Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de Computación
IIC3745 – Testing (II/2020)

Proyecto semestral

1. Objetivos

- Aplicar metodologías ágiles en el contexto de un equipo de desarrollo.
- Profundizar conocimientos del *framework Ruby on Rails* para desarrollar aplicaciones web aplicando técnicas de *testing*.
- Aprender de forma autónoma la utilización de herramientas y definir una solución de *software* en base a las exigencias de un *product owner*.
- Profundizar conocimientos sobre herramientas y buenas prácticas de desarrollo de *software*.
- Aplicar conocimientos adquiridos durante el curso para asegurar calidad de un *software*.

2. Introducción

El proyecto del curso busca que a través de grupos de 5 personas experimenten el desarrollo de una aplicación web aplicando metodologías, prácticas y herramientas que mejoren la calidad del resultado. Con este fin se les solicita que desarrollen el tema definido para el proyecto considerando algunos requisitos mínimos de desarrollo y exigiendo distintos niveles de pruebas sobre el *software*.

3. Requisitos mínimos de desarrollo

Para asegurar un producto de calidad se les pide que utilicen algunas herramientas y buenas prácticas mínimas de desarrollo de *software*. Todas ellas son un estándar básico para la industria de *software* actual y potencian la eficiencia de equipos de desarrollo. Es posible que durante el semestre se agreguen más herramientas a esta lista, pero esto será notificado oportunamente.

3.1. *RoR, Postgres, RSpec*

Se exigirá como mínimo *RoR* $\geq$ 6.0.3, *PostgreSQL* $\geq$ 12.4 y la utilización de *RSpec* junto con Shoulda Matchers para codificar *tests* sobre la aplicación.

3.2. *SimpleCov*

Con la ayuda de la herramienta *SimpleCov* garantizar un 100 % de cobertura del código (sin incluir vistas ni archivos de configuración) al ejecutar sus baterías de *tests*.

3.3. *Git* y *Gitflow*

Para versionar el código de su aplicación utilizarán un repositorio de *git*. Se les exigirá "buenas prácticas" sobre los *commits*.

Además, para coordinar el desarrollo sobre este deben seguir la metodología de *branching Gitflow*. No es necesario seguirla al pie de la letra, pero como mínimo deben hacer la diferencia entre *master*, *develop* y las *branches* de funcionalidades.

3.4. *Rubocop*, *eslint*, *scss-lint*, y *erb-lint*

Definir y seguir guías de estilo de código para *Ruby*, *JS*, *CSS/SCSS* y *ERB/HTML*, controladas por las librerías *Rubocop*, *ESLint*, *scsslint*, *ERBLint* u otras. Las configuraciones de estilo quedan a decisión de cada grupo, pero estas deben ser *razonables* y aprobadas por su *product owner*.

3.5. *Docker*

Deben configurar sus ambientes de desarrollo con *Docker*. No es necesario que utilicen esta herramienta si no lo desean, pero deben asegurarse que los correctores puedan probar sus aplicaciones y correr los *tests* a través de ella.

3.6. *Bundler y yarn*

Gestionar las dependencias de su aplicacióon a través de *Bundler* y *yarn*.

3.7. *Bundler-audit*

Bundler-audit es una herramienta de análisis de código estático que verifica vulnerabilidades presentes en las dependencias del proyecto.

3.8. *Brakeman*

Brakeman es una herramienta que a través de análisis de código estático verifica vulnerabilidades de seguridad en una aplicación.

3.9. *Heroku*

Utilizar la plataforma *Heroku* para publicar sus aplicaciones a producción. **Las funcionalidades se corregirán exclusivamente a través de esta plataforma.**

3.10. *GitHub Actions*

Automatizar la ejecución de análisis estáticos y *tests* con la ayuda de *GitHub Actions*. También deben configurar *deploys* automáticos a Heroku para cada *commit* que pase las pruebas en la *branch master*.

