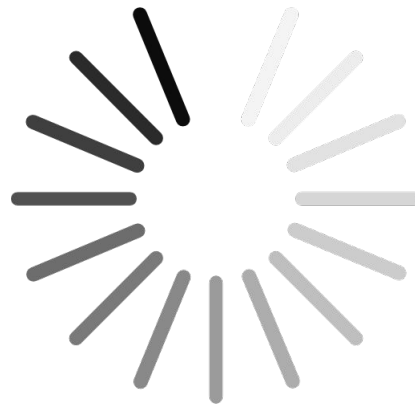


8. Что такое Exchange?

1. Точка обмена валюты

2. Корпоративная служба

3. Маршрутизация сообщений





9. Какие типы Exchange существуют? Опишите их.

1. Разветвление

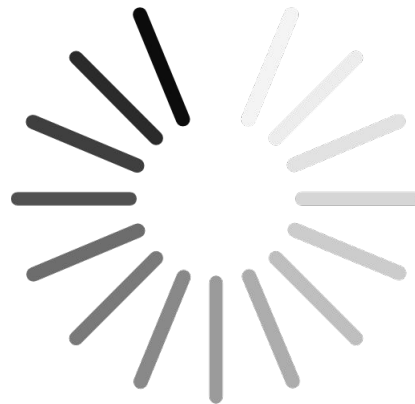
2. Прямая связь

3. Обратная связь

4. Тематическая

5. Заголовочный

6. Обменный





Перечень вопросов, которые также могут также задать:

1. В каких сценариях могут использоваться очереди сообщений?
2. Что такое Apache Kafka? В каких случаях он эффективней RabbitMQ, а в каких наоборот?
3. Что такое RabbitMQ? Для чего используется?
4. Как обеспечить надежность и эффективность системы при использовании RabbitMQ?
5. Чем отличаются схемы независимых подписчиков и конкурирующих подписчиков? Возможен ли смешанный вариант?
6. Для чего используется Exchange? Как Exchange принимает решение, что делать с сообщением?

Рекомендую на них ответить самостоятельно
Ответы к ним вы найдете в лекции 🙌



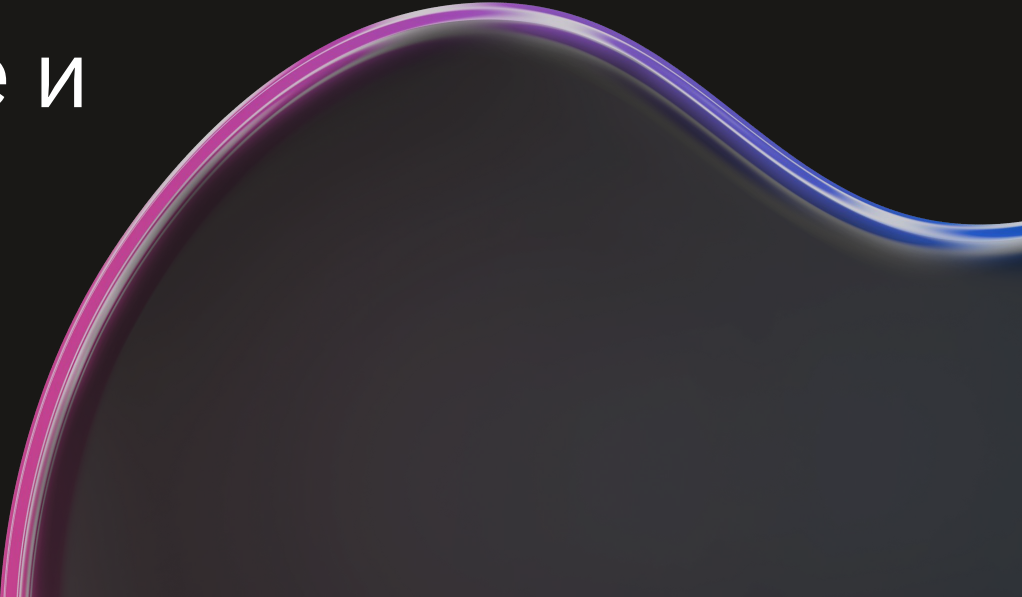


Разбор домашнего задания





Кому удалось улучшить
свое приложение и
каким образом?





