UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DESARROLLO DE UNA APLICACIÓN MÓVIL DE CARTELERA DE CINE

INFORME DE ACTIVIDADES PROFESIONALES

que para obtener el título de

Ingeniero en computación

P R E S E N T A

Ivan Ordoñez Ordoñez

ASESORA DEL INFORME

M.I. Ma. Del Socorro Guevara Rodríguez

Ciudad Universitaria, CDMX, 2016

Índice

Lista de tablas y figuras

Figuras e ilustraciones

Tablas

Introducción

La introducción de los teléfonos inteligentes o *smartphones* ha transformado la forma en la que las personas consumen contenidos, bienes y servicios tanto digitales como tangibles. Es precisamente este cambio de hábitos de consumo lo que ha abierto la puerta a nuevos mercados, pero también ha acercado éstos a más consumidores.

Aunque la introducción de teléfonos inteligentes es de las más recientes de entre las nuevas tecnologías, también es de las que presenta mayor adopción, tomando en cuenta que hace 10 años no existía este concepto y que en contraste hoy en día existen 51.8 millones de dispositivos solo en México, esto es aproximadamente la mitad de la población en México tiene acceso a un smartphone.

Este comportamiento tiene enormes implicaciones en la vida diaria de los mexicanos, ya que los teléfonos inteligentes se han convertido en una herramienta indispensable en la que los usuarios confían y dependen para satisfacer sus necesidades de información, entretenimiento y de consumo comercial que han llegado a coexistir, y en algunos casos inclusive sustituir, conductas cotidianas como leer periódicos, ver televisión o ir de compras.

En la actualidad no es raro ubicar personas en el día a día interactuar con sus dispositivos de bolsillo en lugares como el transporte público, cafeterías, salones de clases, plazas, tiendas e incluso en el hogar. El 94% de los usuarios móviles usa su teléfono mientras realiza una actividad como transportarse o escuchar música/radio, 73% accede a internet diariamente desde su dispositivo y prácticamente el 100% admite nunca salir de casa sin su celular. Por otro lado el 95% de los propietarios de un teléfono inteligente busca información local en su él, y el 89% realiza alguna acción posterior, como adquirir productos, visitar un establecimiento o ponerse en contacto con la empresa.

Estos nuevos comportamientos han modificado la forma de comprar de los consumidores y por ende abren una nueva gama de posibilidades para las industrias que ofrecen productos y servicios a consumidores finales, incluso las compañías que ofrecen servicios tradicionales necesitan una estrategia incluyente de las tecnologías móviles como parte central de su estrategia comercial para aprovechar la oportunidad de atraer a estos clientes potenciales.

En conclusión los smartphones constituyen una herramienta fundamental de compra, dado que el 91% de los usuarios busca productos o servicios en su dispositivo y que las búsquedas en teléfonos inteligentes influyen en las decisiones de los compradores y en las compras a través de canales. El 39% de los usuarios de teléfonos inteligentes realizó una compra a través de su teléfono siempre que consultó información acerca de un producto.

Sin embargo hasta el día de hoy no todos los comercios que tienen una presencia en línea ofrecen una experiencia optimizada para las tecnologías móviles, sino una experiencia *desktop*

adaptada al móvil en el mejor de los casos, el resto se límitan a páginas web con contenidos no aptos para móvil tales como presentaciones multimedia en tecnologías no disponibles en móvil, tales como Adobe Flash o películas QuickTime, o simplemente páginas web cuyo puerto de vista está fijo a una resolución de pantalla horizontal (desktop).

En este informe me dedicaré a exponer el caso particular de las aplicaciones móviles de carteleras de cine. En el año 2013, cuando el proyecto que aquí presento comenzó, mis colegas y yo, nos enfrentamos con un problema cuya solución disponible en ese tiempo era sin duda mejorable en muchos aspectos, la consulta de la cartelera cinematográfica y la adquisición de boletos en línea en la Ciudad de México.

Objetivo

Presentar y documentar el trabajo profesional fundamentado en la carrera de la Ingeniería en Computación realizado dados el contexto de la tecnología disponible y las soluciones en el mercado en el año 2013 y el contexto en el que la necesidad que dio lugar al desarrollo de este proyecto, asimismo se pretende presentar su planteamiento técnico y desarrollo desde el punto de vista de mi puesto de trabajo y la metodología usada en términos técnicos y de implementación para abordarlo.

Adicionalmente al finalizar este análisis, se presentarán los resultados y conclusiones obtenidas tras un poco más de 2 años de fabricar y mantener la aplicación.

Capítulo 1. Descripción de la empresa

Capítulo 1.1 Acerca de Cine+

Cine+ junta todas las taquillas de cine en una aplicación simple y fácil de usar. Dando así la mejor y más simple experiencia para adquirir boletos de cine desde internet.

Enfocado en una estrategia de desarrollo de software ágil, Cine+ construyó dos aplicaciones nativas y toda la infraestructura necesaria para poder conseguir boletos del 90% de los complejos de cine en la República Mexicana. El producto fue Lanzado en Julio del 2013, cuenta con más de 1 millón de descargas y ya ha sido destacada por su excelencia en varias ocasiones por reconocidas marcas de software como Apple, Google e Intel Software, además de hacer partnership con otras compañías como Cinemex, Santander, Scotiabank, Banamex y cines independientes en todo el país.

Cine+ construyó una aplicación para iOS y para Android que reunen la información de todas las taquillas de cine y permite la compra de boletos desde un mismo lugar sin importar el complejo de cine al que se desee ir. Los boletos se encuentran al mismo precio de taquilla y actualmente se cubre el 90% de toda la oferta salas cinematográficas tanto comerciales y no comerciales en el territorio mexicano. Se ofrece una experiencia unificada al ofrecer la mejor información en un mismo lugar y como consecuencia, se eliminan los obstáculos para conseguir entradas de cine.

Nuestros consumidores finales son aquellos que compran entradas de cine desde internet y distribuidoras de películas que buscan una mejor solución de mercadotecnia y sitios estratégicos de *product placement* especializados para sus productos.

Capítulo 1.2 Breve historia de la empresa

Los 3 fundadores de Cine+, Gustavo Delgado, José Luis Dupinet y yo, nos conocimos en la facultad de ingeniería de la UNAM y siempre buscamos la forma de extender nuestros conocimientos en tecnologías emergentes, por lo que nos centramos en el uso de tecnologías móviles que buscamos aprender en actividades extracurriculares durante nuestra formación.

Para explorar el campo, empezamos haciendo juegos sencillos para Android que publicamos en la tienda de aplicaciones Google Play, tales como “Tap Tap Piñata!” O “Pingu Slide” en diciembre de 2011 y Abril de 2012 respectivamente.

Inmersos en la filosofía del desarrollo móvil incursionamos en el desarrollo de aplicaciones móviles y mientras buscábamos aplicar los conocimientos adquiridos a la solución de problemas comunes para la ciudadanía, es así como participamos en el primer *Hackathon* en México enfocado a las tecnologías móviles y la apertura de datos del gobierno local llamado

“Ciudad Móvil” que organizó el Instituto de Ciencia y Tecnología del entonces Distrito Federal en conjunto con Aventura Capital Partners, una organización civil cuyo objetivo es impulsar los proyectos de *Open Data* en México. En este evento el objetivo era construir aplicaciones para dispositivos móviles que resolvieran algún problema de la ciudadanía con los datos abiertos proporcionados por el gobierno en alguna de las áreas de: movilidad, salud, inclusión social, cultura o participación ciudadana. Al final del evento recibimos el reconocimiento a la mejor aplicación del certamen con Metro DF, una aplicación de movilidad en transporte público.

