AGENCY

Spring Boot Summer School 2019 Hackathon

HEAD OFFICE ADDRESS: AG04 Innovative Solutions | Srebrnjak 36 | Zagreb, Croatia
MAIL ADDRESS: AG04 Innovative Solutions | OIB 99895959831 | Antuna Bauera 29 | Zagreb, Croatia
info@ag04.com | +385 99 292 81 10

Spring Boot Summer School 2019 Hackathon

This is an assignment created by AGENCY04 for the Spring Boot Summer School 2019 Hackathon. The assignment is intended to make you use the knowledge gained during the 4-week long course.

Assignment

Applying for the Spring Boot Summer School was easy, right? You had to fill out two forms, then sit back, relax and wait for our response. Meanwhile, behind the scenes, we had a hard time manually narrowing down 582 to 36 applicants with whom we would kick off this years Spring Boot Summer School.

That was only the beginning, as we were sending out emails manually, marking any applicant status change in a spreadsheet, etc.

The application **you** will build is all about automating the process of applying to a summer school (not necessarily a Spring Boot one), rating candidates both automatically and manually, getting the list of candidates that will participate, and finally assigning them to teams.

Read the assignment carefully and approach it as you would a regular project in the future.

How to tackle the user stories is up to you, but because some of them are linked it would be smart to do them in order.

Grading

We do not require you to complete all user stories, but the more you manage to do, the more points you will get.

Half the points will be awarded by leveraging automated tests, so make sure you implement the API specification carefully. The other half will be awarded by manually reviewing the code and the usage of best practices.

Assumptions

- There is no security layer (for now), everybody can do anything, let's keep it simple.
- All request fields are required (unless indicated otherwise).
- If a required request field is missing, the HTTP Response Code `400` (Bad Request) should be returned.
- The standard HTTP Response Code is `200` (OK) (unless indicated otherwise).

Requirements

The requirements for the application are specified using user stories below. As previously mentioned, not all the stories are required to be completed, but the higher number of **correctly** implemented stories means more points.

User stories

SBSS-01: As an event organizer, I want to create events.

The purpose of this story is creating an event. The request contains a couple of fields, most of them are self-explanatory.

- The `name` field is unique, hence two events having the same name can't exist.
- The `teams` field is an array of objects, containing two fields, the `name` which is unique per event and the `mentors` array which is optional and contains the emails of the mentors for that team.
- The `registrationsNotBefore` field indicates the date and time at which the API starts accepting new registrations.
- The `registrationsNotAfter` field indicates the date and time after which the API stops accepting new registrations.
- The `confirmationNotAfter` field indicates the date and time after which the API stops accepting attendance confirmations.

The response body should be empty while the Location header should be set to the resource URI of the newly created event.

Request: `POST /event`

```
{
   "name": "Spring Boot Summer School 2019",
   "maxParticipants": 36,
   "teams": [
        {
           "name": "Amber"
        },
        {
           "name": "Loom"
        },
        {
           "name": "Metropolis"
        },
        {
           "name": "Panama"
        },
        {
           "name": "Skara"
        },
        {
           "name": "Valhalla",
           "mentors": [
              "nikola.masic@ag04.com",
              "mia.matic@ag04.com"
           ]
        }
   ],
   "registrationsNotBefore": "2019-07-01T00:00:00+01:00",
   "registrationsNotAfter": "2019-07-31T23:59:59+01:00",
   "confirmationNotAfter": "2019-08-10T23:59:59+01:00",
   "startDate": "2019-08-15",
   "weeks": 5
}
```

Response:

201 (Created)

```
Header: Location /event/<event_id>
```

400 (Bad Request)

When an event with the same name already exists or multiple teams having the same name were provided.

SBSS-02: As an event participant, I want to register for events.

The purpose of this story is for a participant to submit his registration for an event. In the case the event does not exist a `404` status code should be returned.

Request: `POST /event/<event_id>/registrations`

```
{
   "personal": {
       "name": {
           "first": "Ivan",
           "last": "Horvat"
       },
       "email":"ivan.horvat@ag04.com",
       "phone": "+385 99 292 8111",
       "education": {
           "faculty": "TVZ",
           "year": 3
       },
       "summary": "Interested in Games and Boardgames, a geek at heart. :)",
   },
   "experience": {
       "years": 2,
       "skills": [
           "API",
           "Spring",
           "Spring Boot",
           "Java",
           "Hibernate",
           "JPA"
       ],
       "repositoryUrl": "https://bitbucket.org/ihorvat",
       "summary": "I worked on technologies such as Spring Boot, Hibernate and JPA."
   },
   "motivation": "I want to expand my knowledge on Spring Boot.",
   "preferredOS": "macOS"
}
```

Response:

`201` *(Created)*

```
Header: Location /event/<event_id>/registrations/<registration_id>
```

`404` *(Not Found)*

When the event does not exist.

