Backend Coding Challenge

Hey Otacilio!

First of all, thanks for your interest in working at Outfittery!

We'd like to propose a coding challenge to you. The purpose of this challenge is to:

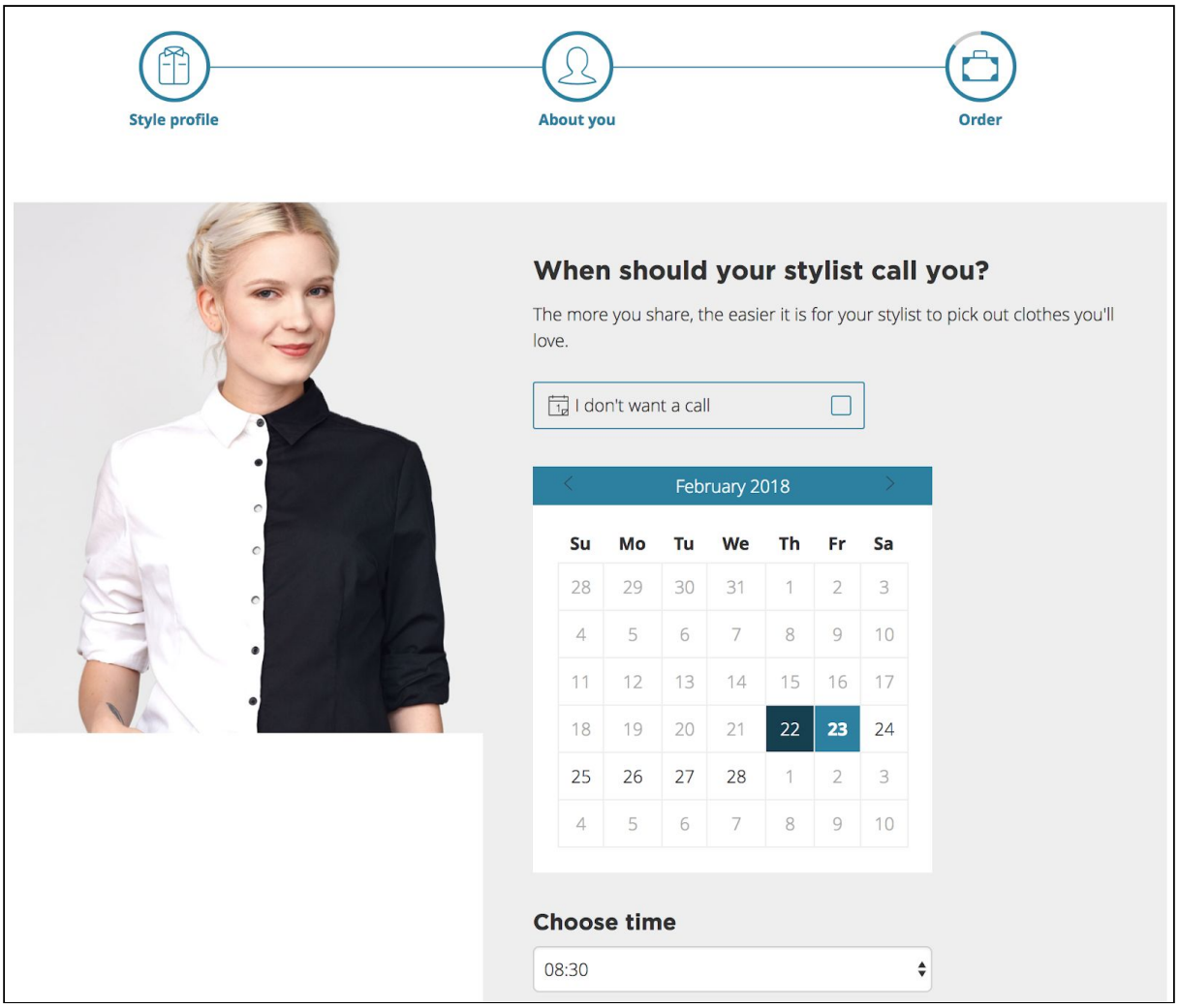
1. enable you to have a basic understanding of a part of our product;
2. exercise modeling of the domain given only non-formal description;
3. let you demonstrate your understanding of API design principles and coding skills;

A little bit of context: Booking a call with a stylist

If you have tried our product already and tried to order a box, you might have been asked if you'd like to book a phone call with a stylist.
The idea is to have a better understanding of what you're looking for and how to pack the best possible box for you.

Once our customer wants to have a chat with a stylist we book it as an appointment for both of them (stylist and customer).
Since we have limited number of Stylists, and many Customers want to use their service, the available time-slots are a scarce resource that have to be managed.

This is how it looks like from Customer’s perspective:



Important thing to notice here is a date and a time.

For this page to be displayed, we need an endpoint behind the scenes that will check the availability of our 150+ stylists and offer our customers their available time slots.

