Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2143 - Ingeniería de Software 2025-2

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, las plataformas de reservas se han convertido en herramientas fundamentales para organizar y asegurar el acceso a distintos productos y servicios. Ya sea para arrendar un espacio, reservar un producto limitado, agendar una actividad recreativa o coordinar la prestación de un servicio, estos sistemas permiten a los usuarios gestionar su tiempo y recursos de manera eficiente. El objetivo principal de una página de reservas es facilitar la planificación, garantizando disponibilidad y evitando imprevistos. De esta forma, los usuarios pueden elegir con antelación la fecha, el lugar o el producto que desean, mientras que los organizadores logran una mejor administración de su capacidad y demanda. En este caso, crearemos un sistema de reservas que busque simplificar este proceso, entregando una experiencia clara, rápida y confiable, donde los usuarios puedan asegurar lo que necesitan y los administradores mantengan un control ordenado de la oferta disponible.

3. Características Generales

La aplicación debe permitir a los visitantes ver información general sobre el sitio y su funcionamiento, esta deberá tener un menú donde se muestren claramente las categorías de reservas disponibles (por ejemplo, productos, servicios o espacios), y cada visitante podrá acceder a los detalles de estas opciones, junto con las reseñas de otros usuarios que ya las han utilizado. Sin embargo, los visitantes no podrán realizar reservas sin antes registrarse. Una vez que un usuario se registra e inicia sesión, tendrá la posibilidad de subir una foto e información personal para completar su perfil. Si lo estima conveniente, podrá enviar solicitudes de reserva para uno o varios servicios de manera simultánea, las cuales pueden ser aceptadas o rechazadas según la disponibilidad definida por los administradores. En caso de ser aceptadas, se habilitará una sección donde el usuario podrá gestionar sus reservas, comunicarse con los encargados, recibir información adicional relacionado con la experiencia reservada. Una vez concluida la actividad o servicio reservado, se puede dejar una reseña evaluando tanto el servicio como al proveedor, con el fin de entregar más información y confianza a futuros usuarios de la plataforma.

4. 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.

4.1. *Plataforma Zoe*

Utilizar la plataforma del curso *Zoe*. Cada equipo deberá agregar en las distintas secciones la información correspondiente y que se note un claro flujo de trabajo que comunique el estado del proyecto a su *Product Owner*.

4.2. *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.

4.3. *Rubocop*

Seguir alguna guía de estilo de código para *Ruby* monitoreado por la gema *Rubocop*. Se les subió una configuración *básica opcional* a seguir. La configuración de estilo queda a decisión de cada grupo, pero una vez fijadas deben respetarse.

4.4. *Minitest + Simplecov*

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. Además se debe cumplir un mínimo de cobertura, para asegurar un correcto testeo.

4.5. *Render*

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

5. 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*.

5.1. *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*.

5.2. 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.

5.3. 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.

5.4. 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:

5.4.1. Sprint 0 (4 de septiembre)

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

5.4.2. Sprint 1 (2 de octubre)

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

5.4.3. Sprint 2 (23 de octubre)

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

5.4.4. Sprint 3 (13 de noviembre)

Funcionalidades negociadas con el *Product Owner*.

5.5. Presentación y Revisión Final

Finalmente, luego del último sprint, se dará el fin de semana para corregir pequeños errores de la aplicación, esta instancia corresponderá a la **Revisión Final**. Posteriormente, durante las dos semanas siguientes se realizarán las presentaciones finales, donde cada grupo realizará una presentación del producto logrado al equipo docente del curso. Las fechas exactas serán informadas con anticipación y se llevará a cabo en horario de clases o ayudantía. En esta presentación se espera que participe el equipo de desarrollo **completo**, exponiendo tanto la experiencia vivida durante el proceso de desarrollo, como los resultados obtenidos, y las principales lecciones aprendidas.

[1] *Minimum Viable Product*

5.6. Nota

Como se especifica en el programa del curso, la nota de proyecto se divide en las siguientes 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 (NP) se calcula como sigue:

$$NP = 0.6 \cdot \overline{E}_P + 0.2 \cdot R_F + 0.2 \cdot P_F$$

6. 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.

7. 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/)