Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación
IIC2143 - Ingeniería de Software (II/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 y múltiples cálculos, la empresa *DCCarrete* descubrió que puede aprovechar la situación actual para construir una gran base de usuarios aprovechando el *boom* social que ocurrirá al finalizar la pandemia. Decidieron crear una *aplicación web* para coordinar *carretes* "abiertos y pagados" de tal manera que los usuarios puedan organizar fiestas que se lleven a cabo después de la pandemia (acumulando una base de usuarios que, con toda probabilidad, volverán a usar la aplicación, 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 usuarios puedan organizar *carretes* describiendo todos los servicios a contratar y también mostrar interés por asistir a otros *carretes* ya organizados.

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. Luego, deberá poder organizar un *carrete* **en su casa**, para el cual debe especificar la comuna en que se llevará a cabo y puede elegir distintos servicios que operen en dicha comuna (cada uno con su costo) y fijar una cantidad máxima de asistentes y una fecha de cierre de búsqueda de asistentes. Por otra parte, deberá poder revisar una lista de *carretes* en busca de asistentes. De interesarle un *carrete*, deberá poder mostrar su interés, explicitando cuánto dinero estaría dispuesto a pagar por asistir a dicho *carrete*. Al finalizar el período de búsqueda de asistentes de un *carrete*, los asistentes que declararon estar dispuestos a aportar más recibirán un código que podrán usar para asistir al *carrete*! Por último, la aplicación debe permitir la creación de nuevos servicios (bebestibles, iluminación, limpieza, etc...). Para poder crear un servicio, se debe llenar un formulario, solicitando ser parte de la comunidad de servicios. Éstos deben ser revisados por un *administrador* de la aplicación, y puede ser aceptado o rechazado. El servicio podrá editar toda su información.

3.1. Ejemplo de uso

Para ilustrar el funcionamiento de la aplicación, supongamos que tenemos un usuario Alicia que desea hacer un *carrete* y un usuario Bob que desea **carretear**. Alicia pondrá una descripción básica de su casa (en la cual será llevado a cabo el *carrete*), seleccionará los servicios que desea (por ejemplo, desea un servicio de limpieza y otro de bebestibles), fijará un máximo de 100 asistentes y dará un mes de plazo de búsqueda de asistentes. Una vez hecho esto, se publica una oportunidad de *carrete* explicitando el costo total del carrete, la cantidad máxima de asistentes y la cantidad de gente que ha mostrado interés en asistir. Es importante notar que **nadie puede ver cuánto está dispuesto a pagar otro usuario**, y que **solamente Alicia** debe poder ver **cuánto suma** el aporte de los 100 asistentes que más están dispuestos a pagar. Ahora, Bob desea asistir al *carrete* de Alicia. Ve que el costo total del *carrete* es de $100.000 y decide que está dispuesto a pagar $700. Una vez finalizado el período de búsqueda de asistentes, la aplicación determina que Bob si está dentro de los 100 asistentes (ya que algunos asistentes estaban dispuestos a pagar mucho, pero la gran mayoría estaba dispuesta a pagar menos que Bob), por lo que puede encontrar en su perfil el *carrete* seleccionado junto con un código que le dará a Alicia en la puerta de su casa el día del *carrete* (junto con $700). Si los 100 asistentes que más estaban dispuestos a aportar no suman lo necesario, el *carrete* no se lleva a cabo. Si suman más de lo necesario, Alicia se queda con todo lo que sobra.

3.2. 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 servicios.
- **Cliente (Un usuario registrado en la página)**: Puede agregar una foto e información personal, puede buscar y ver la información de los distintos servicios. Puede organizar un *carrete*. Puede mostrar interés por asistir a un *carrete*.
- **Administrador general**: Tiene la capacidad de aceptar/rechazar solicitudes de servicios. Además puede eliminar usuarios/servicios según los criterios que deben ser definidos por el grupo de trabajo.

3.3. Comportamiento General

- La aplicación debe permitir el registro y la autenticación de usuarios.
- Un usuario debe poder ver los servicios.
- Un usuario debe poder ver la información de cada servicio, así como también las valoraciones y comentarios que otros usuarios han realizado.
- Un usuario registrado (cliente) debe poder modificar los datos de su cuenta e incluso eliminarla.
- Un usuario registrado (cliente) debe poder revisar un listado de oportunidades de *carretes*.
- Un usuario registrado (cliente) debe poder mostrar interés por asistir a un *carrete*.
- Un usuario registrado (cliente) debe recibir un código único para un *carrete* para el que quedó *seleccionado*.
- Un usuario registrado (cliente) debe poder dejar un comentario y evaluar un servicio que haya utilizado en un *carrete* al que asistió o que organizó.
- Un servicio debe poder editar la información de su perfil (como el máximo de personas para las que puede funcionar, el costo por el servicio, etc).

