Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación
IIC2143 - Ingeniería de Software (I/2020)

# Proyecto Semestral

## 1. Objetivos

- Aplicar metodologías ágiles en el contexto de un equipo de desarrollo.
- Aprender el *framework Ruby on Rails* para desarrollar aplicaciones web.
- Aprender a usar distintas herramientas **por cuenta propia** y explorar distintas soluciones dependiendo de las exigencias del cliente o *Product Owner*.
- Conocer sobre buenas prácticas y herramientas de desarrollo de *software*.

## 2. Introducción

Luego de cuidadosos análisis, la empresa *DCCitas* descubrió que puede aprovechar la situación actual para construir una base de usuarios aprovechando el *boom* social que ocurrirá al finalizar la cuarentena. Decidieron crear una *aplicación web* para coordinar citas a semi-ciegas de tal manera que los usuarios hagan *match* y puedan juntarse después de la cuarentena (y después la sigan usando, logrando así una gran tracción). En este contexto, a tu equipo de trabajo le han solicitado realizar una nueva e innovadora aplicación web, donde los clientes puedan describir sus gustos y su localización y hacer *match* con gente cercana de gustos similares.

## 3. Características Generales

La aplicación debe permitir a los visitantes ver información sobre el sitio y su funcionamiento. Una vez que un cliente se registra e inicia sesión, éste podrá subir una foto e información personal sobre gustos y ubicación. Luego, deberá poder revisar una lista de sus potenciales *matches* y decidir si le interesa o no cada potencial *match*. Finalmente, una vez que se hace algún *match*, se debe poder elegir un local para tener la cita y una fecha. Por otra parte, la aplicación debe permitir la creación de nuevos locales de citas (restaurantes, cafés, etc...). Para poder crear un local de citas, se debe llenar un formulario, solicitando ser parte de la comunidad de locales de citas. Éstos deben ser revisados por un *administrador* de la aplicación, y puede ser aceptado o rechazado. El local de citas podrá editar toda su información.

### 3.1. Tipos de usuario

Tu aplicación debe manejar los siguientes tipos de usuario:

- **Visita (No registrado en la página)**: Puede acceder a la *landing page* de la aplicación para ver información general sobre el uso de la misma. Puede buscar y ver la información de los distintos locales de cita.

- **Cliente (Un usuario registrado en la página)**: Puede agregar una foto e información sobre gustos y localización, puede buscar y ver la información de los distintos locales de cita. Puede revisar potenciales *matches* y decidir si hacer o no hacer *match*. Puede decidir un local de citas y una fecha una vez que hace *match*.
- **Administrador general**: Tiene la capacidad de aceptar/rechazar solicitudes de locales de citas. Además puede eliminar usuarios/locales de citas según los criterios que deben ser definidos por el grupo de trabajo.

### 3.2. Comportamiento General

- La aplicación debe permitir el registro y la autenticación de usuarios.
- Un usuario debe poder ver los locales de citas.
- Un usuario debe poder ver la información de cada locale de citas, así como también el menú y las valoraciones y comentarios que otros usuarios han realizado.
- Un usuario registrado debe poder modificar los datos de su cuenta e incluso eliminarla.
- Un usuario registrado (cliente) debe poder revisar un listado de potenciales *matches*.
- Un usuario registrado (cliente) debe poder decidir si le interesa o no hacer *match* con un potencial *match*.
- Un usuario registrado (cliente) debe poder elegir la fecha y el local de citas en el que concretar la cita luego de hacer un *match*.
- Un usuario registrado (cliente) debe poder dejar un comentario y evaluar un local de citas donde haya concretado una cita.
- Un local de citas debe poder editar la información de su perfil.
- Un local de citas debe poder ver la cantidad de citas concretadas en sus dependencias.
- Un usuario registrado (administrador) debe poder moderar los comentarios en los locales de citas.
- Un usuario registrado (administrador) debe poder aceptar o cancelar el registro de un local de citas.
- Un usuario registrado (administrador) debe poder añadir, editar o eliminar comunas.
- Un usuario registrado (administrador) debe poder añadir, editar o eliminar gustos.