En este evento conocimos a dos emprendedores que nos ofrecieron una oportunidad laboral en su proyecto, un startup de tecnología de transporte privado *on demand* llamado Yaxi, en donde obtuvimos una experiencia profesional más integra aplicando los conocimientos que habíamos desarrollado en los 2 años previos en un ambiente más estructurado y participando en la elaboración de un producto desde cero guiado principalmente por los requerimientos del mercado. En este proyecto construimos las aplicaciones móviles del usuario final, el operador y el backend web del servicio.

Con esta nueva experiencia adquirida y presenciando de primera mano el desarrollo del negocio en *Yaxi* nos inspiramos a emprender los 3 compañeros de la universidad juntos una empresa de software enfocados en un problema muy recurrente de la industria del comercio electrónico, el boletaje de en salas de exhibición cinematográfica.

Sabíamos que era difícil conseguir tus boletos de cine con las plataformas actuales. Las aplicaciones y sitios web de las cadenas de cine son complicadas y poco intuitivas, además de que ofrecen una experiencia fragmentada. Opinamos que las cadenas de cines son muy buenas ofreciendo un gran servicio dentro del cine. El problema es que estas compañías no tienen experiencia desarrollando software y por ende construyen sus plataformas solamente como herramientas de marketing.

Así que pensamos en Cine+, una plataforma de consulta y venta de boletos amigable, intuitiva y eficiente para smartphones. Con la que usamos nuestra experiencia construyendo plataformas enfocadas en móviles para mostrar al usuario todo el catálogo de películas disponibles en su localidad sin importar en qué complejo de cine se esté exhibiendo y le damos toda la información necesaria para que tenga el poder de tomar la mejor decisión al momento de elegir una película, un complejo y una función de cine, con la posibilidad de comprar sus boletos desde la aplicación.

Gracias al fondeo de una aceleradora de Estados Unidos llamada 500 startup, y su mentoría en temas de negocio, operación y legal constituimos una empresa en México. Actualmente Cine+ funciona en todo el país con Cinépolis y Cinemex, pero seguimos sumando salas independientes en toda la república mexicana.

Capítulo 1.3 Descripción del puesto de trabajo (Ingeniero de sistemas computacionales especializado en backend web)

Durante mi estancia laboral en Cine+ desempeñé una serie de labores varias, sin embargo para motivos de brevedad, únicamente describiré las labores relevantes para este informe relacionadas con el desarrollo de software con el título de Ingeniero de sistemas computacionales especializado en backend web.

Dado el planteamiento fundamentado en los antecedentes y la historia de la empresa, tuve la oportunidad de idear, planear, desarrollar y posteriormente escalar el sistema de la aplicación. A continuación se describe en puntos muy concretos las actividades de las que este puesto es responsable y el conjunto de habilidades técnicas que se requieren para llevar a cabo las tareas.

Actividades principales:

- Diseñar una arquitectura del sistema central escalable y de alto desempeño.
- Diseñar y administrar y mantener una base de datos.
- Diseñar y construir un sistema de transacciones concurrentes en tiempo real.
- Construir una *API rest* extensible que sirvan a las aplicaciones móviles.
- Construir endpoints de recepción de datos de funciones de cines, flexible para dar soporte a diferentes proveedores.
- Diseñar un sistema dinámico de escalamiento para reducir costos de infraestructura.
- Realizar pruebas unitarias y de estrés del API rest.

Habilidades indispensables:

- Experiencia un lenguaje de programación estructurado, programación orientado a objetos y concurrencia.
- Experiencia usando un framework web.
- Experiencia construyendo servicios web.
- Experiencia utilizando un sistema de manejo de bases de datos relacionales (RDBMS).
- Experiencia construyendo aplicaciones web sobre AWS, Google Cloud u otras plataformas.
- Conocimiento de arquitectura de servidores.
- Experiencia escalando sistemas.
- Uso habitual de control de versiones y documentación.
- Inglés técnico avanzado.

Las actividades y el flujo de trabajo están sujetas a las necesidades del manejo de producto, acoplamiento a sistemas secundarios de proveedores, tales como cines y partners estratégicos, tales como bancos y plataformas de pago, y decisiones comerciales.

Capítulo 2. Antecedentes

Actualmente se venden $1,200 millones de USD en boletos de cine al año en México. De acuerdo a eMarketer, 13% se compran a través de medios electrónicos y existe un crecimiento anual de hasta 70% por lo que en 2017, la mitad de todos los boletos serán vendidos a través de plataformas electrónicas como Cine+. Adicionalmente a esta estadística, según datos de la Cámara Nacional de la Industria Cinematográfica (CANACINE), México es el cuarto mercado más taquillero del mundo con 270 millones de boletos anuales, solo por detrás de India (2,724 mdba), Estados Unidos (1,258 mdba) y China (470 mdba) y el país más taquillero en toda América Latina, siendo que uno de cada dos boletos despachados en AL es vendido en México y de uno de cada 4 vendido también en AL es vendido solo en la Ciudad de México.

Sin embargo, el comprar boletos de cine en línea ha sido una experiencia difícil y fragmentada desde que existe la posibilidad de hacerlo por varias razones. Primeramente porque no existe un lugar en donde puedas ver toda la oferta de la cartelera que incluya a todos los cinemas. En segundo lugar no existe una forma de consultar solo los cines que se encuentran cerca de ti, sino que hay que consultar la cartelera de todos los cines de la región. La tercera razón es que en muchas ocasiones nos encontramos con el dilema de que se necesita crear una cuenta simplemente para acceder a la cartelera. Finalmente nos encontramos con que las cadenas de cines son muy buenas ofreciendo un gran servicio dentro del cine, el problema es que estas compañías no tienen experiencia desarrollando software y por ende construyen sus plataformas solamente como herramientas de marketing, dejando de lado las buenas prácticas de implementación, el diseño meticuloso de una experiencia de uso de software y ni hablar de la iteración y mejora continua de software.

Ya que el cine es un medio de entretenimiento muy casual, es muy fácil dejar de asistir por los diferentes obstáculos que involucra este proceso, por lo que es común que se desista de ir al cine y se opte por hacer otra actividad.

Con la teoría de que una aplicación móvil amigable puede ser una herramienta útil a la hora de captar y dirigir clientes potenciales a que terminen una transacción comercial aprovechando la afinidad del público mexicano por el séptimo arte, nos propusimos a reinventar la venta electrónica de entradas de cine a través del diseño de una experiencia de usuario simple y fluida que eliminará todos los obstáculos innecesarios a la hora de decidir qué mirar en el cine.

Capítulo 3. Definición del problema

El primer paso para tomar comenzar fue encontrar la necesidad base que nos lleve a pensar en un producto con características que puedan ser mejores que las opciones existentes, tal y como se plantea en los temas de introducción, antecedentes y descripción de la empresa. En resumen, una vez validado el concepto definimos el objetivo principal del proyecto como: crear una plataforma móvil enfocada a la consulta de la cartelera de cine en toda la república mexicana y la compra confiable de boletos para dichas funciones, cuyo uso sea intuitivo y que permita a los usuarios finales adquirir boletos de cine de forma segura y sencilla.

