Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2143 - Ingeniería de Software 2022-2

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

Muchas veces nos vemos en la situación de querer realizar algún panorama para pasar un buen tiempo, desde luego, el aburrimiento es algo natural en todos y las ganas de evitarlo nunca faltan. Sin embargo, te das cuenta que muchas veces, dependes de la disponibilidad de algo, ya sea porque es un producto arrendado, es un lugar con capacidad limitada o incluso una persona que presta servicios, es por esto que tu Product Owner le ha solicitado a tu equipo desarrollar una aplicación en torno a un servicio de su necesidad que permita reservar cosas de acuerdo a fechas y/o disponibilidades preestablecidas, sera tu misión descubrir cuales serán sus necesidades.

3. Características Generales

La aplicación debe permitir a los visitantes ver información sobre el sitio y su funcionamiento, esta deberá tener un menú donde se muestre claramente qué es lo que está ofreciendo. Una vez que un visitante se registra e inicia sesión, este podrá subir una foto e información personal. Luego podrá acceder de manera detallada a la información del servicio que ofrece la página como fechas, capacidad restante, precio, etc. Si lo estima conveniente, podrá enviar una solicitud de reserva la cual puede ser aceptada o rechazada por los moderadores, en caso de ser aceptada, se habilitará una opción donde el usuario puede realizar consultas o conversar con otras personas según corresponda. Una vez concluida, se puede dejar una reseña indicando el nivel de satisfacción del servicio ofrecido por la página entregando a futuros clientes, mejor información.

3.1. Tipos de usuario

Su 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 básica de lo que ofrece la aplicación.

- **Cliente (Un usuario registrado en la página):**
  - Puede agregar una foto e información personal.
  - Puede buscar y ver la información de distintos productos/servicios disponibles.
  - Puede mandar solicitudes de reserva además de poder cancelarlas.
  - Puede mandar mensajes.
  - Puede dejar reseñas.

- **Moderador (Un usuario especial de la pagina)**
  - Puede hacer todo lo que un cliente normal puede hacer.
  - Puede aceptar o rechazar solicitudes.
  - Puede modificar los recursos de la aplicación.

- **Administrador general:**
  - Puede eliminar usuarios/publicaciones/reseñas 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 qué es lo que se ofrece para poder reservar en la aplicación.
- Un usuario registrado debe poder modificar los datos de su cuenta e incluso eliminar esta.
- Un usuario registrado debe poder solicitar reservas.
- Un usuario moderador debe poder aceptar o rechazar solicitudes que envían los otros usuarios.
- Un usuario registrado debe ser capaz de cancelar una reserva.
- Un usuario registrado debe ser capaz de comunicarse (con un chat, por ejemplo), con otros usuarios con los que comparta una reserva si es que esta involucra varios usuarios simultáneamente.
- Un usuario registrado debe poder ver un listado de reservas solicitadas y su estado (aceptada / rechazada / pendiente), así como un historias de las mismas.
- Un usuario registrado debe poder dejar reseñas de el servicio que ha recibido.
- El administrador debe ser capaz de moderar los recursos de la aplicación (usuarios, reservas, reseñas, etc), llegando a eliminarlos si es necesario.

4. Atributos mínimos

4.1. Usuario

Debe manejar al menos los siguientes aspectos de un usuario:

- Nombre
- Imagen de perfil

- Número de teléfono
- Correo
- Contraseña

4.2. Recurso

- Este puede ser modelado de distintas maneras, todo dependerá de la temática de proyecto que les indicara su *Product Owner*.

4.3. Reserva

Debe manejar al menos los siguientes aspectos de una reserva:

- Recurso solicitado.
- Descripción
- Día inicio reserva
- Día fin reserva

4.4. Chat o mensajería

- Este puede ser modelado de distintas maneras, sin embargo, se pide que se puedan encontrar mensajes enviados por todos los usuarios que forman parte de una misma reserva.

4.5. Reseña

Debe manejar al menos los siguientes aspectos de una reseña:

- Calificación
- Contenido

5. Funcionalidades mínimas

Su aplicación debe abarcar las siguientes funcionalidades mínimas:

- CRUD[1] de usuarios.
- CRUD de recursos.
- CRUD de solicitudes.
- CRUD de mensajes.
- CRUD de reseñas.
- *Sign up* de usuario.
- *Log in* de usuario.
- Actualización de información de cuenta de usuario.
- Aceptar o rechazar solicitudes.

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

6. *Nice to have*

Pueden incluir otras funcionalidades cómo:

- Sistema de reclamos al administrador.
- Sistemas de filtros.
- Integración con servicios externos a la aplicación.

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. *GitHub Projects*

Utilizar el servicio de *GitHub Projects*. Cada equipo debera configurar su tableroque vera el Ayudante. 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*. Se les subirá una configuración básica opcional a seguir. La configuración de estilo queda a decisión de cada grupo, pero una vez fijadas deben respetarse.

7.4. *RSpec*

Escribir tests para la aplicación utilizando la plataforma de testing en *Ruby* llamada *RSpec*. Una vez escritos los tests, estos **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* distinto, 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 **tres** inmediatamente siguientes días hábiles 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. Detalles de ésta se especificarán luego de la primera entrega.

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, individual y la coevaluación respondida.

8.4. Sprints

En total, son 4 Sprints. Cada sprint se realiza mediante su repositorio asignado de grupo en la organización de *GitHub* del curso, donde se corregirá el último *commit* en la rama *main* dentro de plazo. Luego del sprint 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, algunos sprints incluyen un aspecto obligatorio o evaluación específica a realizar. A continuación, se listan a grandes rasgos los sprints:

8.4.1. Sprint 0 (9 de Septiembre)

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

8.4.2. Sprint 1 (30 de Septiembre)

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

8.4.3. Sprint 2 (21 de Octubre)

Funcionalidades negociadas con el *Product Owner*.

8.4.4. Sprint 3 (11 de Noviembre)

Funcionalidades negociadas con el *Product Owner*.

8.5. Presentación y Revisión Final

Finalmente, luego del ultimo sprint, se dará un tiempo de aproximadamente **1 semana** para arreglar pequeños errores de la aplicación, esta corresponderá a la Revisión Final. Además, en esta etapa se realizara 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.

8.6. Nota

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

- $\overline{E}_P$: Promedio de notas de sprints parciales.
- R_F: Nota de revisión final, como producto desarrollado.
- P_F: Nota de presentación final.

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

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

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 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 Académica 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.