## 4. Atributos mínimos

### 4.1. Usuario

Debe manejar al menos los siguientes aspectos de un usuario:

- Nombre
- Descripción corta
- Edad
- Imagen de perfil
- Gustos
- Número de teléfono
- Comuna

### 4.2. Comuna

Debe manejar al menos los siguientes aspectos de una comuna:

- Nombre

### 4.3. Local de citas

Debe manejar al menos los siguientes aspectos de un local de citas:

- Nombre
- Descripción
- Valoración
- Comentarios
- Imagen/es (opcional)

### 4.4. Gusto

Debe manejar al menos los siguientes aspectos de un gusto:

- Nombre
- Descripción

### 4.5. Comentario

Se debe manejar al menos los siguientes aspectos de un comentario (para un local de citas):

- Local de citas al que corresponde
- Usuario
- Fecha y hora de creación
- Contenido

### 4.6. Inscripción de locales de citas

Las decisiones de modelación de un local de citas, en cuanto a cómo se administran, quedan a libre criterio de cada grupo, sin embargo se deben cumplir con los siguientes requerimientos mínimos:

- Contar con los atributos mínimos mencionados en la sección **4.3**.
- Debe ser administrado por un usuario debidamente registrado.
- Deben ser creados mediante una solicitud que luego es aprobada o rechazada por un administrador de la aplicación.
- El local de citas no puede ser visible sin previa aprobación.
- El administrador del local de citas debe poder editar la información de su local.

## 5. Información que se puede proporcionar

Puedes usar tu creatividad pero aquí se ofrecen algunas ideas:

1. Usuarios pueden filtrar los locales de citas según popularidad.
2. Visitas pueden ver cantidad total de *matches*/*matches* por día.
3. Mandar mensajes de texto con recordatorios para que vuelvas a usar la aplicación.

## 6. Funcionalidades mínimas

Tu aplicación debe abarcar las siguientes funcionalidades mínimas:

- CRUD[1] de usuarios.
- CRUD de comunas.
- CRUD de locales de citas.
- CRUD de gustos.
- CRUD de comentarios.
- *Sign up* de usuario.
- *Log in* de usuario.
- Actualización de información de cuenta de usuario.
- *Flow* de *matches* (ver potenciales *matches*, decidir si hay o no interés, una vez realizado el *match* concretar una fecha y un local de citas).
- Búsqueda de locales de citas.
- Votación y comentarios a locales de citas.
- Solicitud y respuesta de aplicación de locales de citas.

## 7. Requisitos mínimos de desarrollo

Para asegurar un producto de calidad, se les pide que utilicen las siguientes herramientas y buenas prácticas de desarrollo de *software*. Todas ellas son un estándar básico para la industria de software actual y potencian la producción de equipos de desarrollo.

### 7.1. *Kanban*: *Trello*

Utilizar el servicio de *Kanban Trello* para organizar su trabajo como equipo. Cada equipo debe tener un tablero de *Trello* que compartirá con su *Product Owner*. Éste puede tener la estructura (columnas) que el equipo encuentra conveniente mientras se note un claro flujo de trabajo que comunique el estado del proyecto a su *Product Owner*.

[1] *Create, Read, Update and Delete*

### 7.2. *Gitflow*

Para desarrollar la aplicación, gestionarán su proyecto en un repositorio *git*. Sobre esto, deben seguir el modelo de uso *Gitflow*. No es necesario seguirlo al pie de la letra, mientras se ocupen al menos dos *branches* principales y una *branch* por funcionalidad.

### 7.3. *Rubocop*

Seguir alguna guía de estilo de código para *Ruby* monitoreado por la gema *Rubocop*. Las configuraciones de estilo quedan a decisión de cada grupo, pero una vez fijadas **deben** respetarse.

### 7.4. *Heroku*

Utilizar la plataforma *Heroku* para publicar sus aplicaciones a producción.

## 8. Entregas, hitos y evaluación

El proyecto se llevará a cabo mediante desarrollo ágil inspirado en *Scrum*. Cada entrega se separa en un *Sprint* de dos semanas, donde el trabajo para cada *Sprint* es negociado con su *product owner* en reuniones de *Sprint Review*.

### 8.1. *Product Owner* y *Sprint Review*

Cada grupo de desarrollo tendrá asignado un *Product Owner* (ayudante) quien actúa como la contraparte del proyecto. Tras cada término de *Sprint* (entrega) se debe agendar una reunión (*Sprint Review*) con su *Product Owner* para discutir y monitorear el avance del proyecto. Además, deben definir junto a ella o él los pasos a seguir para el siguiente *Sprint*. **Todos los miembros del equipo deben asistir al *Sprint Review*** y debe planificarse para realizarse entre los **cuatro** días hábiles inmediatamente después del fin de un *Sprint*.