El siguiente hito fue definir la interfaz de usuario (Anexo 1) para de ahí diseñar el flujo de trabajo para cada una de las plataformas que iban a conformar el sistema:

- Aplicación de Android.
- Aplicación de iOS.
- Backend Web.

Una vez que esto quedó definido se derivaron las tareas de las que yo sería responsable para la operación de la aplicación, **las que corresponden al Backend Web**, necesario como centro de conectividad entre los clientes móviles (apps), los proveedores y las plataformas de pago.

El desarrollo del backend debe apegarse a la metodología de desarrollo elegida, apegándose al *roadmap* del producto, garantizando la calidad y mantenibilidad del código escrito, y asegurando la estabilidad de la plataforma tras cada *deploy*.

Capítulo 4. Metodología utilizada

Capítulo 4.1 Definición de requerimientos

Los requerimientos a satisfacer en el backend web se resumen a los siguientes puntos:

- Diseño de una arquitectura del sistema escalable y mantenible.
- Selección de tecnologías backend acorde a las necesidades de la plataforma.
- Diseño las entidades de base de datos necesarias para desplegar la cartelera, soportar usuarios y registrar transacciones.
- Implementación de una cola de tareas asíncrona para realizar transacciones concurrentes
- Diseño e implementación de un API rest que sea acorde al uso de las aplicaciones según el diseño del flujo de la aplicación (anexo 1).
- Sincronización de datos de las funciones de cine de las salas cinematográficas.
- Escalamiento dinámico del sistema en base al número de usuarios en la plataforma que optimice los recursos y costos.
- Pruebas unitarias y de estrés para garantizar la calidad del software antes de cada deploy
- Seguridad en el intercambio de información durante las transacciones.
- *Landpages* de películas y festivales que se indexen en google para mayor visibilidad.
- Módulo de reportería con información de uso de la plataforma para la toma de decisiones del producto.

Capítulo 4.2 Metodología de desarrollo de software

Existen diversos marcos de trabajo para el desarrollo de software, todos ellas con particularidades que se enfocan a las características del producto, el equipo de desarrollo, los recursos disponibles o la filosofía de la compañía que los ocupa.

Dado las condiciones de nuestro equipo, un equipo pequeño con urgencia de lanzar un producto mínimo viable y la necesidad de iterar rápidamente de acuerdo con la interacción y comentarios de los usuarios, modificando el comportamiento de la plataforma de acuerdo con las necesidades del producto, decidimos usar la metodología scrum debido a que:

- Es enfocado a resultados, por lo que la prioridad es el alcance de los objetivos.
- Fomenta la comunicación continua en todas las áreas de desarrollo.
- Itera de forma solapando ciclos de desarrollo, optimizando la productividad de cada ciclo.
- Facilita la toma informada de decisiones en el producto.
- Documenta por escrito las actividades y progreso del proyecto.

El método scrum contempla reuniones periódicas breves en las que un representante del producto y un mediador del equipo de desarrollo revisan las actividades y retos que cada área está cubriendo para detectar problemas potenciales, tales como dependencias entre hitos o falta de especificación del requerimiento, y con el objeto de que todo el equipo tenga una visión del avance del desarrollo. Esto potenciado por las buenas prácticas de desarrollo como:

- *Pair programming.*
- Uso de *Lints* en el código.
- Uso de control de versiones.
- Uso de servidores de integración continua.
- Implementación de pruebas unitarias y de estrés.

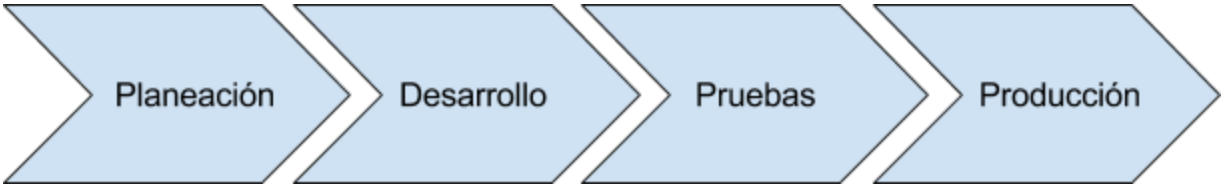


Figura 4.2.1 Flujo de trabajo del código

Capítulo 4.3 Selección de tecnologías

Una de las partes más importantes en la toma de decisiones respecto al desarrollo del proyecto es la selección de las tecnologías que serán usadas, ya que de esto depende la flexibilidad en el futuro para hacer cambios en la lógica de negocios, mantener y escalarlo de forma adecuada. Una mala elección en estas tecnologías puede resultar en cambios mayores en etapas extremadamente tempranas del desarrollo.

Por estas razones se debe seleccionar un grupo de lenguajes de programación, *frameworks*, bibliotecas y otras herramientas que se adapten a los objetivos del proyecto. En lo personal creo firmemente que el mejor acercamiento es elegir un lenguaje de programación suficientemente flexible para garantizar un desarrollo ágil y a partir de ahí elegir un framework suficientemente robusto que permita desentenderme de las tareas comunes y concentrarme en desarrollar los *features* principales, especialmente en un proyecto cuya ambición es amplio tal como éste lo es.

Una de las metas es elegir software libre que nos exentará de pagar licencias de software y que nos permitiera su uso en proyectos comerciales y que a su vez tuviera un nivel de confiabilidad y seguridad que nos blinde de problemas por lo que cada una de las elecciones de software tiene que estar probada en cuanto a su calidad y buen funcionamiento. En base a esto elegí productos que tuvieran tiempo en el mercado y hubieran sido usados por mucha gente con experiencias positivas. Los siguientes productos cumplen por mucho con estos criterios:

Existen pocos proyectos libres tan confiables como lo es el proyecto Ubuntu, la distribución multipropósito más popular de linux y por mucho la mejor documentada y mantenida por una comunidad global, por lo que la elegí como sistema operativo de los servidores web en este proyecto.

Para la elección de lenguaje de programación, elegí Python por su facilidad de lectura del código escrito que se asemeja al lenguaje natural, lo que lo hace fácil de aprender y fácil de mantener, además de que cuenta una gran comunidad de desarrolladores que han forjado un gran número de bibliotecas y *frameworks* de código abierto para trabajar en múltiples escenarios, cosa que lo convierte en uno de los lenguajes de programación multipropósito más completos.

Python posee una de los frameworks web más populares y completos que es Django, que permite el rápido desarrollo, reduce el número de amenazas de seguridad y está pensado para ser escalable en número de usuarios. Django cuenta con un Object-Relational Mapping (ORM) que le permite trabajar con múltiples bases de datos, ya sea relacionales o no relacionales, sin depender de queries específicos del sistema manejador de base de datos que se esté utilizando, también cuenta con un engine de templates que permite la generación de páginas web en tiempo real y un administrador web tipo CRUD que permite acceder a los datos de la DB de manera más eficiente que desde el DBMS. Adicionalmente se le pueden agregar plugins para añadir funcionalidades como *caché*, *query debuggers* y *request debuggers* entre otros.

En resumen el stack de las tecnologías base para el servidor es:

- Python v2.7.
- Django v1.4.22 (LTS).
- Ubuntu v14.04 (LTS).