- Un servicio debe poder ver la cantidad de *carretes* en los que ha participado.
- Un usuario registrado (administrador) debe poder moderar los comentarios en los servicios.
- Un usuario registrado (administrador) debe poder aceptar o cancelar el registro de un servicio.
- Un usuario registrado (administrador) debe poder añadir, editar o eliminar categorías de servicios.
- Un usuario registrado (administrador) debe poder añadir, editar o eliminar comunas.

4. Atributos mínimos

4.1. Usuario

Debe manejar al menos los siguientes aspectos de un usuario:

- Nombre
- Edad
- Imagen de perfil
- Número de teléfono

4.2. Comuna

Debe manejar al menos los siguientes aspectos de una comuna:

- Nombre

4.3. Servicio

Debe manejar al menos los siguientes aspectos de un servicio:

- Nombre
- Descripción
- Capacidad máxima
- Precio
- Comunas en las que opera
- Valoración
- Comentarios
- Imagen/es (opcional)

4.4. Carrete

Debe manejar al menos los siguientes aspectos de un *carrete*:

- Título
- Descripción
- Servicios
- Comuna en la que se lleva a cabo
- Dirección específica
- Capacidad máxima de asistentes
- Costo total
- Estado de la búsqueda de asistentes
- Imagen/es (opcional)

4.5. Comentario

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

- Servicio al que corresponde
- Usuario
- Fecha y hora de creación
- Contenido

4.6. Inscripción de servicios

Las decisiones de modelación de un servicio, 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 servicio no puede ser visible sin previa aprobación.
- El administrador del servicio debe poder editar la información de su servicio.

5. Información que se puede proporcionar

Puedes usar tu creatividad pero aquí se ofrecen algunas ideas:

1. Usuarios pueden filtrar los servicios según precio total/cercanía/cantidad de asistentes.
2. Visitas pueden ver cantidad total de *carretes* totales/*carretes* por día.
3. Mandar mensajes de texto con recordatorios para que revises las nuevas oportunidades de *carretes* en tu comuna.

6. Funcionalidades mínimas

Tu aplicación debe abarcar las siguientes funcionalidades mínimas:

- CRUD[1] de usuarios.
- CRUD de comunas.
- CRUD de servicios.
- CRUD de carretes.
- CRUD de comentarios.
- *Sign up* de usuario.
- *Log in* de usuario.
- Actualización de información de cuenta de usuario.
- *Flow* de *carretes* (organizar un *carrete*, mostrar interés en un *carrete*, una vez terminada la búsqueda de asistentes mostrar de alguna manera a los seleccionados un código para poder asistir al *carrete*.
- Búsqueda de servicios.
- Votación y comentarios a servicios.
- Solicitud y respuesta de aplicación de servicios.

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*.

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.

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

7.4. *RSpec*

Escribir tests para la aplicación utilizando la plataforma de testing en *Ruby* llamada *RSpec*. Una vez escritos los tests, éstos **deben** pasar. La aplicación no puede tener tests fallando.

7.5. *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 (11 de Septiembre)

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

8.4.2. Entrega 1 (9 de Octubre)

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

8.4.3. Entrega 2 (1 de Noviembre)

Funcionalidades negociadas con el *Product Owner*.

8.4.4. Entrega 3 (22 de Noviembre)

Funcionalidades negociadas con el *Product Owner*.

8.4.5. Entrega Final (11 de Diciembre)

Funcionalidades faltantes. Implementación completa de la aplicación web.

8.5. Presentación final o Demo (11 de Diciembre)

Finalmente, luego de las entregas parciales se realizará una presentación (o demo) 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.

8.6. Nota

Como se especifica en el programa del curso, la nota de proyecto se divide en tres componentes:

- $\overline{E}_P$: Promedio de notas de entregas parciales.
- E_F: Nota de entrega final, como producto desarrollado.
- P_F: Nota de presentación final.

La nota de proyecto (P) se calcula como sigue:

$$P = 0.5 \cdot \overline{E}_P + 0.2 \cdot E_F + 0.3 \cdot P_F$$

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 el *Sprint 0* a pesar de que no terminen siendo todos finalmente implementados. Pueden encontrar las fechas actualizadas de las entregas del proyecto en el *calendario oficial del curso*.

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.