### 8.2. Coevaluación

Por cada entrega deberá responderse una coevaluación de sus compañeros. Ésta puede afectar positiva o negativamente su calificación.

### 8.3. Evaluación

Su ayudante asignado es el encargado de evaluar su avance, además de llevar el rol de *Product Owner*. Para cada *Sprint Review*, su ayudante hará una sesión de corrección que dependerá de la entrega. Ésta puede implicar evaluación grupal y/o evaluación individual de conocimientos. Las notas parciales de cada entrega son **individuales** y consideran el avance grupal e individual.

### 8.4. Entregas

En total, son 4 entregas parciales y una entrega final. Cada entrega se realiza mediante su repositorio asignado de grupo en la organización de *GitHub* del curso, donde se corregirá el último *commit* en la rama *master* dentro de plazo. Luego de la entrega 0, todas incluyen avance de funcionalidades. Cuáles de ellas deben incluir en cada entrega depende de sus negociaciones con su *product owner*. Si bien este documento sirve como guía base de proyecto, **el resultado final puede (y debe) variar**. Adicionalmente, algunas entregas incluyen un aspecto obligatorio o evaluación específica a realizar. A continuación, se listan a grandes rasgos los entregables:

#### 8.4.1. Entrega 0 (10 de Abril)

Relatos de usuario y aplicación mínima "Hello World!" con una configuración de *rubocop* publicada en *Heroku*.

#### 8.4.2. Entrega 1 (8 de Mayo)

Modelación mediante diagrama E/R de la aplicación y funcionalidades negociadas con el *Product Owner*.

#### 8.4.3. Entrega 2 (5 de Junio)

Funcionalidades negociadas con el *Product Owner*.

#### 8.4.4. Entrega 3 (26 de Junio)

Funcionalidades negociadas con el *Product Owner*.

#### 8.4.5. Entrega Final (3 de Julio)

Funcionalidades faltantes. Implementación completa de la aplicación web.

### 8.5. Presentación final (7 a 10 de Julio)

Finalmente, luego de las entregas parciales se realizará una presentación del producto logrado al equipo docente del curso. En esta oportunidad se busca que el equipo de desarrollo **completo** presente lo experimentado durante el desarrollo, el resultado obtenido y las lecciones aprendidas.

## 9. Recomendaciones Finales

Tienen bastante libertad para construir una aplicación de acuerdo a lo que cada grupo considere que es importante dado el espíritu y los objetivos que se explicaron al comienzo. Las funcionalidades (features) deben ser negociadas con el *product owner* (el ayudante que les seguirá durante todo el semestre). Se recomienda identificar y levantar la mayor cantidad posible de épicas y relatos de usuario en la *Entrega o Sprint 0* a pesar que no terminen siendo todos finalmente implementados.

## 10. Política de integridad académica

Los alumnos de la Escuela de Ingeniería de la Pontificia Universidad Católica de Chile deben mantener un comportamiento acorde a la Declaración de Principios de la Universidad. En particular, se espera que mantengan altos estándares de honestidad académica. Cualquier acto deshonesto o fraude académico está prohibido; los alumnos que incurran en este tipo de acciones se exponen a un Procedimiento Sumario. Es responsabilidad de cada alumno conocer y respetar el documento sobre Integridad Academica publicado por la Dirección de Docencia de la Escuela de Ingeniería en el SIDING. Específicamente, para los cursos del Departamento de Ciencia de la Computación, rige obligatoriamente la siguiente política de integridad académica. Todo trabajo presentado por un alumno para los efectos de la evaluación de un curso debe ser hecho individualmente por el alumno, sin apoyo en material de terceros. Por "trabajo" se entiende en general las interrogaciones escritas, las tareas de programación u otras, los trabajos de laboratorio, los proyectos, el examen, entre otros. Si un alumno copia un trabajo, obtendrá nota final 1.1 en el curso y se solicitará a la Dirección de Pregrado de la Escuela de Ingeniería que no le permita retirar el curso de la carga académica semestral. Por "copia" se entiende incluir en el trabajo presentado como propio partes hechas por otra persona. Obviamente, está permitido usar material disponible públicamente, por ejemplo, libros o contenidos tomados de Internet, siempre y cuando se incluya la referencia correspondiente. Lo anterior se

entiende como complemento al Reglamento del Alumno de la Pontificia Universidad Católica de Chile. Por ello, es posible pedir a la Universidad la aplicación de sanciones adicionales especificadas en dicho reglamento.