`405` *(Method Not Allowed)*

When the event is not accepting registrations.

It's important that registration ids are not easily guessable (e.g. sequential numbers) but rather something more obscure (e.g. UUID).

SBSS-03: As an event participant, I want to be able to withdraw my registration.

The purpose of this story is to enable a participant to withdraw his registration for an event. In the case the event or the registration do not exist a `404` status code should be returned.

Request: `DELETE /event/<event_id>/registrations/<registration_id>`

Response:

`200` *(OK)*

`404` *(Not Found)*

When the event or the registration do not exist.

SBSS-04: As an event organizer, I want registrations to be automatically scored.

The purpose of this story is scoring registrations automatically by the means of a simple scoring engine, the rules for scoring are as follows:

- For every **year of education 2 points** are awarded.
- For every **year of experience 5 points** are awarded.
- For every **skill** in the following list **20 points** are awarded:
    - Java
    - Spring
    - Spring Boot
- For every **skill** in the following list **10 points** are awarded:
    - Hibernate
    - JPA
    - Scala
- For every **other skill 5 points** are awarded.
- If a **repository url** (BitBucket, GitHub, GitLab, ...) is provided **10 points** are awarded

The final score should be stored for future use (where is up to you).

SBSS-05: As an event organizer, I want to manually score a registration.

The purpose of this story is to enable the event organizer to add or subtract points from the registration `score` field manually. The addition and subtraction is performed by prepending an addition or subtraction symbol (+ and - respectively) to a whole number (providing the symbol is required).

Request: `PUT /event/<event_id>/registrations/<registration_id>/score`

```
{
	"score": "+10",
	"comment": "Recommended by one of our employees."
}
```

Response:

`204` *(No Content)*

`400` *(Bad Request)*

When the `score` field is malformed.

`404` *(Not Found)*

When the event or the registration do not exist.

It’s not enough to just apply the operation to the score, all operations and comments should be stored and viewable later.

SBSS-06: As an event organizer, I want to view a single registration.

The purpose of this story is to enable the event organizer to view a single registration.

Request: `GET /event/<event_id>/registrations/<registration_id>`

Response:

`200` *(OK)*

```
{
            "personal": {
                ...
            },
            "experience": {
                ...
            },
            "score": 128,
            "comments": [
                {
                    "score": "+100",
                    "comment": "Highly recommended by our employee."
                }
            ]
 }
```

Most of the response fields have been omitted for clarity, they are defined as part of the SBSS-02 user story.

`404` *(Not Found)*

When the event or registration do not exist.

SBSS-07: As an event organizer, I want to view all registrations.

The purpose of this story is to enable the event organizer to view all registrations in a paginated fashion, ordered by the `score` field descending. Page number and size are controlled using the `page` and `size` query parameters.

Request: `GET /event/<event_id>/registrations?page=1&size=10`

Response:

`200` *(OK)*

```
{
   "content": [
       {
           "personal": {
               ...
           },
           "experience": {
               ...
           },
           "score": 128,
           "comments": [
               {
                   "score": "+100",
                   "comment": "Highly recommended by our employee."
               }
           ]
       },
       {
           "personal": {
               ...
           },
           "experience": {
               ...
           },
           "score": 111,
           "comments": []
           ...
       },
       ...
   ],
   ...
}
```

Most of the response fields have been omitted for clarity, they are defined as part of the SBSS-02 user story.

404 (Not Found)

When the event does not exist.

SBSS-08: As an event organizer, I want to send an invite to chosen participants.

The purpose of this story is to enable the event organizer to send an invite containing the registration token (id) for filling out additional details (kickoff attendance, T-shirt size, etc.) to the participants with the highest score. The number of participants which should receive the email is defined in the `maxParticipants` field of the event.

Request: `PUT /event/<event_id>/invite`

Response:

200 (OK)

```
{
   "participants": [
       {
           "email": "ivan.horvat@ag04.com",
           "registration": "435bb543-45a5-4dfb-8658-fd4e671fd74e"
       },
       {
           "email": "petra.peric@ag04.com",
           "registration": "1494dc8d-1dc5-4f45-87d2-ecce9871a2c7",
       },
       ...
   ]
}
```

404 (Not Found)

When the event does not exist.

405 (Method Not Allowed)

When the invites have already been sent.

SBSS-09: As an event participant, I want to confirm my participation.

The purpose of this story is to enable a participant to confirm his participation using the previously received registration token.

