SMART LIVING: CAFETERÍA VIRTUAL PARA LA INSTITUCIÓN UNIVERSITARIA ANTONIO JOSÉ CAMACHO

Manuel Alejandro Pastrana Pardo, Ana Milena Rojas, Nicolás Ortiz y Nicolás Bermúdez

Semillero de Investigación ITmedia

Grupo de Investigación GrinTIC

Institución Universitaria Antonio José Camacho

Recibido: 03/03/2020. Aprobado: 21/04/2020

Cómo citar este artículo:

Pastrana Pardo, M.A., Rojas, A.M., Ortiz, N. y Bermúdez, N. (2020). Smart living: cafetería virtual para la Institución Universitaria Antonio José Camacho. Revista Sapientía, 12 (23), 47-59.

RESUMEN

El macroproyecto ecosistema Smart Campus, que se lleva a cabo en la UNIAJC, busca impactar a la comunidad en general a partir de la optimización de los servicios institucionales. Estas mejoras están enmarcadas dentro de proyectos que, en casos como el del presente artículo, están apoyados por trabajos de grado y en otros directamente por el equipo propio de becarios de la Dirección de Tecnologías de la Información y la Comunicación de la universidad. Todas las soluciones generadas a partir de Smart Campus están orientadas en 6 ejes: Smart People, Smart Goverment, Smart Enviroment, Smart Mobility, Smart Economy y Smart Living. Cada uno de estos ejes tiene unos objetivos estratégicos alineados con el plan de desarrollo institucional, por lo que son de alta relevancia para el desarrollo de la institución.

El presente documento refleja los resultados más relevantes del trabajo de Bermúdez & Ortiz (2020), quienes, alineados al eje de Smart Living, generan una solución denominada cafetería virtual, que permite realizar compras, reserva de almuerzos, visualización de todos los productos ofrecidos y la calificación del servicio ofrecido por la cafetería, siendo su mayor ventaja ser multiplataforma.

PALABRAS CLAVE

Smart campus, smart living, desarrollo web, arquitectura de software.

ABSTRACT

The Smart Campus ecosystem macroproject, carried out at UNIAJC, seeks to impact the community in general through the optimization of institutional services. These improvements are framed within projects that, in cases like the one in this article, are supported by undergraduate projects and in others directly by the team of fellows of the Directorate of Information and Communication Technologies of the university. All the solutions generated from Smart Campus are oriented in 6 axes: Smart People, Smart Goverment, Smart Enviroment, Smart Mobility, Smart Economy and Smart Living. Each of these

axes has strategic objectives aligned with the institutional development plan, making them highly relevant to the institution's development.

This document reflects the most relevant results of Bermúdez & Ortiz (2020), who, aligned with the axis of Smart Living, generate a solution called virtual cafeteria, which allows purchases, reservation of lunches, display of all the products offered and the rating of the service offered by the cafeteria, being its greatest advantage being cross-platform.

KEYWORDS

Smart Campus, smart living, web development, software architecture.

INTRODUCCIÓN

El proyecto Smart Campus inscrito en el Decanato Asociado de Investigaciones de la Institución Universitaria Antonio José Camacho, liderado por la ingeniera Ana Milena Rojas del grupo de investigación GrinTIC con colaboración del semillero ITmedia, está desarrollando diversas soluciones orientadas al mejoramiento de la institución. Esta labor incluye la colaboración activa de estudiantes en la búsqueda del potenciamiento de su formación académica y su perfil profesional, incrementando sus habilidades en diversas áreas de la ingeniería de sistemas.

Smart campus cuenta con unas subcategorías específicas o ejes donde desarrollar los proyectos relacionados con el mejoramiento institucional, estos son:

- Smart Living: Desarrolla servicios que fomenten la integración social, la inclusión, el bienestar, la cultura y la empleabilidad
- Smart Goverment: Busca la participación de toda la comunidad y la transparencia de la política y la toma de decisiones.
- Smart Enviroment: Centra sus esfuerzos en el uso de las TI para optimizar, gestionar y racionalizar correctamente los recursos naturales.
- Smart People: Se enfoca en las habilidades de trabajo y teletrabajo mediante las TI; en disponer acceso a la educación, a la enseñanza y aprendizaje continuo tanto a estudiantes como a personal de la propia universidad; a promover una sociedad inclusiva que mejore la creatividad, la innovación y la participación en la vida pública.
- Smart Economy: Se enfoca en la educación como recurso base de la economía de la universidad, fomentando la competitividad e impulsando el emprendimiento personal, profesional y la innovación.
- Smart Mobility: Tiene como objetivo gestionar de manera sostenible, segura y eficiente la accesibilidad y los transportes de la universidad.