Capítulo 4.4 Arquitectura del sistema

Al tomar decisiones acerca de la arquitectura que sostendrá el desarrollo de un sistema cuyos requerimientos principales son mantenibilidad y escalabilidad, se necesita pensar a futuro ya que al igual que la elección de software una serie de decisiones tomadas a la ligera pueden resultar en cambios mayores en un futuro cercano e inclusive tiempos offline del sistema en producción, así como pérdida de datos y de usuarios.

Para evitar estas complicaciones lo más recomendable es mantener una capacidad de respuesta superior a la demanda en todo momento en el servidor web, sin embargo al ser variable la demanda este acercamiento puede llevar a tiempos muertos de procesamiento y por ende altos costos de infraestructura mal aprovechados. Es por esta situación la mejor aproximación es idear un sistema que de manera dinámica, regule la capacidad de respuesta del sistema en base a la demanda actual para optimizar los costos de infraestructura. Existen sistemas que ofrecen este servicio ya implementado pero a un costo considerablemente mayor

que el servicio regular tales como *Elastic Beanstalk* de *Amazon Web Services* o *Autoscaling* de *Google App Engine* por mencionar algunos.

Debido a que el principal objetivo es reducir el costo de los servidores la primera elección que tomé fue la de el proveedor de la infraestructura que es Amazon Web Services el cual tiene una gran flexibilidad debido a que cuenta con muchas opciones configurables de servidores a precios accesibles. Comencé por mantener un servidor y una base de datos simples (figura 6.4.1), sin escalamiento, y posterior al lanzamiento implementé el autoescalamiento con ayuda de un *load balancer* (figura 6.4.2), que como su nombre lo indica balancea la carga de trabajo entre los servidores a través de un enrutador interno dando más trabajo a los servidores menos saturados y aligerando la carga de los más saturados y de ser necesario levanta más servidores para evitar que todos los servidores se saturen y en caso de que la carga de trabajo sea muy ligera, da de baja los servidores sobrantes para mantener activos solamente el número de servidores necesarios para atender los *requests* entrantes.

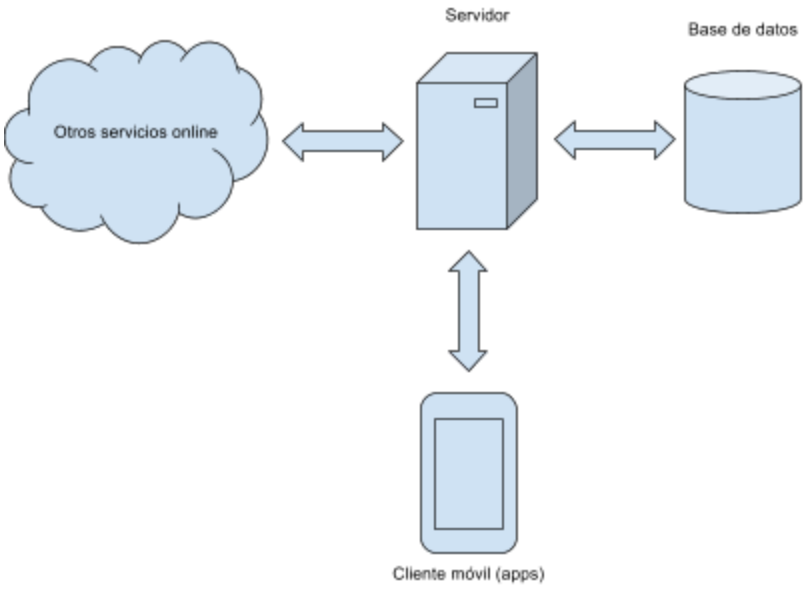


Figura 4.3.1 Arquitectura básica de aplicación móvil

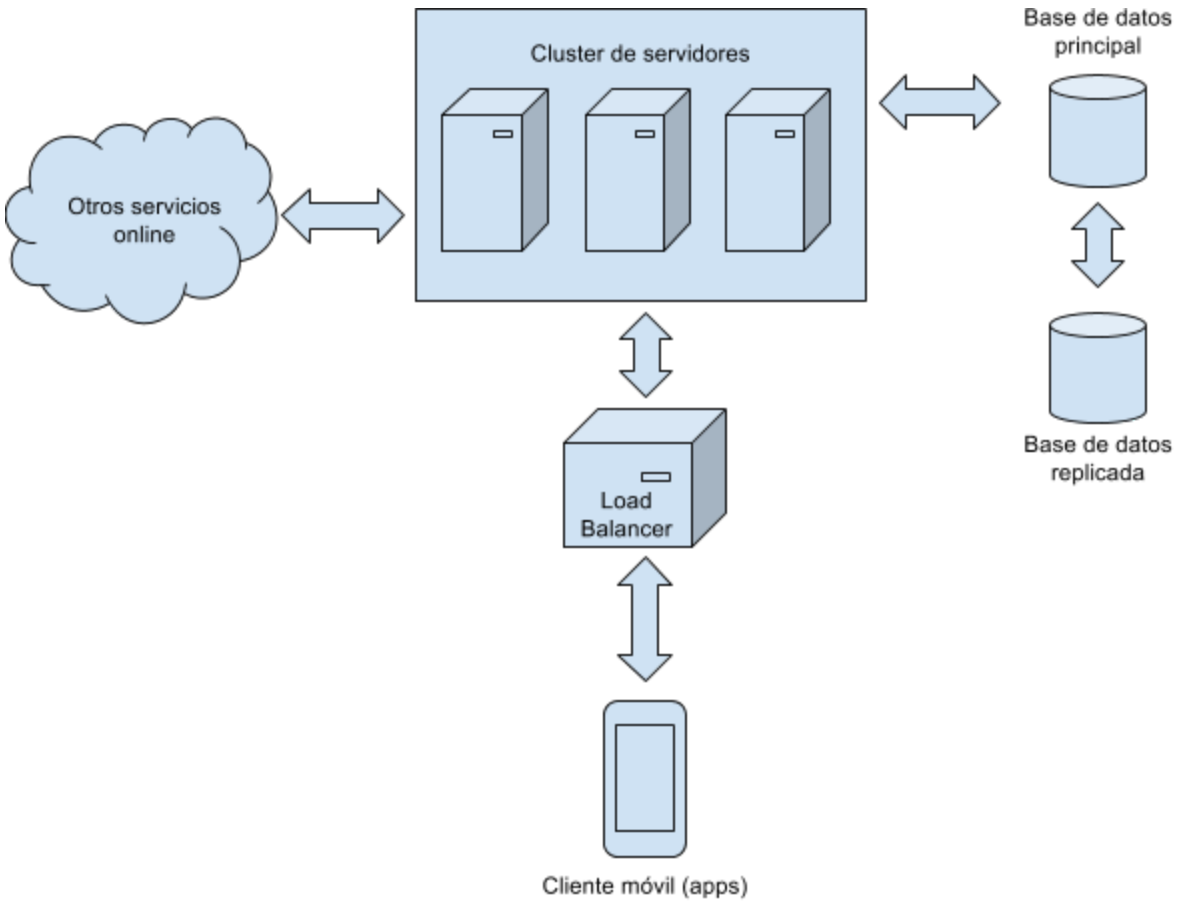


Figura 4.3.2 Arquitectura de aplicación móvil con load balancer

El *loadbalancer* no es más que un servidor ubuntu más que a través del API de AWS monitorea el uso de cada uno de los servidores activos para establecer el nivel de salud de cada uno, además tiene permiso al repositorio del código y el servidor de integración continua desde la cual requiere una copia cada vez que levanta una nueva instancia dando entrada a un nuevo servidor con el código más nuevo y estable, dando paso también a *rollouts* paulatinos de nuevas versiones del sistema y en caso de ser necesario reemplaza todos los servidores en funcionamiento a una versión específica del código a través de un comando manual para dar paso así a *deployments* uniformes o en caso de ser necesario *rollbacks* a versiones anteriores estables del código.