T-shirts come in many sizes and flavours, to differentiate the `t-shirt` field is a composite of F and M (standing for female and male respectively) and the size.

Request: `PATCH /event/<event_id>/registrations/<registration_token>`

```
{
   "participation": true,
   "kickoff": true,
   "t-shirt": "M XL",
   "gitlab": "ihorvat"
}
```

Response:

`204` *(No Content)*

`404` *(Not Found)*

When the event or the registration do not exist.

`405` *(Method Not Allowed)*

When the event is not accepting confirmations, the registrant has not been invited to participate or participation has already been confirmed.

SBSS-10: As an event organizer, I want to divide participants into teams.

The purpose of this story is to divide the confirmed participants into teams (defined when creating the event) by using the individual participant scores, summing them together and having the sum almost equal between teams. You can use the greedy partitioning algorithm to achieve the desired outcome.

Request: `PUT /event/<event_id>/teamUp`

Response:

`200` *(OK)*

```
{
   "teams": [
       {
           "name": "Valhalla",
           "members": [
               "ivan.horvat@ag04.com",
               "petra.peric@ag04.com",
               ...
           ]
       }
   ]
}
```

`404` *(Not Found)*

When the event does not exist.

`405` *(Method Not Allowed)*

When the participants have already been divided into teams.

SBSS-11: As an event organizer, I want to track the progress of participants week by week.

The purpose of this story is to provide a status and a comment for each participant week by week to track their progress. The `status` field is optional and it's value can be one of the following: *NOT_STARTED*, *IN_PROGRESS*, *PR_OPEN*, *COMPLETED*.

Request: `PUT /event/<event_id>/participants/<participant_id>/week/<week_no>`

```
{
   "status": "IN_PROGRESS",
   "comment": "Doing really well, potential candidate for employment."
}
```

Response:

`204` *(No Content)*

`404` *(Not Found)*

When the event or the participant do not exist.

`405` *(Method Not Allowed)*

When the week number is out of bounds.

SBSS-12: As an event organizer, I want to send registration and invitation emails.

This user story is about sending an actual email to the user when:

- The registration form is filled out (SBSS-02)
- The event invitations are sent (SBSS-08)
- The participation is confirmed (SBSS-09).

SMTP credentials will be provided upon request.

SBSS-13: As an event organizer, I want to have a richer API available.

So you arrived at the last (non-optional) user story, or are just having a quick read through? Either way the last user story is about exposing all the things you implemented through the API.

Request: `GET /event/<event_id>/participants?page=1&size=10`

Response:

200 (OK)

```
{
   "content": [
       {
           "personal": {
               ...
           },
           "experience": {
               ...
           },
           "score": 128,
           "comments": [
               ...
           ],
           "participation": true,
           "kickoff": true,
           "t-shirt": "M XL",
           "gitlab": "ihorvat",
           "progress": [
```

```
            {
                "week": 1,
                "flow": [
                    {
                        "status": "NOT_STARTED"
                    },
                    {
                        "status": "IN_PROGRESS",
                        "comment": "Started working on the task, still no MR."
                    },
                    {
                        "comment": "Still no MR, something is weird!"
                    },
                    {
                        "status": "PR_OPEN",
                        "comment": "MR open, so far so good."
                    },
                    {
                        "status": "COMPLETED",
                        "comment": "Merged, passed with flying colors."
                    }
                ]
            },
            ...
        ]
    },
    ...
],
...
}
```

Request: `GET /event/<event_id>/teams`

Response:

200 (OK)

```
{
   "teams": [
       {
           "name": "Valhalla",
           "mentors": [
               "nikola.masic@ag04.com",
               "mia.matic@ag04.com"
           ],
           "members": [
               {
```

```
                "personal": {
                    ...
                },
                "experience": {
                    ...
                },
                ...
            },
            ...
        ]
    }
  ]
}
```

Most of the response fields have been omitted for clarity, but by now if you came this far you know what to do. :)

SBSS-14 (optional): As an event organizer, I want automatic registration scoring to also score based on data retrieved from GitHub.

This user story is about using some useful data to improve the score of the registrants using GitHub, namely:

- number of repositories
- programming languages used in repositories
- activity

You can define the scoring yourselves, any other data which you might find useful is welcome.

There are Java libraries available to interface with the GitHub API, which one to use is up to you.

SBSS-15 (optional): As an event organizer, I want to send an invite to chosen participants automatically.

This story is about improving SBSS-08 to automatically send out invitations without having to call an API. The action should be scheduled after the date and time in the `confirmationNotAfter` field has passed.

SBSS-16 (optional): As an event organizer, I want to view everything visually.

In your favourite frontend technology, implement a simple UI on top of the API.