Alienando esfuerzos con el proyecto mencionado, específicamente dentro del eje Smart Living, se concibe este trabajo de grado llamado Sistema Cafetería Virtual UNIAJC, que mediante un aplicativo permite ofrecer el servicio de las cafeterías de una forma más accesible, todo esto apalancado en un modelo de Universidad Inteligente, soportada en servicios y recursos digitales, en la búsqueda de impactar un proceso misional como lo es el proceso de bienestar universitario y, por ende, a toda la comunidad académica.

Las cafeterías de la universidad ofrecen sus productos para el consumo de la comunidad universitaria y en su día a día se ven afectados por los diferentes horarios que tienen los estudiantes, docentes y directivos en la hora de almuerzo, haciendo que sus productos no tengan una demanda establecida. Este proceso afecta a la comunidad, porque en algunas ocasiones no pueden obtener el servicio deseado. Por lo anterior, surge la posibilidad de realizar el presente trabajo de grado con el fin de generar una alternativa para esta problemática, mediante una App que permita a estudiantes, profesores y administrativos

de la institución universitaria realizar pedidos. Esto optimizará tiempos de espera y facilitará la compra sin necesidad de hacer filas en las cafeterías.

Para la creación de este aplicativo se utiliza la metodología de desarrollo de software de Smart Campus, compuesta por los frameworks Scrum y XP, esto en pro de mantener como eje transversal de todo el proceso la calidad preventiva. La construcción del aplicativo será bajo el lenguaje de programación IONIC 3, que permite un desarrollo híbrido para las plataformas IOS y ANDROID, logrando abarcar así a toda la comunidad.

MARCO TEÓRICO

En el proceso de desarrollo, las etapas por las que un proyecto debe pasar han sido conceptualizadas por diversos autores, como se resume a continuación:

- Análisis: etapa donde se recopila la información necesaria para funcionalmente determinar qué debe hacer el sistema, delimitar el alcance del mismo y delimitar las restricciones a las que se enfrenta el proyecto, como indica Sommerville (2011).

- Diseño: etapa donde el equipo de trabajo determina cómo construir el sistema basado en los atributos de calidad del sistema Sommerville (2011). Aquí toma relevancia que cada atributo se correlaciona con un patrón de diseño, que no es más que una solución que ya existe, está probada y funciona idealmente para solucionar temas particulares, como muestra el trabajo de Petre (2013). En su mayoría estos patrones son provistos por frameworks de desarrollo.

- Desarrollo y calidad: las metodologías de desarrollo de software como RUP y MSF, comúnmente denominadas tradicionales, mantienen las etapas de construcción y pruebas separadas, generando un enfoque de calidad correctivo. Adicionalmente, bajo este mismo enfoque la ejecución es secuencial, siendo el desarrollo lo que se ejecuta primero. Contrario a lo anterior, los modelos ágiles plantean que estos dos pasos deben de ser ejecutados al tiempo bajo un marco preventivo que potencie los resultados finales, sin tener que esperar mucho para obtener la retroalimentación y mejora de las funcionalidades desarrolladas.

- Despliegue: etapa donde el equipo de trabajo pone en ambiente final o de producción el aplicativo para ser utilizado por todos los usuarios que se ha estimado interactuarán con él, como indica Pressman (2015).

Para efectos de este artículo, los autores centran la atención en la etapa de diseño, donde se sustentan los aspectos guías de la arquitectura a través de los atributos de calidad y los patrones de diseño que soportan la solución. En este sentido es importante referenciar el modelo de 4 vistas más 1, propuesto por Kruchten (2006), en el que plantea una revisión de los componentes que integran la solución desde diversas perspectivas, como lo ilustra la figura 1.

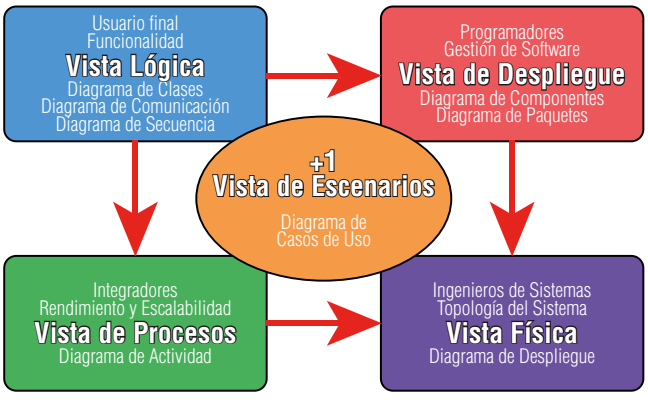

Figura 1. Modelo 4 vistas + 1 Modelo Fuente: Moya (2012)