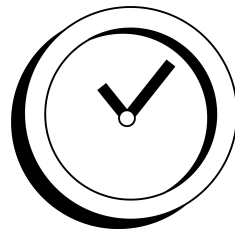
Практика





Задание 1. Запускаем RabbitMQ и используем различные типы Exchange

Предлагаю поработать с RabbitMQ из консоли и посмотреть на основные операции работы с различными типами данных.



10 мин.

Что потребуется на уроке?



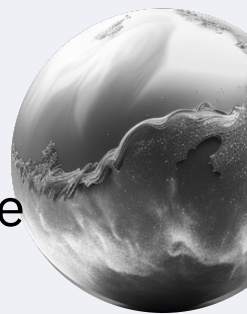
Установить RabbitMQ по [инструкции](#)



Наше приложение хорошо работает и обеспечивает все нужды: LA (нагрузка на процессор) уменьшился!

Но поток сообщений большой, а API Telegram настолько ненадежный, что часто при отправке сообщений происходят ошибки, отчего теряются напоминания.

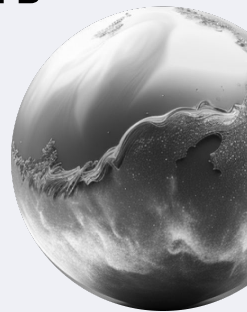
Такая же проблема наблюдается, если скрипту приходит много сообщений одновременно — он обрабатывает их последовательно и, если в 10-м сообщении возникает исключение, то остаются необработанными все входящие сообщения.





Это невозможно терпеть, поэтому вы вместе с тимлидом решаете вынести механизм отправки сообщений в очереди.

Так вы сможете избежать проблем при исключениях во время отправки сообщений и, в случае исключений на стороне API Telegram, вы просто сможете снова обработать событие.





Задание 2. Разбор ТЗ с тимлидом

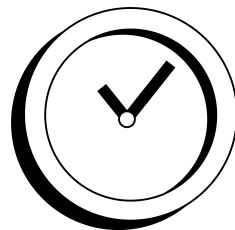
Для работы с RabbitMQ мы будем использовать библиотеку **php-amqplib**.

Вместе с тимлидом определяем порядок разработки. Алгоритм:

1. Создать адаптер, который будет соединяться с RabbitMQ, отправлять, получать сообщения и помечать их прочитанными. Так как предполагается, что обработчик будет работать в 1 поток, то стоит подумать о многопоточности и о том, как делать сложный ask сообщений не нужно. Для этого предлагается реализовать интерфейс.
2. Переделать EventSender таким образом, чтобы он не отправлял сообщения сразу, а сохранял их в очередь, после чего сообщения будут отправляться в фоне обработчиком очереди. Для этого введем интерфейс Queueable.

Не забудьте поменять классы, конструирующие EventSender(EventRunner и TgEvents), добавив создание класса Queue (в нашем случае реализации RabbitMQ)

3. Создайте обработчик, который будет проходить по очереди и обрабатывать сообщения.



15 мин.



Задание 3. Создаем адаптер для работы с RabbitMQ

Создать адаптер, который будет соединяться с RabbitMQ, отправлять, получать сообщения и помечать их прочитанными. Так как предполагается, что обработчик будет работать в 1 поток, то стоит подумать о многопоточности и о том, как делать сложный аск сообщений не нужно.

Для этого предлагается реализовать интерфейс

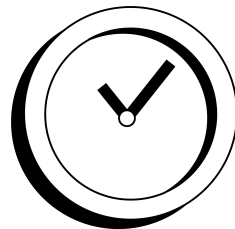
```
<?php

namespace App\Queue;

interface Queue
{
    public function sendMessage($message): void;

    public function getMessage(): ?string;

    public function ackLastMessage(): void;
}
```



10 мин.

Задание 4. Переносим обработку EventSender в очереди



5 мин.