3.11. *README.md*

Sus repositorios deben contener archivos *README.md* atingentes al desarrollo que contengan toda la información necesaria para corregir las entregas (como por ejemplo funcionalidades desarrolladas/por desarrollar, link a Heroku, pasos necesarios para ejecutar la aplicación y sus *tests*).

4. 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* debe ser acordado con su *product owner* al inicio de cada *sprint*, lo cual será responsabilidad de cada grupo.

Se publicarán tareas mínimas a realizar para los distintos *sprints*, pero deberán acordar el detalle de estas con su *product owner*. Por lo mismo se tendrá en consideración al evaluar cada entrega el alcance propuesto en comparación a lo efectivamente logrado.

4.1. Coevaluación

Para cada entrega se deberá responder una coevaluación sobre el trabajo de sus compañeros, la cual podría afectar a la nota obtenida. Detalles de ésta se especificarán luego de la primera entrega.

4.2. Evaluación

Su ayudante asignado es el encargado de evaluar su avance, además de llevar el rol de *product owner*. De ser necesario, su ayudante podrá pedirles una sesión de corrección que dependerá de cada entrega. Ésta puede implicar evaluación grupal y/o evaluación individual de conocimientos.

4.3. Entregas

En total habrá 4 entregas parciales. Cada entrega se realizará mediante un repositorio asignado a un grupo en la organización de *GitHub* del curso, donde se corregirá el último *commit* en la rama *master* dentro del plazo estipulado. Además, al momento de la entrega la última versión de la aplicación de un grupo debe estar disponible a través de Heroku.

Todas las entregas incluyen avances en las funcionalidades de la aplicación. Cuáles de ellas se deben incluir en cada entrega depende de los acuerdos con su *product owner*. Si bien este documento sirve como guía base del 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:

4.3.1. Entrega 1 (16 de septiembre)

En esta entrega deben definir los requisitos de su aplicación y configurar el ambiente de desarrollo. En otras palabras, se pide lo siguiente:

- Definición de requisitos en forma de relatos de usuarios
- Planificación de funcionalidades a desarrollar para cada *sprint*
- Modelo de datos de la aplicación
- Mock-ups de las principales vistas
- Configuración del ambiente de desarrollo considerando los requisitos mínimos
- CRUD de usuarios y lógica de sesiones/roles
- *Tests* de modelos, controladores, vistas y rutas

4.3.2. Entrega 2 (14 de octubre)

- Correcciones comentarios entrega 1
- Desarrollo de más funcionalidades y tests
- Actualización de planificación

4.3.3. Entrega 3 (4 de noviembre)

- Correcciones comentarios entrega 2
- Desarrollo de más funcionalidades y tests
- Actualización de planificación
- Agregar *tests* de aceptación con Capybara

4.3.4. Entrega 4 (25 de noviembre)

- Correcciones comentarios entrega 3
- Desarrollo de más funcionalidades
- Actualización de planificación
- Agregar *tests* de usabilidad con usuarios

4.4. Presentación final (11 de diciembre)

Luego de las entregas parciales deberán entregar un video sobre una presentación del desarrollo logrado durante el curso. En esta oportunidad se busca que el equipo presente lo experimentado durante el desarrollo, el resultado obtenido y las lecciones aprendidas.

En particular, la presentación se deberá enfocar más en la gestión y aseguramiento de calidad que en mostrar las funcionalidades realizadas (tests, herramientas, metologías, entre otros aspectos). Además, se solicitarán pruebas de sistema sobre la aplicación desarrollada.

4.5. Criterio de corrección

Cada entrega se evaluará cualitativamente según lo solicitado en base a los siguientes criterios:

Nota	Explicación
7.0	Destacado
6.0	Más allá de lo esperado
5.5	Muy bueno
5.0	Bueno
4.5	Correcto
4.0	Cumple con lo esperado
3.0	Insuficiente
2.0	Muy insuficiente

4.6. 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 entregas parciales.
- P_F: Nota de presentación final.

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

$$P = 0,175 * E_1 + 0,175 * E_2 + 0,175 * E_3 + 0,175 * E_4 + 0{,}3 * PF$$

5. 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.