Cada vista de las anteriormente referenciadas en la figura 1, presenta las decisiones de diseño reflejadas a través de diagramas UML específicos que denotan la implementación de patrones de diseño y satisfacen los atributos de calidad propuestos. A continuación, se detalla un poco más cada una de estas vistas:

- **La vista lógica:** basada en la definición de Kruchten (2006) y reforzada en el trabajo de Pressman (2015), esta vista comprende las abstracciones fundamentales del sistema a partir del dominio de problemas. En esta vista se representa la funcionalidad que el sistema proporcionará a los usuarios finales. Es decir, se ha de representar lo que el sistema debe hacer, y las funciones y servicios que ofrece. Para completar la documentación de esta vista se pueden incluir los diagramas de clases, de comunicación o de secuencia de UML.

- **La vista de proceso:** como propone Kruchten (2006), esta vista comprende el conjunto de procesos de ejecución independiente a partir de las abstracciones anteriores y su comunicación entre sí. Pressman (2015) indica que esta vista se representa desde la perspectiva de un integrador de sistemas y, de manera específica, describe el flujo de trabajo paso a paso, los componentes de negocio y operacionales que conforman el sistema. Para completar la documentación de esta vista se puede incluir el diagrama de actividad de UML.

- **La vista física:** según Kruchten (2006), esta vista se muestra desde la perspectiva de un ingeniero de sistemas, exhibiendo todos los componentes físicos del sistema, así como las conexiones físicas entre esos componentes que conforman la solución (incluyendo los servicios). Para completar la documentación de esta vista se puede incluir el diagrama de distribución de UML.

- **La vista de desarrollo:** para definir esta vista, Kruchten (2006) invita a ver el sistema desde la perspectiva de un programador, es decir, el sistema a partir de los componentes que lo integran, reflejando dentro de los diagramas UML los componentes necesarios que permiten la composición de capas de software y dan paso al flujo de mensajes modelando comportamientos predeterminados que garanticen óptimos resultados operativos. Para completar la documentación de esta vista, Pressman (2015) sugiere que se pueden incluir diagramas de componentes y/o de paquetes.

- **Vista de Escenarios (+1):** Kruchten (2006) indica que esta vista es representada por casos de uso o por historias de usuario, y tiene la función de unir y relacionar las otras 4 vistas. En este orden de ideas, Pressman (2015) sugiere que en esta vista se pueden incluir el diagrama de casos de uso de UML, diagramas de escenario y diagramas de estado.

- En el caso del framework ágil Scrum, esta vista no se usa debido a que es representada de una manera más flexible y precisa, con el uso de las historias de usuario.

Como se mencionó anteriormente, el eje central para el uso de estas vistas son los atributos de calidad y su correlación con los patrones de diseño. La tabla 1 menciona los principales:

Tabla 1. Resumen atributos de calidad y patrones de diseño de la solución. Fuente: Bermúdez & Ortiz (2020)

Atributo de Calidad	Descripción	Tácticas /Patrón de Arquitectura
Confidencialidad	Según (Barbacci & Kazman, 1997), se define como la ausencia de acceso no autorizado a la información.	Patrón: *Rol Base Acces* Es un enfoque de restricción del acceso al sistema a los usuarios autorizados mediante roles.
		Patrón: Autorizador-Autenticador, Mediante este patrón con un grado de complejidad garantiza de que no cualquier personal no autorizado pueda ingresar a nuestra aplicación
		Patrón: *Trusted Subsystem,* Mediante esta técnica se prevé protección contra el acceso a datos e información no autorizado, ya sea accidental o deliberadamente.
Portabilidad	El trabajo de Patidar & Suman (2015) la define como la habilidad del sistema para ser ejecutado en diferentes ambientes de computación. Estos ambientes pueden ser hardware, software o una combinación de los dos.	Patrón: *multi-Layers* Consiste en estructurar aplicaciones que pueden ser descompuestas en grupos de subáreas, las cuales se clasifican de acuerdo con un nivel particular de abstracción.

