Time	~ 2 - 3 hours
Learning Goals	• Learn how background jobs work • Use Sidekiq to process background jobs

PairBnb - Deliver Later (Workers)

Do you guys notice that your Booking page becomes much slower after implementing email sending? We've built a mail delivery system that immediately delivers a booking email to our hosts.

This may not be a huge issue if we only have one email being sent once at a time. Since we rely on external services to send our email, imagine what if your SMTP service provider (Gmail, Mandrill, Sendgrid) was going through some issues and became very slow? What if I got 15 customers using at the same time?

That's not ideal at all. We need to offload some work to the background and immediately tell the user to wait for their email instead of setting them waiting for a respond for my app. This is where Background Jobs come in handy!

In this challenge, our goal is simple:

```
We want to create a background job to send email asynchronously
```

To achieve this, we're going to need Sidekiq and Redis Server. Let's work in pairs so **let's** find a partner among your peers.

Concept

The concept of background jobs exists in the real world! To help you understand further, let's imagine a restaurant with only the chef working. As soon as you get to a restaurant, your goal is to place your order, sit down, relax and wait for your food to come.

If there was only the chef, as soon as you place your order, the chef has to go to the kitchen, cook and leave the front of the restaurants unsupervised. New patrons coming to the restaurants will sit at their tables frustrated, waiting to place their order.

Now what if there were 2 people working at the restaurant, a chef and a waitress? The waitress will take orders and send them to the chef. New patrons coming in will be able to place their orders without being left hanging! Indeed, a much better experience!

Back to web development, think of Rails as your waitress and your background job as your chef!

Rails will take commands and return what is necessary immediately. The background job will take what can be done later (but still important) and do it behind-the-scenes.

This is not the only use case of background jobs. Background jobs also widely use in many areas, particularly in long processing background operations. It's particularly important in cases like Youtube. When you upload a video to Youtube, Youtube will do post processing on your video (compressing, decoding etc) on the spot. Imagine if post-processing was also done by Rails (your frontend worker)? How slow will your upload process be!

---

1) Configure the Tools

Sidekiq

Sidekiq is a Redis-based queue that allows your app server to push a task into a queue that a background worker processes. This takes a few milliseconds and frees up the app server to respond to other requests.

Let's update our mailer to send emails asynchronously with Sidekiq.

Add the `sidekiq` gem to your `Gemfile` and run `bundle install`.

Redis

Redis is not a Gem. Redis like, Postgres is a database. We use Redis to store jobs. Imagine the restaurant scenario again.

If the waitress merely tells the chef what he has to cook, he will surely not remember many of them and end up cooking the wrong thing or not cooking something someone ordered.

So the waitress decided to write down the order on a piece of paper and clip the paper in an ordered manner (first come first serve) on the chef's table.

Now that's a queue system. Redis is the storage infrastructure we use to connect our background worker and our Rails app. Rails will insert a task to the queue, the background worker will pick up the task from the queue (Redis).

The installation is different from Mac to Ubuntu.

On Mac:

```
$ brew install redis
```

On Ubuntu:

```
$ sudo apt-get install redis-server -y
```

Question to discuss with your partner:

1. Are Redis and Sidekiq installations that simple?
2. Should we do some research on how to configure both of them?

---

2) Make a Feature Branch

As usual, let's make a feature branch on git and switch to it for development.

---

3) Create a Job

It's time to code. Now, let's create a job in Rails. You might need to read up this page to get some knowledge.

If everything goes smoothly, you should get something like this:

```
# app/jobs/reservation_job.rb


class ReservationJob < ActiveJob::Base
  queue_as :default


  def perform(...)
    ...
  end
end
```

---

4) Configure the Job to Send Email

Okay, now we need to configure the job to acutally send an email. Since this is an independent process, it's better for you to plan out your data path. Example,

1. Should the job sources the information on its own?
2. How can I pass parameters to the job for operations?

Discuss with your partner. Perform research if needed.

---

5) Update Reservation Controller with Background Job

Now that we have a job for sending email, we should use it in our controller.

We do notice that job shares similarity with Mailer, they're like Model classes. Hence, to use them is similar to Mailer but a different class.

Proceed with your upgrade. If all goes well, it should look something as follows:

```
class ReservationsController < ApplicationController

  ...

  def create
    ...
    if @reservation.save
      ReservationJob.perform_later(...)
    else
      ...
    end
  end
```

```
  ...
end
```

---

6) Setup Sidekiq, Redis and Rails

All right, now that we're all good, let's proceed to run Redis, Sidekiq and Rails.

Since Rails is the waitress, Sidekiq is a chef, and ticket is a system, we'll need 3 different processes, hence 3 terminals.

To bring it up, let's bring the servers up in sequence:

Terminal Session 1

```
$ redis-server
```

Terminal Session 2

```
$ rails s
```

Terminal Session 3

```
$ bundle exec sidekiq -q default -q mailers
```

Give it a try.

---

7) Integrate with ActiveJob?

Recently, Rails finally supported background job with what is know as ActiveJob. ActiveJob allows direct integration with Sidekiq. However, in the past, community uses Sidekiq to direclty creates background jobs. More information: https://github.com/mperham/sidekiq/wiki/Active-Job#action-mailer

The questions to be discussed with partner and mentor:

```
How should we integrate Sidekiq? Into ActiveJob or directly?
```

Others

- Consider doing a research on `sucker_punch` and `delayed_job`. These are famous workers including `sidekiq`.
- Besides the mailer, discuss with your peers and mentor on what other work can be moved to be a background job (for example, auto-tweets, report generation, etc).

Once done and your email is sending while your Rails doesn't perform slowly, congrats! You've done it! You can proceed to commit the changes, raise pull request and merge back to master branch.