Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2143 - Ingeniería de Software 2023-2

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

Hoy en día existen muchas tiendas y plataformas destinadas a la adquisición de distintos tipos de productos, incluso, gracias a esto, han surgido diversas formas de conseguir nuevas cosas, una de ellas, es mediante intercambios. El propósito de los intercambios es poder obtener nuevos productos a cambio de uno que tengamos en nuestra posesión, que consideremos que tiene un valor similar. De esta manera, se fomenta la economía circular y se reduce la producción de residuos, promoviendo un enfoque más sostenible para el consumo. Además, los intercambios pueden fortalecer las comunidades al crear lazos sociales y promover el espíritu de colaboración entre sus miembros, lo que contribuye no solo a la adquisición de bienes, sino también al bienestar general de la sociedad.

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, también podrá acceder a la información de los productos que distintos usuarios ofrecen para intercambiar, junto con comentarios o algo que se espera a cambio. Una vez que un visitante se registra e inicia sesión, este podrá subir una foto e información personal. Si lo estima conveniente, podrá enviar solicitudes de trueque por uno o varios de estos de manera simultánea, las cuales pueden ser aceptadas o rechazadas por los dueños de los productos, en caso de ser aceptadas, 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 evaluando tanto al servicio como a los usuarios involucrados, a modo de entregar más información a futuros clientes.

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 disponibles.
  - Puede mandar solicitudes de intercambio además de poder cancelarlas.
  - Puede mandar mensajes.
  - Puede dejar reseñas.
- **Administrador general:**
  - Puede eliminar usuarios/publicaciones/reseñas según los criterios que deben ser definidos por el grupo de trabajo.

3.2. Comportamiento General

- Permitir a los usuarios crear una cuenta y posteriormente ingresar a ella.
- Permitir que los usuarios actualicen su información personal y/o eliminen su cuenta.
- Un usuario debe poder ver qué es lo que se ofrece para poder intercambiar en la aplicación.
- Permitir a los usuarios escoger entre los productos ofrecidos y generar una solicitud de intercambio.
- Permitir a los usuarios dejar reseñas de los productos adquiridos y/o del proceso de intercambio.
- Permitir a los usuarios mandar mensajes (a través de un chat) para consultar sobre el proceso de intercambio.
- Un usuario registrado debe ser capaz de cancelar una reserva.
- Un usuario registrado debe poder ver un listado de solicitudes y su estado (aceptada / rechazada / pendiente), así como un historial de las mismas.
- 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 recomendados

4.1. Usuario

Debe manejar al menos los siguientes aspectos de un usuario:

- Nombre
- Descripción

- Imagen de perfil
- Número de teléfono
- Correo
- Contraseña

4.2. Producto

- Este puede ser modelado de distintas maneras, todo dependerá de la temática de proyecto conversada con su *Product Owner*.

4.3. Solicitud

Debe manejar al menos los siguientes aspectos de una solicitud:

- Productos solicitados.
- Descripción
- Fecha de reserva
- Estado

4.4. Chat o mensajería

- Este puede ser modelado de distintas maneras, sin embargo, se pide que se pueda realizar entre varios usuarios a la vez.

4.5. Reseña

Debe manejar al menos los siguientes aspectos de una reseña:

- Calificación
- Contenido

5. *Nice to have*

Pueden incluir otras funcionalidades cómo:

- Sistema de reclamos al administrador.
- Sistemas de filtros.
- Integración con servicios externos a la aplicación (API's).
- Sección de lista de deseos.
- Sistema de Mailer.

6. 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.

6.1. *GitHub Projects*

Utilizar el servicio de *GitHub Projects*. Cada equipo deberá configurar su tablero que verá el Ayudante. Este 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*.

6.2. *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.

6.3. *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.

6.4. *Minitest*

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

6.5. *Render*

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

6.6. *Yard*

Utilizar la herramienta *Yard* para documentar el código escrito.

7. 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*.

7.1. *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*.

7.2. Coevaluación

Por cada entrega deberá responderse una coevaluación de sus compañeros. Esta puede llegar a afectar de forma negativa su calificación en caso de que se estime necesario.

7.3. 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. Esta 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.

7.4. 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:

7.4.1. Sprint 0 (8 de septiembre)

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

7.4.2. Sprint 1 (29 de septiembre)

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

7.4.3. Sprint 2 (28 de octubre)

Funcionalidades negociadas con el *Product Owner* con el objetivo de conseguir un *MVP*[1].

7.4.4. Sprint 3 (17 de noviembre)

Funcionalidades negociadas con el *Product Owner*.

7.5. Presentación y Revisión Final

Finalmente, luego del último sprint, se dará un tiempo de aproximadamente **5 días** para arreglar pequeños errores de la aplicación, esta corresponderá a la Revisión Final. Además, en esta etapa 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.

7.6. Nota

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

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

[1] *Minimum Viable Product*

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

$$P = 0.6 \cdot \overline{E}_P + 0.2 \cdot R_F + 0.2 \cdot P_F$$
$$NP = 0.9 \cdot P + 0.1 \cdot T_R$$

8. 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 de que no terminen siendo todos finalmente implementados.

9. Política de integridad académica

Los/as estudiantes 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/as estudiantes que incurran en este tipo de acciones se exponen a un Procedimiento Sumario. Es responsabilidad de cada estudiante conocer y respetar el documento sobre Integridad Académica publicado por la Dirección de Docencia de la Escuela de Ingeniería.

Específicamente, para los cursos del Departamento de Ciencia de la Computación, rige obligatoriamente la siguiente política de integridad académica. Todo trabajo presentado por un/a estudiante para los efectos de la evaluación de un curso debe ser hecho individualmente por el/la estudiante, 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.

En particular, si un/a estudiante copia un trabajo, o si a un/a estudiante se le prueba que compró o intentó comprar 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. En caso de que corresponda a "copia" a otros estudiantes, la sanción anterior se aplicará a todos los involucrados. En todos los casos, se informará a la Dirección de Pregrado de la Escuela de Ingeniería para que tome sanciones adicionales si lo estima conveniente.

También se entiende por copia extraer contenido sin modificarlo sustancialmente desde fuentes digitales como Wikipedia o mediante el uso de asistentes inteligentes como ChatGPT o Copilot. Se entiende que una modificación sustancial involucra el análisis crítico de la información extraída y en consecuencia todas las modificaciones y mejoras que de este análisis se desprendan. Cualquiera sea el caso, el uso de fuentes bibliográficas, digitales o asistentes debe declararse de forma explícita, y debe indicarse cómo el/la estudiante mejoró la información extraída para cumplir con los objetivos de la actividad evaluativa.

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 Estudiante de la Pontificia Universidad Católica de Chile (https://registrosacademicos.uc.cl/reglamentos/estudiantiles/). Por ello, es posible pedir a la Universidad la aplicación de sanciones adicionales especificadas en dicho reglamento.

Compromiso del Código de Honor

Este curso suscribe el Código de Honor establecido por la Universidad, el que es vinculante. Todo trabajo evaluado en este curso debe ser propio. En caso de que exista colaboración permitida con otros/as estudiantes, el trabajo deberá referenciar y atribuir correctamente dicha contribución a quien corresponda. Como estudiante es un deber conocer el Código de Honor (https://www.uc.cl/codigo-de-honor/)