Continúa en la página 52

Viene de la página 51

Usabilidad	Nielsen (2010), definen este atributo de calidad como la facilidad con que las personas pueden utilizar una herramienta particular o cualquier otro objeto fabricado por humanos con el fin de alcanzar un objetivo concreto.	Patrón: *Aprendibilidad* Mediante este patrón se prevé que el usuario entienda el mecanismo del sistema y el objetivo de que cumpla con todos los pasos de este. Patrón: *Eficiencia* Con este patrón se garantiza que las respuestas del controlador se muestren de manera rápida y efectiva, mediante uso de llamadas AJAX. Patrón: Protección contra errores de usuario Mediante este patrón se evita que el usuario cometa errores. Patrón: *Estética de la interfaz de usuario.* Mediante este patrón se trata de que la interfaz satisfaga la interacción con el usuario.
Mantenibilidad	Patidar & Suman (2015) lo definen como la capacidad de someter a un sistema a reparaciones y evolución.	**Patrón: Model-View Controler** Divide una aplicación interactiva en tres componentes. El modelo (model) contiene la información central y los datos. Las vistas (view) despliegan información al usuario. Los controladores (controllers) capturan la entrada del usuario. Las vistas y los controladores constituyen la interfaz del usuario.
Modificabilidad	Se define como la capacidad de modificar el sistema de manera rápida y a bajo costo (Folmer, Gurp, & Bosch, 2005).	

METODOLOGÍA

El proceso de desarrollo de software usado para la elaboración del producto se centra en el marco de trabajo ágil Scrum y las etapas tradicionales descritas en detalle a continuación:

ANÁLISIS

En esta fase se elicitan los requisitos o necesidades funcionales y las restricciones del sistema a través del product backlog, que contiene la recopilación de la información mediante historias de usuarios, detallando en estas los criterios de aceptación para cumplir con la funcionalidad del sistema.

Debido a que las historias de usuario no poseen un estándar dentro del mundo ágil, a diferencia de la norma IEEE 830 en su última revisión y que aplica para los casos de uso utilizados en metodologías como RUP, deben ser elaboradas con un modelo usado y reconocido en la industria. Por lo anterior, este proyecto se basa en la estructura propuesta por Pastrana, Ordoñez, Ordoñez, Thom & Merchan (2018) para su construcción: Yo como [*usuario*] requiero [*una funcionalidad*] para [*un motivo*], bajo los criterios de aceptación [*definidos*].

Una vez obtenida la lista de funcionalidades a través de historias de usuario, se realizan una serie de validaciones con el objetivo de constatar la conformidad por parte del cliente con lo recopilado en el artefacto mencionado, y aclarar dudas respecto a funcionalidades o requerimientos antes de comenzar el diseño y la construcción. Lo anterior no implica que esta fase termine aquí, dado que en cada planning meeting nuevamente se revisa el detalle actual de las historias de usuario y estas son susceptibles al refinamiento constante por estar escritas en lenguaje natural y no en un lenguaje de única interpretación como Business Proces Models como sugieren Pastrana, Ordóñez, Ordonez, Thom, & Merchan (2018).

PLANEACIÓN

Determina el alcance total del esfuerzo requerido para elaborar el proyecto. Según PMI (2017), el orden de avance de la planificación inicia con el registro de las partes interesadas y el acta de constitución de proyecto; con estos resultados se procede a:

- Elaborar el plan de gestión de los interesados, el cual se va a representar en una matriz de los interesados.
- Elaborar la matriz de la comunicación de los interesados y el equipo.
- Elaborar el presupuesto.
- Elaborar el plan de gestión de riesgos, el cual se representa en la matriz de riesgos.

Durante la planeación Scrum juega un papel muy importante. Previo al inicio del proyecto el product backlog es estimado y priorizado para una mejor comprensión del trabajo total a realizar.

DISEÑO

Tomando como referencia los requisitos del cliente, se procede a modelar el software bajo el estándar UML dentro de un modelo 4+1 de Kruchten (2006).

Así mismo, se construyeron los prototipos de pantalla con la herramienta mockflow; el diseño gráfico utilizado para dar una sensación visual agradable al usuario es realizado con Adobe Ilustrator.

Es importante resaltar que la vista física representa el uso de modelo en la nube tipo IaaS (Infraestructura como Servicio). Los servicios de Amazon Web Services permiten utilizar contenedores de sistemas operativos y de base de datos que simplifican el modelo de mantenimiento y administración de estos, por lo que se adopta

el uso de un contenedor EC2 con el sistema operativo de Linux.