The task

We would like you to implement a backend application with RESTful API.
The application should offer a way to:

- add a Stylist
- add a Customer
- list available time-slots for appointments
- book an appointment in a time-slot of choice

Domain overview:

- **Customers and stylists are unique and characterized by the ID.**
  Adding properties like ***name*** and/or ***email*** into your model is advised, that will make it feel like a real thing and will help to prevent doubles.
- **Each customer can be handled by any stylist.** API does not allow to choose a particular Stylist to be assigned to the appointment with a Customer. Any of **available** Stylists will be picked.
- **Appointment** describes a connection between: Customer, Stylist and particular time-slot.
- **Stylist with an appointment can't serve any other customer in the same time-slot** - he/she is booked.
- **Time-slots are of equal length and start at fixed intervals.**
  Time-slot has a date with time, think of it as an event in calendar.
  Start with these assumptions:
    - each time-slot is 30 minutes long
    - there are 16 of them in the day
    - the first one starts at 9:00
    - stylists work 7 days a week (ouch!) so you don't have to worry about weekends

  Good time-slot model would allow easy future adjustments like change of slot length and change of number of slots per day.
  Feel free to ask us for details if you find it hard to find a balance between flexibility and simplicity.
- **Time-slot** may be implemented as an entity in the database, or just something that is determined and computed by counting existing stylists and appointments. It's up to you. Both solutions have their merits.
- **When to initialize available time-slots?** - good question!
  You have many valid options here. It might be a dedicated API endpoint - something to explicitly initiate free slots in calendar of stylist S for next X days starting from now (just an example).
  Another valid option is to look for a way to avoid a notion of **time-slot** entity as suggested in previous bullet point - so you will not have to initialize the

collection of free slots at all.
Again, both approaches come with challenges of their own.

- **Stylists availability** - it's enough if customers see availability in the next X days only; so if you decide to crete **time-slot** database entity then you only generate records for X days in advance.
- **Time-slot** is no longer offered to the customer only if **all** stylists are already booked at this time.

Features requirements:

- API endpoints for:
  - add Customer
  - add Stylist
  - list free time-slots for appointments
  - book an appointment for customer in the time-slot of his choice
  - (optional - implementation dependent) initialize free time-slots for the stylist
- Appointment creation service respects available resources (free and busy slots in stylists calendars) and reacts to obvious corner cases.

Random hints, and things to keep in mind:

- Transactions. We're talking about an Internet business, hence between a customer visualizing a time slot and confirming the selected time slot, another customer might have already taken that particular slot.
- Security, authentication, authorization - skip it entirely.
- Some kind of in-memory database is a sane choice. H2 is ok.
- It's a single application. Resist the temptation to create multiple microservices.
- Your code should be elegant and structured.
- We expect to see some tests. There is no specific test coverage to achieve. Just a few tests methods are fine as long as they test something important.
- RESTful APIs - there are not that many standards out there, but you know some generally accepted conventions and good practices for sure. Please use them.
- In case you run out of time, we'd like to know what was your overall plan, what did you leave out, why did you leave that out and what did you plan to do next; prioritizing what gets done and what's left out is also part of the task.
- Our description of Domain is not very formal. Use your best educated guess to make it work. We leave the details to you. Don't hesitate to ask.
- You will often ask yourself if you have right balance between simplicity and realism. One good idea is to think of this application as a proof of concept that you will polish later. It IS a model, but you should avoid obvious *dead-end* oversimplifications.

Examples

Example 1.
Starting condition: no stylists are defined
Request “get all available time slots” returns empty list

Example 2.
Starting condition: 1 stylist is defined, no appointments are booked yet.
Request to “get all available time slots” returns a list of all possible appointments, every slot is still free:

```
{
   ...
   "availableTimeSlots": [
      "2019-01-12 09:00:00",
      "2019-01-12 09:30:00",
      "2019-01-12 10:00:00",
      ….
}
```

Next, an appointment is created on “2019-01-12 09:00:00”.
Request to “get all available time slots” returns updated list. One slot was taken, so it no longer appears in the list:

```
   ...
   "availableTimeSlots": [
      "2019-01-12 09:30:00",
      "2019-01-12 10:00:00",
      ….
```

Example 3.
Starting condition: 2 stylists are defined, appointment at “2019-01-12 09:00:00” is booked.
Request to “get all available time slots” returns a list of all possible appointments. Every slot is still free because even though there is one appointment booked, the second stylist is still available at this time:

```
   ...
   "availableTimeSlots": [
      "2019-01-12 09:00:00",
      "2019-01-12 09:30:00",
      "2019-01-12 10:00:00",
      ….
```

It would require another customer to book an appointment at “2019-01-12 09:00:00” for this slot to disappear from the list.

The technical side of solution

You are free to use any JVM-based tech stack you are comfortable with.

We suggest:

- Language: Java / Groovy / Kotlin
- Framework(s): SpringBoot + Spring-data-JPA + Spring-web-MVC
- Persistence: H2
- API Doc: Swagger
- Build: Gradle / Maven
- Tests: JUnit / TestNG / Spock / Mockito

That’s popular choice among our candidates and is very welcomed by our reviewers too.
Feel free to change it to your liking and accordingly to your skills.

It would be a nice touch if your solution is not only easy to build, but also easy to run.

If you’d like to ask questions or share your thoughts during the challenge, you can use Slack channel that will be set up for you.

By the end of your test period, you should make your solution available in a GitHub repo or send it as an archive via email.

Good luck!