Переделать EventSender таким образом, чтобы он не отправлял сообщения сразу, а сохранял их в очередь, после чего сообщения будут отправляться в фоне обработчиком очереди. Для этого введем интерфейс Queueable

```
//Переделать EventSender таким образом, чтобы он не отправлял сообщения сразу, а
сохранял их в очередь, после чего сообщения будут отправляться в фоне обработчиком
очереди. Для этого введем интерфейс Queueable
<?php
```

```
namespace App\Queue;
```

```
interface Queueable
```

```
{
```

```
    public function handle(): void;
```

```
    public function toQueue( ... $args):void;
```

```
}
```

```
//Не забудьте поменять классы, конструирующие EventSender(EventRunner и TgEvents),
добавив создание класса Queue(в нашем случае реализации RabbitMQ)
```

```
$queue = new RabbitMQ('eventSender');
```

```
$eventSender = new EventSender(new TelegramApiImpl($this->app-
>env('TELEGRAM_TOKEN')), $queue);
```

```
//После чего реализуем его:
<?php
```

```
namespace App\Actions;
```

```
use App\Queue\Queue;
```

```
use App\Queue\Queueable;
```

```
use App\Telegram\TelegramApi;
```

```
class EventSender implements Queueable
{
```

```
    public function __construct(
        private TelegramApi $telegram,
        private Queue       $queue
    )
```

```
    {
```

```
    {
```

```
    }
```

```
...
}
```



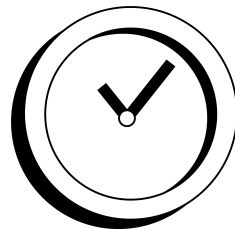
Задание 5. Создаем обработчик сообщений очереди

Создайте обработчик, который будет проходить по очереди и обрабатывать сообщения:

```
<?php
namespace App\Commands;

use App\Application;
use App\Queue\Queueable;
use App\Queue\RabbitMQ;

class QueueManagerCommand extends Command
{
    protected Application $app;
    public function __construct(Application $app)
    {
        $this->app = $app;
    }
    public function run(): void
    {
        while (true) {
            // TODO реализовать получение сообщений
            // из очереди и их запуск
            sleep(10);
        }
    }
}
```



5 мин.

Запустить можно командой:

```
php8.2 runner -c queue_manager
```



Домашнее задание

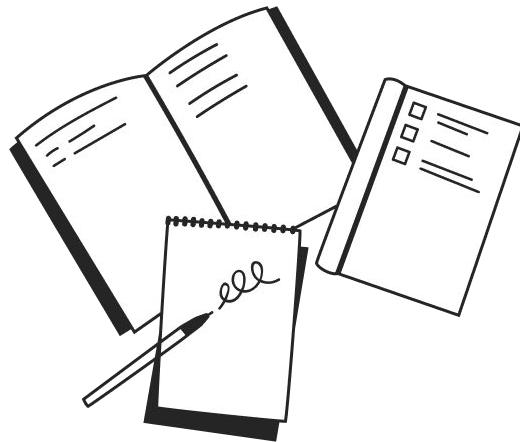




Домашнее задание

Задание

1. Сделайте новую ветку из той, которую мы создали на прошлом уроке. Это нужно для того, чтобы мы могли работать с кодом из прошлого урока.
2. Загрузите весь код из сегодняшнего урока в Git в новую ветку, создайте новый pull request. Пришлите на проверку ссылку на pull request.
3. Завершайте работу над проектом. Доделайте прошлые домашние задания.
4. ✖ Улучшайте проект, чтобы им можно было гордиться!



В качестве решения приложить:

- ссылку на pull request в вашем репозитории с домашним заданием

✖ – дополнительное задание 💪











Подведем итоги





Подведение итогов

-  потренировались отвечать на вопросы про очереди на собеседовании
-  научились работать в команде, понимать требования и выполнять задачи, поставленные тимлидом
-  научились разбивать задачу на более мелкие подзадачи, чтобы легче было управлять проектом и следить за прогрессом
-  узнали, как работать с RabbitMQ из консоли
-  узнали, как работать с обменниками в RabbitMQ
-  узнали, как работать с RabbitMQ в PHP
-  научились выносить отдельные функции приложения в очередь
-  научились писать адаптеры работы с очередями



**Поздравляю с окончанием курса
«Вокруг PHP – экосистема веб-приложений»!**





Спасибо за внимание