CONSTRUCCIÓN

Basado en la etapa de análisis y la solución obtenida en la etapa de diseño, se procede a la codificación del proyecto, donde resalta el estilo arquitectural Modelo Vista Controlador. A su vez se debe tener en cuenta las pruebas funcionales y los estándares de codificación para garantizar el buen funcionamiento de la aplicación y el cumplimiento de los objetivos del proyecto.

Las herramientas por utilizar para esta etapa son el editor Brackets. Desde la capa de lógica y acceso a datos, el lenguaje PHP en su versión 7.0 y para la capa de presentación se utiliza el framework para aplicaciones de internet enriquecida (Rich Internet Applications o RIA) Bootstrap, la librería JQuery. Por último, como servidor de aplicaciones se utiliza Apache 2.4.

PRUEBAS E INTEGRACIÓN

Las pruebas de software son la investigación empírica y técnica realizada para facilitar a los interesados información sobre la calidad del producto IEEE Computer Society (2014).

Dentro del proyecto, con el objetivo de obtener una retroalimentación constante del avance del mismo manteniendo la más alta calidad posible, se hace la implementación de pruebas automatizadas a partir del uso de PhpUnit y la herramienta de integración continua Jenkins. Estas pruebas se centran en comprobar la calidad de la implementación verificando si lo que se ha realizado afecta el desplegable de manera negativa o positiva. En caso de ser negativo notifica de ello al equipo de trabajo. Adicionalmente, las pruebas unitarias permiten validar el código, verifican distintas variables, como la revisión de los parámetros en los métodos, el tipo de dato que se devuelve, el estado final de los métodos, entre otros.

Las pruebas unitarias son un filtro de calidad que tiene el equipo de desarrollo para garantizar el cumplimiento de la historia de usuario construida por cada integrante del equipo. Este tipo de prueba centra su objetivo en poder identificar que las operaciones funcionales se ejecuten correctamente y que, en caso de presentarse un error, el sistema pueda reaccionar de manera adecuada controlando la excepción e impidiendo el colapso del sistema. Esto fomenta el aprendizaje continuo y el cambio sobre la cultura de desarrollo del equipo, organizando la refactorización del código implementado y mejorándolo continuamente.

Todo esto sucede previo a las pruebas de aceptación con el usuario final y en paralelo al desarrollo del proyecto.

DESPLIEGUE

El despliegue se realizó considerando una concurrencia aproximada de 3000 usuarios simultáneos en todo el país. Para esto se realizaron constantemente pruebas no funcionales de carga, estrés y volumen que permitieran prever posibles fallos antes de su puesta en marcha definitiva y garantizar así su correcta operación en el ambiente de producción.

RESULTADOS

Para la etapa inicial del proyecto se definió un documento de product backlog aprobado por todos los interesados, que dio paso a la especificación funcional del sistema. Durante la etapa de diseño posterior a la selección de atributos de calidad y patrones de diseño, se elaboró el documento de arquitectura de software, en el que destacan a nivel de UML los resultados de la toma de decisiones. A continuación, se expone la vista de despliegue que sirve de guía principal para el equipo a la hora de interpretar el modelo a seguir en la construcción del proyecto.

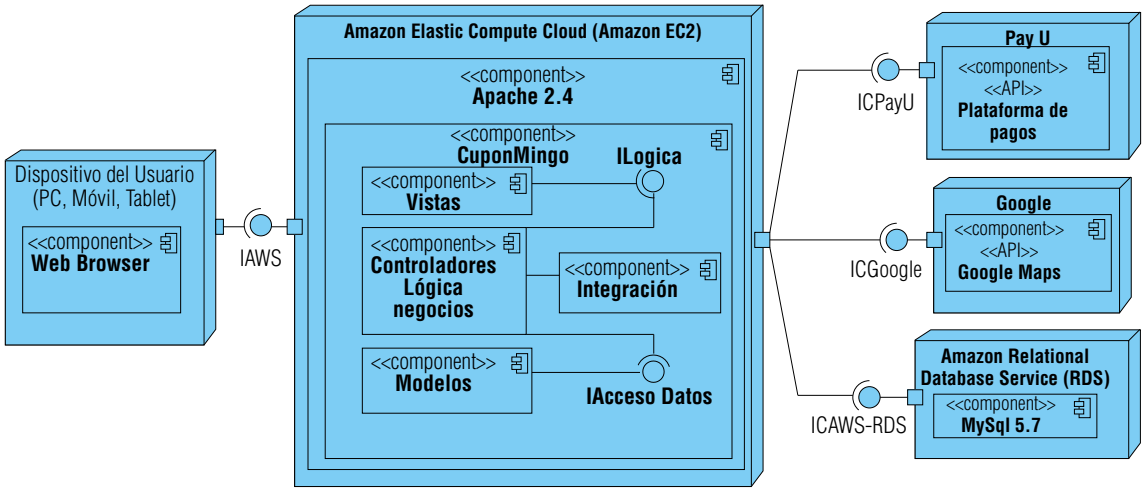

Figura 2. Vista de despliegue. Fuente: Bermúdez & Ortiz (2020)