Capítulo 4.5 Implementación del sistema

Capítulo 4.5.1 Diseño de base de datos

Requerimiento a satisfacer: Diseñar un esquema de base de datos con las entidades necesarias para desplegar la cartelera, soportar usuarios y registrar transacciones.

Selección de tecnología: Como PostgreSQL por ser un sistema manejador de base de datos rápido, replicable y compilante a las especificaciones de lenguaje de consultas relacionales estándar. El esquema de la base de datos es generado a través del ORM.

Consideraciones técnicas: Elegí un RDBMS relacional a pesar del inconveniente de tener que hacer migraciones que dificultan los *deployments* incrementales del esquema de la DB por garantizar mayor consistencia en los datos e implementación sencilla de una base de respaldo que replica el contenido de la base principal. El modelado de las entidades se hace a través de el ORM de Django, que permite modelar a través de una interfaz de programación orientada a objetos el esquema de la base de datos lo que genera el esquema de la base de datos y las migraciones producidas por los cambios consecuentes.

```python
class Cadena(models.Model):
    nombre = models.CharField(
                max_length=128,
                unique=True
             )
```

Figura 4.5.1.1 Código ORM Django (Python)

```sql
CREATE TABLE `cinema_cadena` (
        `id`    int(11)    NOT    NULL
AUTO_INCREMENT,
  `nombre` varchar(128) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `nombre` (`nombre`)
)    ENGINE=InnoDB    AUTO_INCREMENT=5
DEFAULT CHARSET=utf8
```

Figura 4.5.1.2 Código SQL generado

Con respecto a las migraciones, estas tienen que hacerse con cautela ya que un *deployment* inadecuado en el que difiera la descripción de las entidades en el ORM con el esquema actual de la base de datos es proclive a errores de la aplicación web en tiempo de ejecución, lo cual genera *downtimes.*

En cuanto a la planificación de entidades a continuación se muestra una simplificación del esquema de la base de datos. La base de datos real contiene más entidades que contemplan detalles de la operación y otros features que no son relevantes para este informe.

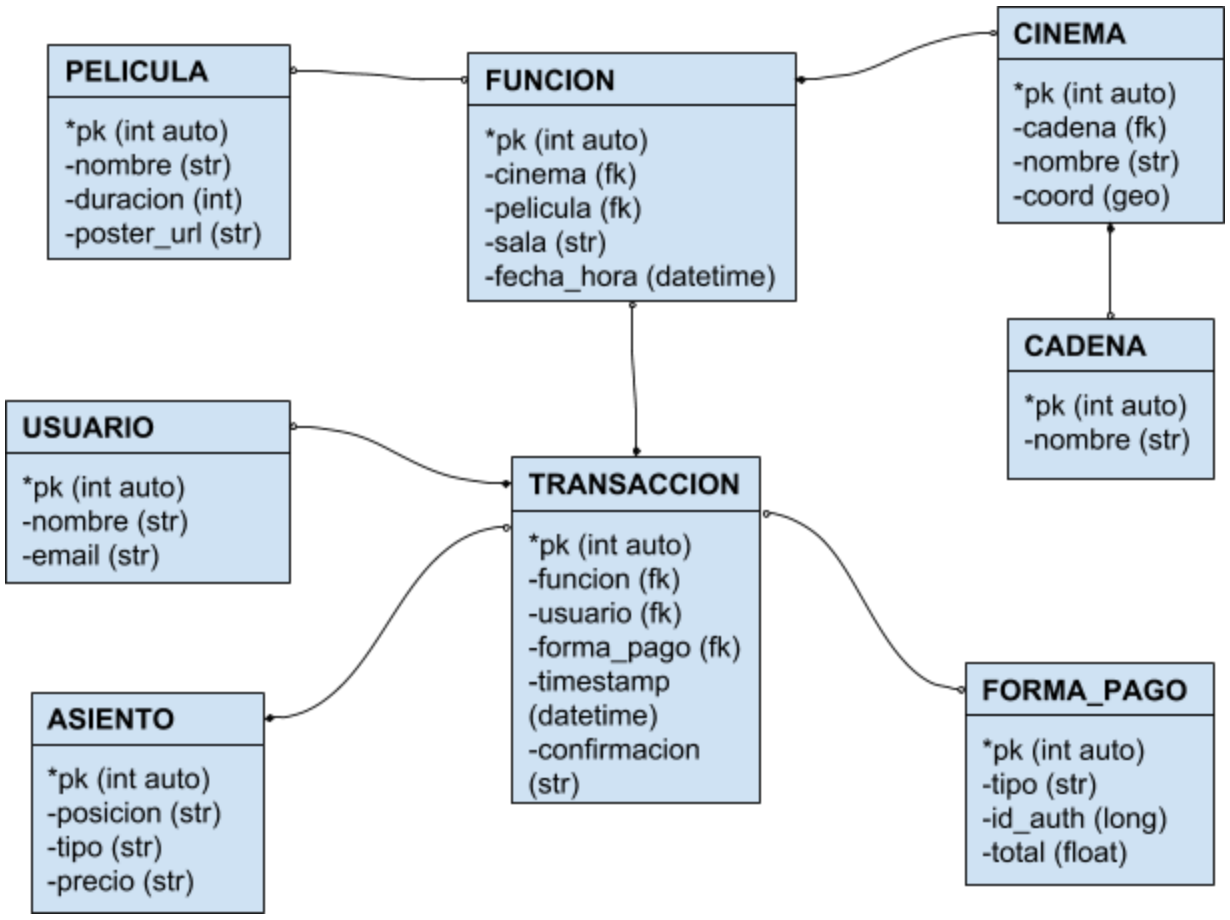


Figura 4.5.1.3 Diagrama de entidades de la base de datos

Capítulo 4.5.2 Colas de tareas

Requerimiento a satisfacer: Implementación de una cola de tareas asíncrona para realizar transacciones concurrentes.

Selección de tecnología: Un grupo de aplicaciones comúnmente usada para este tipo de situaciones es Rabbit MQ como medio de comunicación entre aplicaciones, Celery como cola de de tareas y django-celery como interfaz para conectarlo con la aplicación web de django.

Consideraciones técnicas: La operación transaccional que implica la plataforma requiere de un variado número de procesos que van desde la consulta de asientos disponibles en una sala de cine, hasta la solicitud de transacciones bancarias correspondientes al pago de boletos. Para que esto ocurra de manera amigable y transparente para el usuario final, estos procesos ocurren de manera asíncrona y al mismo tiempo que otras operaciones tratando de evitar bloquear la interfaz de usuario de forma que éste no se sienta restringido en la medida de lo posible.

Para lograr dicho objetivo se requiere un mecanismo que permita la comunicación de distintas partes del sistema de manera concurrente bloqueando aquellas peticiones que puedan producir duplicidad de datos, tales como vender las mismas localidades varias veces, duplicar cargos o sobrevender las funciones.

