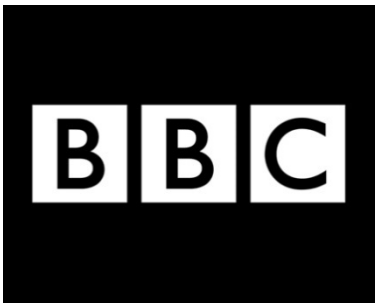

Software Engineer exercise

1. We're looking for people with a real passion for collaboratively creating great software. Please give an example of a software component you have designed and written from concept to deployment, outlining the steps you took. **(1000 character limit)**

For my final project at Makers Academy, I collaboratively built and deployed a hybrid iOS game in which players can view, attack and destroy monsters on a real-time map.

Steps taken:

- *Agreed on our MVP.*
- *Created wireframes, basic mockups and user stories.*
- *Decided on technologies and architecture spiking. During this process we had to execute a pivot as the project had been intended to be built in swift. We chose to write our front-end code in Javascript using Cordova; a decision informed by our need to quickly achieve MVP and ensure scalability. We also chose to build our own remote API server in Rails to handle user authentication and game logic.*
- *Diagrammed the flow of the application, specifically the JSON between the rails API and the Cordova app.*
- *We divided into two teams, with two pairs programming on the frontend and another on the backend at all times.*
- *We started to build incrementally via TDD. When necessary, we would pause the process for feature spiking.*
- *After reaching MVP (a Cordova application that could determine the location of the player and render them as a sprite on a map) we continued to iterate towards MVP2, and so on until we reached our minimum marketable product.*

2. Using the example that you provided above, tell us about a significant decision you made to solve a technical challenge. Give details of technologies that you chose and why you chose them. **(1000 character limit)**

The most significant technical challenges posed by this project were brought about by the decision to build a pure server-side API that would serve our client-side JavaScript

application. One particular challenge that this posed was the handling of user authentication and ensuring that our software was secure.

We decided to authenticate API requests using HTTP token based authentication. This would require the client to include an API key of some sort in the HTTP Authorization header of each request, as follows:

1. *User requests access with username/password*
2. *API validates credentials*
3. *API provides a signed token to the client*
4. *Client stores that token and sends along with every request*
5. *Server verifies token and responds with data*

This was implemented by adding the following methods to the User model to generate an API key on creation of a new user:

```ruby
class User < ActiveRecord::Base

  # Assign an API key on create
  before_create do |user|
    user.api_key = user.generate_api_key
  end

  # Generate a unique API key
  def generate_api_key
    loop do
      token = SecureRandom.base64.tr('+/=', 'Qrt')
      break token unless User.exists?(api_key: token)
    end
  end
end
```

On the controller side, we implemented the authentication using the built in Rails method ***authenticate_or_request_with_http_token:***

```ruby
  before_action :authenticate

  protected

  # Authenticate the user with token based authentication
  def authenticate
    authenticate_token || render_unauthorized
  end

  def authenticate_token
    authenticate_with_http_token do |token, options|
      @current_user = User.find_by(api_key: token)
    end
  end

  def render_unauthorized(realm = "Application")
    self.headers["WWW-Authenticate"] = %(Token realm="#{realm.gsub(/"/, "")}"
    render json: 'Bad credentials', status: :unauthorized
  end
end
```

A considerable benefit of using token based authentication was that we were not storing information about our user on a server or a session. The statelessness of the API renderered

the application more scalable and, ultimately, more secure: a token, not a cookie, is sent on every request, offering greater protection from CSRF attacks.

3. Using the example that you provided above, tell us about how you ensured your software was fit for purpose and of high quality. What did you learn and what would you do differently next time to do a better job? **(1000 character limit)**

---

*Throughout the project we adhered to **XP values** and **Agile methodologies**, using **test-driven development** and **refactoring** to help uncover the most effective design. We were aware that this incremental process encouraged an iterative approach that ensured quality and functionality. The use of **pair-programming**, regular refactoring, maintaining **a sustainable pace** and ensuring **full test coverage** at unit and feature level also provided quality assurance.*

*As it was a Cordova app outside of the browser using the built-in simulator. We also **deployed the app at MVP** so we could test functionality on a phone.*

*We utilised **Git version control** throughout the project, with each new feature being built on a separate branch and merged into the master following a code review. This ensured that new implementations did not interfere with existing code.*

This process taught me the value of 'best practice' and provided me with an appreciation for the way in which following XP and Agile practices can cultivate quality in your software.

If I were to do something different in the process, it would be to set out smaller iterative loops. We were initially overly ambitious with what could be achieved and so incorporated too many features into each of our iterations.