En la figura 2 se aprecia que el estilo arquitectural principal está guiado por Modelo Vista Controlador. Este modelo genera una interacción del usuario final con la vista que, mediante peticiones HTTP/POST, se comunica con los controladores pertenecientes a una capa de lógica de negocios completamente separada, tanto de la capa anterior denominada presentación y de la siguiente capa de acceso a datos, que se encarga de la manipulación de los modelos y las peticiones a la base de datos. La tabla 2 refleja la selección de tecnologías para la elaboración del proyecto:

Tabla 2. Selección de tecnologías para el proyecto. Fuente: Bermúdez & Ortiz (2020)

Tecnologías	**Herramientas**
BD	Mysql 5.7
Framework test	PhpUnit
Maquetación	Html5- css3
Lenguaje front-end	Javascript (jquery)
Lenguaje back-end	Php 7.0
Framework front-end	Boostrap 3.7
Servidor de aplicaciones	Apache 2.4
Servidor web	Amazon Elastic Compute Cloud (Amazon EC2)
Sevidor de bases de datos	Amazon Relational Database Service (RDS)
Sistema de gestión de bd	Workbench 6.3
Integración continua	Jenkins 2.46
Servidor de repositorios	Bitbucket
Cliente git	SourceTree 2.4.7
Editor de código	Brackets 1.10

Durante el proceso de calidad, las pruebas unitarias mediante el uso del framework PhpUnit juegan un papel crucial en el resultado final del producto de software.

Además del uso de PhpUnit, se hace uso de la herramienta de integración continua Jenkins y del versionador de código Bitbucket. Gracias a este ecosistema de calidad preventiva, a raíz de cada actualización del código fuente en el repositorio, todo el proceso se inicia y se mide constantemente el avance o retroceso de los resultados, permitiendo la mejora continua como una constante durante todo el proceso. En la figura 3 se refleja el comportamiento de estas herramientas:

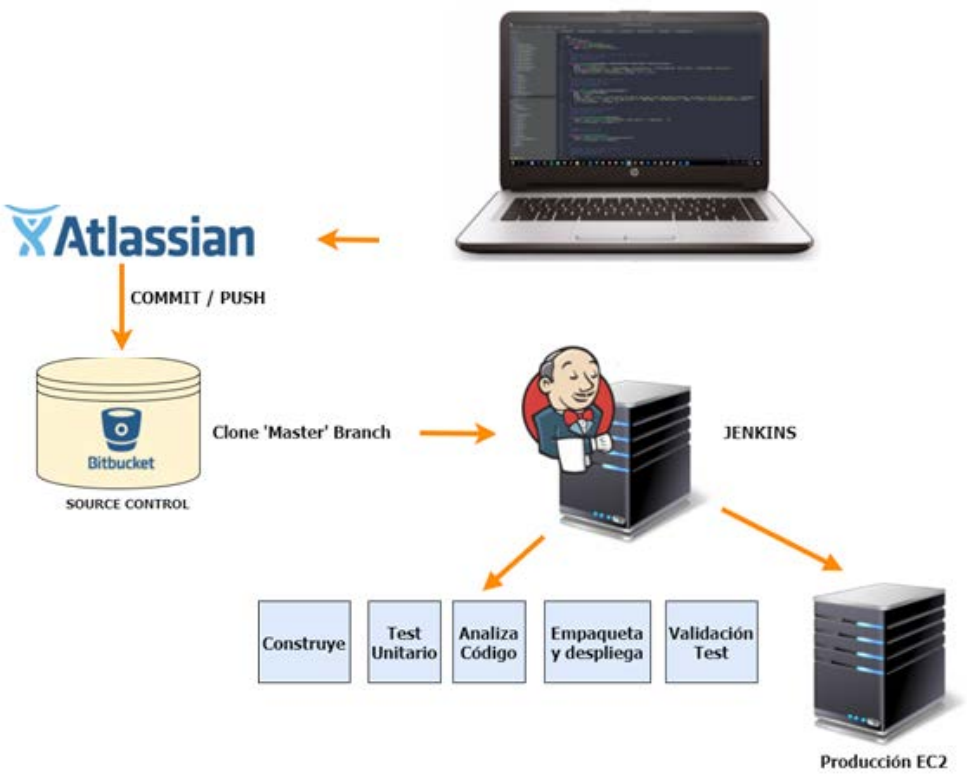

Figura 3. Integración y Pruebas Unitarias. Fuente: Bermúdez & Ortiz (2020)