En este apartado diseñe el mecanismo ayudado de banderas booleanas en las bases de datos que pudieran ser consultadas antes de activar los mecanismos de transacciones para evitar problemas de duplicidad, una vez verificado este comportamiento, una tarea asíncrona y autónoma sería puesta en marcha que al ser resuelta comunica el resultado al proceso responsable de notificar al cliente móvil y desbloquea la bandera correspondiente de la base de datos.

Capítulo 4.5.3 API Rest

Requerimiento a satisfacer: Diseño e implementación de un API rest que sea acorde al uso de las aplicaciones según el diseño del flujo de la aplicación (anexo 1).

Selección de tecnología: Memcaché, para implementar caché de las queries a la base de datos. JSON, una librería de python para serializar estructuras de datos a formato json y viceversa. SSL, una capa de seguridad que permite el intercambio de datos sobre el protocolo HTTP cifrando los datos que se transmiten entre el cliente y la aplicación web.

Consideraciones técnicas: Dentro de las partes más importantes del proyecto está el diseñar e implementar de forma segura y eficiente un mecanismo de comunicación entre el frontend móvil y el servidor que hospeda la aplicación web para sincronizar la información y efectuar transacciones. El acercamiento que elegí es implementar un API rest, que no es más que una interfaz de programación web de transmisión de información, basada en un grupo de URLs dentro un servidor que representan acciones ligadas a una entidad de datos y que utiliza los métodos del protocolo de aplicación HTTP según el tipo de acción a realizar como lo especifica la arquitectura REST.

Para comenzar con esta aproximación, primero definí un conjunto de entidades para las que se pueden realizar acciones:

- Usuario.
- Transacción.
- Pelicula.
- Funciones.
- Cinema.

Luego se definen las acciones que se pueden realizar desde el API por cada una de las entidades, basado en el modelo CRUD (crear, consultar, actualizar, eliminar).

Tabla 4.5.3.1 Tabla de acciones sobre entidades

	Crear	**Consultar**	**Actualizar**	**Eliminar**
Usuario	Sí	Sí	Sí	No
Transacción	Sí	Sí	No	No
Película	No	Sí	No	No
Funciones	No	Sí	No	No
Cinema	No	Sí	No	No

Enseguida procedemos a crear los *endpoints* basados en cada una de las acciones, las cuales serán mapeadas en el método HTTP correspondiente para obtener los siguientes:

Tabla 4.5.3.2 Lista de endpoints del API rest

Endpoints de usuario		
POST	/api/usuario/	Genera un nuevo usuario
GET	/api/usuario/{id_usuario}	Consulta los detalles del usuario dado
PUT	/api/usuario/{id_usuario}	Modifica los detalles del usuario dado
Endpoints de transacción		
POST	/api/transaccion/	Genera una nueva transacción
GET	/api/transaccion/[id_transaccion]	Lista las transacciones. En caso que se especifique un id de transacción, lista los detalles de ésta
Endpoints de película		
GET	/api/pelicula/[id_pelicula]	Lista las películas. En caso que se especifique un id de película, lista los detalles de ésta
Endpoints de función		
GET	/api/funcion/[id_funcion]	Lista las funciones. En caso que se especifique un id de función, lista los detalles de ésta
Endpoint de cinema		
GET	/api/cinema/[id_cinema]	Lista los cinemas. En caso que se especifique un id

		de cinema, lista los detalles de éste

Finalmente para cada uno de los endpoints se genera la documentación o request de ejemplo especificando los parámetros adicionales necesarios para cada uno de los casos, tal y como se muestra en el ejemplo de documentación de un servicio web mostrado en el anexo II de este documento.

Los resultados de las peticiones regresan un objeto json gracias a la serialización de la librería json. Como medida de optimización en el tiempo de respuesta de los request web implementé un caché de las queries a la base de datos. Para agregar una capa de seguridad contra ataques maliciosos que afecten el rendimiento de la API también implementé oauth en los endpoints del API para evitar requests no autorizados a ésta.

Capítulo 4.5.4 Web Scrapers

Requerimiento a satisfacer: Construir endpoints de recepción de datos de funciones de cines flexible para diferentes proveedores.

Selección de tecnología: Utilicé la cola de tareas que implementé en el punto 6.5.2 para ejecutar las tareas de manera independiente y librerías nativas para parsear datos en formato xml, json y csv.

Consideraciones técnicas: En el evento de la programación de funciones de cine existen tres consideraciones que hay que tomar en cuenta:

- Los cinemas programan con diferente periodicidad sus funciones.
- Los formatos de la información de las funciones de cada cadena varían.
- La fuente de información puede venir desde un servicio web, una página de internet o un archivo CSV.

En cada caso la información debe ser interpretada, guardada y actualizada en nuestra base de datos a fin de mantener las funciones actualizadas siempre.

Para lograr esto diseñé un mecanismo bicapa con una interfaz común para todas las cadenas en la primera capa y una implementación particular para cada caso en la segunda.

La interfaz de cada una de los cinemas procesa la información y la convierte en un formato estándar para poder ser validada por la siguiente etapa que descarta la información de funciones pasadas o no confirmadas y actualiza las aquellas cuya información ha cambiado.

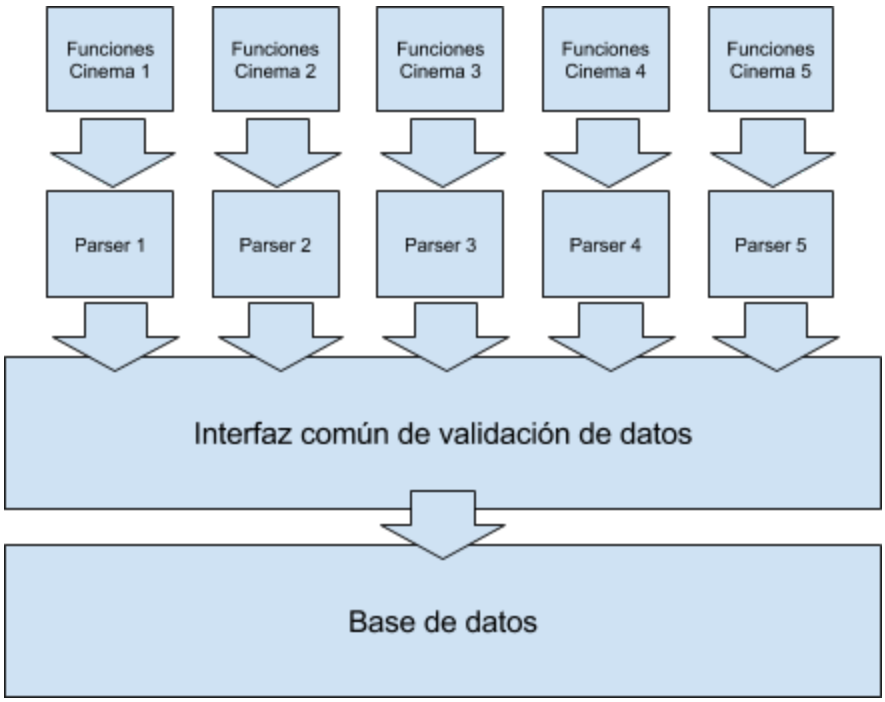


Figura 4.5.4.1 Flujo de información de funciones, desde el cinema hasta la base de datos

Adicionalmente, programé una tarea para cada una de las cadenas de cine para que automáticamente se actualice la información dependiendo de cada cadena. Esta actualización puede ocurrir desde una vez al mes hasta varias veces al día dependiendo de la necesidad de la operación de cada cine.