A continuación, se incluye un ejemplo práctico sobre la clase OperacionesDB.php para mayor comprensión del lector sobre las pruebas unitarias:

Test sobre un método 'create', que se encarga de insertar información en la base de datos:

```
public function testCreate(){
  $info = array("nombre_departamento"=>"BOYACÁ","total_personas"=>2422319);
  $this->assertNotNull($this->instanceOperacionesDb->create('cp_loc_departamento',$info));
  $this->assertGreaterThan(0, $this->instanceOperacionesDb->create('cp_loc_departamento',$info));
}
```

Figura 4. Prueba Unitaria. Fuente: Bermúdez & Ortiz (2020)

La figura 4 analiza el método 'create', en el cual se realizan las respectivas aserciones. Adicionalmente, se valida que el registro no retorne ningún valor nulo, y que el resultado del registro debe retornar un valor mayor a 0.

Visto desde Jenkins, la figura 5 muestra el resultado de la ejecución de la prueba unitaria.

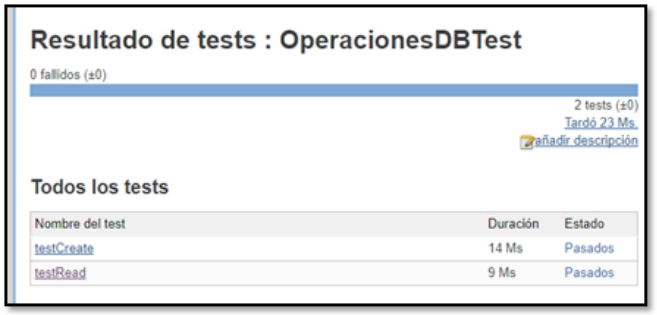

Figura 5. Resultado de la Prueba unitaria en Jenkins. Fuente: Bermúdez & Ortiz (2020)

Por parte de BitBucket también se realiza una revisión de la calidad de los resultados a través del avance del proyecto. En la figura 6 se aprecia el comportamiento de las pruebas unitarias del proyecto, indicando la cantidad de errores en color rojo y las pruebas exitosas en azul.

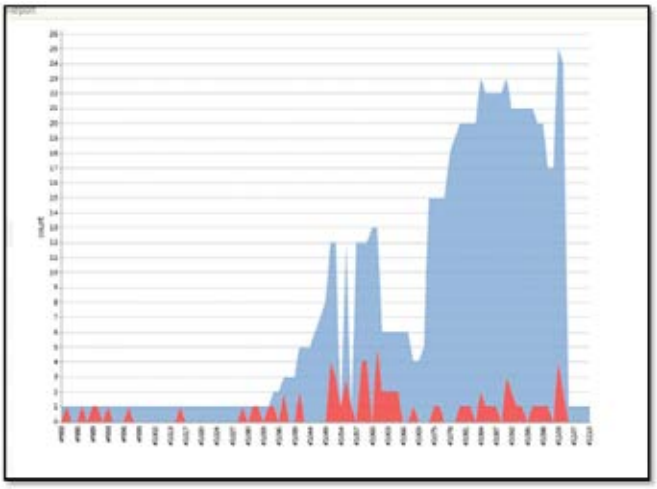

Figura 6. Gráfico de total de Pruebas Unitarias. Fuente: Bermúdez & Ortiz (2020)

Finalmente, el proyecto fue elaborado y entregado a satisfacción de todos los interesados y actualmente está puesto en producción.

CONCLUSIONES

El realizar el desarrollo de software dentro de un marco ágil como scrum permite, gracias a la retroalimentación constante, que el equipo de desarrollo y todos los interesados conozcan el avance diario y los posibles impedimentos para lograr los objetivos del sprint. Esto hace que todos sean capaces de reaccionar a tiempo a cualquier impedimento y darle una solución temprana. Adicionalmente, la integración del marco de trabajo scrum con los elementos de calidad preventiva planteados dentro del proyecto, como las pruebas unitarias y la integración continua hacen que el desarrollo tenga el mínimo de reproceso posible a la hora de liberar el producto. Herramientas como el versionador de código son fundamentales en un proceso de desarrollo para mantener el código fuente con trazabilidad de cambios, estandarizado y con capacidad de recuperación a fallos, incrementando la confiabilidad, escalabilidad y mantenibilidad del aplicativo a futuro.

Es importante resaltar que la identificación de atributos de calidad y patrones de diseño detallada en el Documento de Arquitectura de Software DAS, facilitó la selección de tecnologías y de frameworks de desarrollo asociados al lenguaje. Lo anterior, hace que la etapa de desarrollo de software sea más eficiente, dado que el equipo de desarrollo no tendrá que preocuparse por pensar que hacer y cómo hacerlo, sino que solamente se enfocará en entender cómo implementar los patrones seleccionados para ese framework de desarrollo sin tener que explorar todo el framework a totalidad.

Gracias a que el resultado del proyecto es una aplicación multiplataforma, se garantiza que podrá ejecutarse en cualquier dispositivo móvil.

Esto permitirá una mayor cobertura y el ofrecimiento de nuevos servicios de Bienestar Universitario para toda la comunidad académica. Al mismo tiempo, las cafeterías de la universidad podrán innovar al trabajar con esta nueva plataforma mejorando la comunicación con sus clientes.

Como trabajos futuros se recomienda ampliar el detalle del producto sobre la tabla nutricional, incluir consejos de alimentación sana y generar un resumen nutricional para que el usuario pueda identificar que tan sano está comiendo.

REFERENCIAS BIBLIOGRÁFICAS

Bermúdez, N., & Ortiz, N. (2020). *Sistema Cafetería Virtual UNIAJC* (tesis de pregrado). Institución Universitaria Antonio José Camacho, Cali, Colombia.

Folmer, E., Gurp, J. Van, & Bosch, J. (2005). Software Architecture Analysis of Usability. *Architecture*, 38–58. Retrieved from http://www.springerlink.com/index/87jl9wkck2gwyyd4.pdf

IEEE Computer Society. (2014). IEEE Standard for Software Quality Assurance Processes. *IEEE Std 730-2014 (Revision of IEEE Std 730-2002)*. https://doi.org/10.1109/IEEESTD.2014.6835311

Kruchten, P. (2006). Planos Arquitectónicos: El Modelo de "4 + 1" Vistas de la Arquitectura del Software. *IEEE Software,* 12(6), 1–16.

Moya, (2012), Ilustración Modelo 4 vistas +1. Fuente https://jarroba.com/modelo-41-vistas-de-kruchten-para-dummies/)

Nielsen, J. (2010). What Is Usability? In User Experience Re-Mastered. https://doi.org/10.1016/b978-0-12-375114-0.00004-9

Pastrana, M., Ordóñez, H., Ordonez, A., Thom, L. H., & Merchan, L. (2018). Optimization of the Inception Deck Technique for Eliciting Requirements in SCRUM Through Business Process Models. *Business Process Management Workshops, 4928*(January), 649–655. https://doi.org/10.1007/978-3-319-74030-0_52

Patidar, A., & Suman, U. (2015). A survey on software architecture evaluation methods. 2015 International Conference on Computing for Sustainable Global Development, INDIACom 2015.

Petre, M. (2013). UML in practice. Proceedings - International Conference on Software Engineering. https://doi.org/10.1109/ICSE.2013.6606618

PMI. (2017). PMI Fundamentos para la Dirección de Proyectos (guía PMBOK - 6ta edición).. https://doi.org/10.1002/pmj.20125

Pressman, R. S., & Maxim, B. R. (2015). Software Engineering : A Practitioner's Approach, Eighth Edition. In ACM SIGSOFT Software Engineering Notes. https://doi.org/10.1145/1226816.1226822

Sommerville, I. (2011). *Ingeniería del software*. Pearson 7a edición

AUTORES

Manuel Alejandro Pastrana Pardo
Magister en ingeniería de software, docente tiempo completo Facultad de Ingenierías, director del proyecto de grado y perteneciente al proyecto Smart Campus del grupo de investigación GrinTic. Correo: mapastrana@admon.uniajc.edu.co

Ana Milena Rojas
Magister docente de Carrera adscrita a la Facultad de Ingeniería y directora del proyecto Smart Campus del grupo GrinTic. Correo: amrojas@admon.uniajc.edu.co

Nicolás Ortiz
Estudiante de ingeniería de sistemas UNIAJC en espera de recibir su título de grado. Correo: nico.ortiz.a@gmail.com

Nicolás Bermúdez
Estudiante de ingeniería de sistemas UNIAJC en espera de recibir su título de grado. Correo: nocicu97@gmail.com