Este formato hace extensible la operación al hacer fácil el agregar y quitar cines de la cartelera sin tener que modificar el código fuente común.

Capítulo 4.5.5 Pruebas unitarias y pruebas de estrés

Requerimiento a satisfacer: Realizar pruebas unitarias y de estrés del API rest.

Selección de tecnología: Unit test de django para las pruebas unitarias del API REST y para las pruebas de estrés utilicé Siege, una herramienta open source que realiza test de carga de peticiones para medir el desempeño de la arquitectura del servidor en parámetros como el tiempo de respuesta y el status devuelto.

Consideraciones técnicas: Para medir la calidad del software entregado tras cada iteración es necesario realizar pruebas que garanticen la estabilidad y la buena implementación del software desarrollado. Para saberlo hay que tener en cuenta que:

- El software cumpla con los requerimientos tras cada actualización.
- El desempeño del servidor sea bueno.

El primer punto parece trivial sin embargo al iterar en el proceso de desarrollo por mantenimiento o para integrar nuevos features pueden existir modificaciones al código fuente que inciden en los procesos o servicios existentes y que modifiquen su comportamiento, por lo que es necesario comprobar que las actualizaciones del código fuente no modifiquen los requerimientos ya satisfechos en la plataforma a través de pruebas unitarias (ver anexo III).

Por otro lado, también es recomendable tener en cuenta el desempeño del servidor, el cual medimos a través de pruebas de estrés aplicadas a un endpoint o varios, en los cuales se obtendrán medidas cualitativas y cuantitativas que nos indican el rendimiento del servidor. La información que obtenemos es:

- Tiempo de respuesta del servidor.
- Número de request que se pueden atender por segundo antes de que el performance comience a ser afectado.

Con esta información podemos inferir si el sistema está demorando demasiado y si necesitamos optimizar el código para obtener un mejor rendimiento (ver anexo IV).

Capítulo 5. Resultados

Posterior al desarrollo de la primera versión y el lanzamiento obtuvimos un crecimiento bastante acelerado en número de usuarios y por consiguiente en número de requests a la plataforma, que demandó bastantes optimizaciones a la plataforma como la introducción de réplica de base de datos, caché e introducción de características de seguridad tales como ssl y oauth.

Durante los primeros seis meses de vida de la plataforma pudimos apreciar cómo se alcanzó una tasa de crecimiento bastante considerable para un producto de esta naturaleza, lo que hablaba bien de nuestro concepto original y que validaba nuestras primeras hipótesis del producto. En los primeros 6 meses alcanzamos más de 100 mil usuarios comenzando desde cero, además de rebasar el hito de más de 1 millón de sesiones de la aplicación mensualmente y las casi 10 millones de consultas al api rest al mes de donde la mayor actividad registrada es el fin de semana y días feriados dado que la aplicación entra en la categoría de entretenimiento.

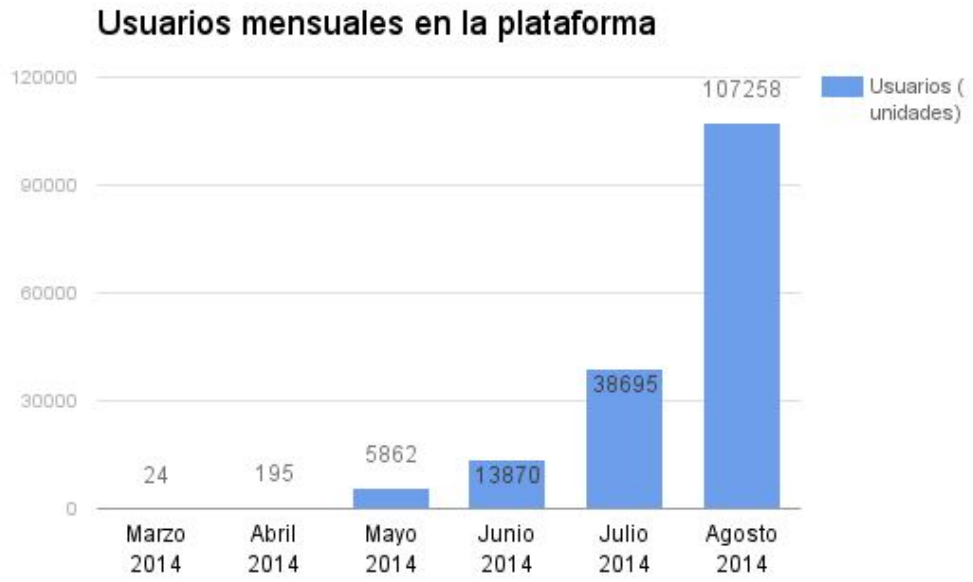


Figura 5.1 Gráfica de crecimiento de usuarios mes a mes

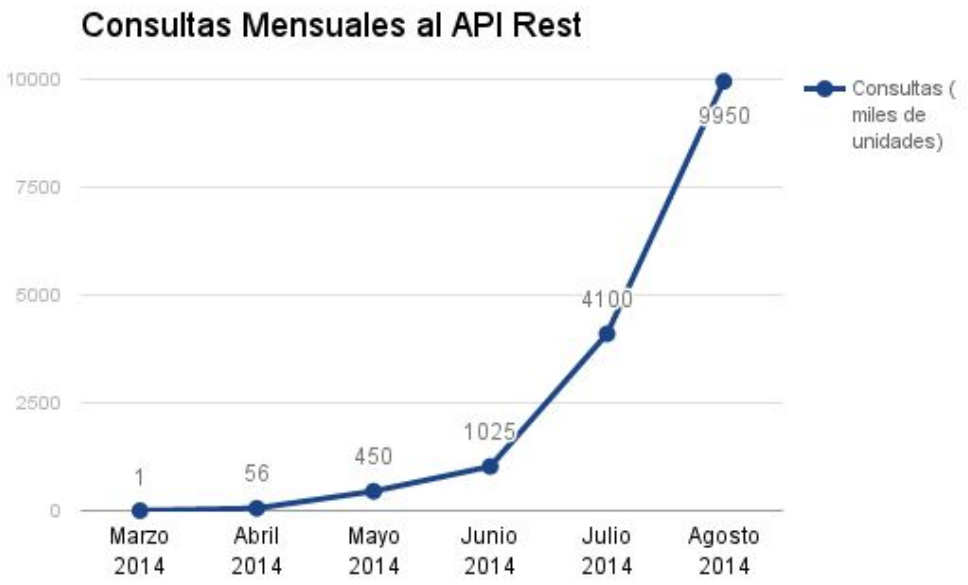


Figura 5.2 Gráfica de crecimiento en el uso de la plataforma vía API rest

Dada la naturaleza y el cuidado del flujo de desarrollo y la técnica de los deployments se alcanzó una tasa de 100% *uptime* en la plataforma en este mismo periodo además de que la replicación de la base de datos nos ayudó a obtener una alta fidelidad en los datos que se generan en la plataforma.

Sin embargo otros tipo de errores en el código tales como los diferentes husos horarios que rigen el país trajo problemas de operación los cuales se arreglaban a través de servicio a cliente revisando caso por caso, lo cual no es escalable dado el tiempo que se invierte en cada caso, por lo que este tipo de fallos eran prioridad en nuestras reuniones scrum.

Conclusiones

El enfoque práctico que es obtenido durante la carrera de ingeniería en computación podría no parecer aplicable directamente en su basta mayoría ya que en la práctica se presentan dilemas complejos muy diferentes a aquellos que se plantean en la teoría sin embargo la teoría ayuda a comprender extensivamente los retos que se presentan y fundamentar la creación de soluciones óptimas, simples y mantenibles.

Una de las cosas que me ha quedado muy clara es que al diseñar una plataforma con una proyección de gran escala, ésta debe ser planteada cuidadosamente. Sin embargo desde el punto de desarrollo ágil no siempre se puede construir todo desde el principio o siquiera darle prioridad al uso ejemplar de las tecnologías de información, muchas veces factores externos como la mercadotecnia, la inteligencia de negocio o la visión de la empresa se anteponen, sólo el tiempo y el desarrollo de las necesidades del negocio pueden dirigir el rumbo del manejo del producto.

Las tecnologías de la información se han convertido en el corazón de la operación empresarial por lo cual he tenido la oportunidad de convivir con un equipo multidisciplinario con diferente tipo de formación, lo que me ha ayudado a mejorar mis habilidades de comunicación y ampliar mi punto de vista en la resolución de problemas de ingeniería, en lo particular he podido ver de cerca el desarrollo y ejecución de un plan de negocio.

A partir de esta nueva visión he aprendido que la toma de decisiones en el desarrollo siempre debe ser fundamentada en la información que genera la plataforma durante su uso, por lo cual es fundamental el tomar lectura de todas las acciones que toman lugar tanto en los clientes como en los servidores, para posteriormente generar informes que faciliten la comprensión de los resultados.

En otros aspectos, es igual de importante la actualización continua de conocimientos, por lo cual la lectura de documentación o artículos, así como la presencia a seminarios, expos o conferencias de prensa es muy importante para entender la nueva realidad de las tecnologías de la información.

Glosario

Aceleradora: También conocida como aceleradora de startups, es un programa de incubación de empresas que incluye componentes de mentoría y financiamiento que culmina en un evento público llamado *demo day.*

API: Por sus siglas en inglés, *Application Programming Interface*, en español Interfaz de Programación de la Aplicación, es un set de protocolos y herramientas diseñadas para la comunicación de aplicaciones externas a una plataforma dada.

Backend: En ingeniería y arquitectura de software se refiere a la capa de acceso de datos de la aplicación.

Cluster: Un grupo de computadoras autónomas que trabajan juntas para alcanzar un fin específico.

Control de versiones: Es un software manejador incremental de cambios en el código fuente de un proyecto.

CRUD: Funciones básicas de un manejador de base de datos (create, read, update, edit) y nombre que se le da a un software administrador de estas funciones en la práctica.

Deployment: Todas las actividades cuyo resultado sea el funcionamiento de un sistema de software.

Desktop: Cliente de software de escritorio para usuarios finales llamado así para ser distinguido de los clientes móviles.

Downtime: Término usado para referirse a los periodos en los cuales un sistema de software no está disponible.

Endpoint: Entidad de una de las terminales de conexión de la capa de transporte del software.

Feature: Característica desarrollada en software que satisface un requerimiento.

Framework: Un conjunto reusable de librerías, clases o componentes de software en general que se encargan de manejar tareas específicas comunes en un sistema de software.

Frontend: En ingeniería y arquitectura de software se refiere a la capa de presentación de datos de la aplicación.

Hackathón: Del inglés *Hackathon*, es un evento en el que equipos de desarrollo de software colaboran intensamente en proyectos de software por periodos continuos y prolongados.

Integración continua: Es la práctica de integrar y probar todas las versiones de desarrollo de un proyecto de software regularmente en periodos que van desde una vez, hasta várias al día.

Lint: Una pieza de software que sirve para detectar problemas en el código fuente que van desde commandos ambiguos hasta malas prácticas de desarrollo.

On demand: En operación de negocios, se refiere a una metodología de operación en la que la oferta es satisfecha conforme se presenta la demanda.

ORM: De las siglas inglesas *Object-Relation Mapping*, se puede referir, entre otras cosas, a la herramienta que convierte datos de sistemas incompatibles en objetos de lenguajes de programación orientados a objetos.

Pair programming: Técnica de desarrollo ágil de software en la cual dos programadores trabajan juntos en la misma estación de trabajo en la que uno escribe el código mientras

que el otro observa y hace recomendaciones mientra el código es escrito, esta dinámica continúa y los roles son cambiados frecuentemente.

Product placement: Una técnica de publicidad usada por compañías con el objetivo de sutilmente promocionar sus productos a través de técnicas de publicidad no tradicionales como la aparición de productos en una película u otro tipo de media.

Puerto de vista: En teoría de computación gráfica, es una región poligonal relevante para el usuario.

Replicación (de datos): Es la técnica por la cual una base de datos respalda su información en otra base de datos auxiliar en tiempo real.

Roadmap: En ingeniería de software, se refiere a un plan de visión del producto que indica a detalle las etapas de desarrollo del proyecto.

Rollback: La actividad de desplegar una nueva versión de software.

Rollout: La actividad de regresar a una versión anterior del software.

Scrum: Una técnica variante de la metodología ágil de software.

Unit testing: En español prueba unitaria, es un método de pruebas de software unidades de código fuente diseñado para probar la estabilidad y efecto del código fuente objetivo son ejecutadas y cuyos resultados son evaluados para generar un veredicto binario (passed/not passed).

Referencia

Instituto Nacional de Estadística y Geografía (INEGI), “Estadística a propósito del Día Mundial del Internet 2016. Datos nacionales” [en línea] Consultado el 30 Octubre 2016. url:http://www.inegi.org.mx/saladeprensa/aproposito/2016/internet2016_0.pdf

Google, “Our Mobile Planet - México 2013”, [en línea] Consultado el 30 Octubre 2016. url:https://think.storage.googleapis.com/intl/es-419_ALL/docs/our-mobile-planet-mexico_research-studies.pdf

Camara Nacional de la Industria Cinematográfica (CANACINE), “Resultados definitivos 2015” [en línea] Consultado el 31 Octubre 2016. url:http://canacine.org.mx/wp-content/uploads/2014/04/Resultados-Definitivos-2015-ATI-1-1.pdf

eMarketer, “Mexico online” [en línea] Consultado el 1 Noviembre 2016. url:http://www.slideshare.net/Engelnator/e-marketer-mexico-online-2009-presentation

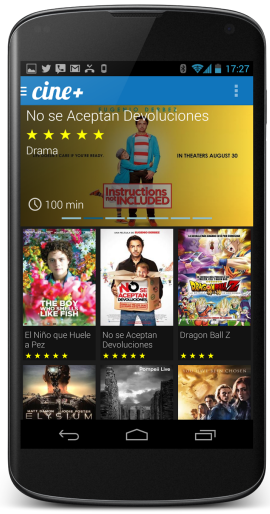
iApps Gratis, “Cinépolis (2013)” [en línea] Consultado el 29 Octubre 2016. url:www.iappsgratis.com/2012/07/cinepolis.html

Django Software Foundation “Django documentation” [en linea] Consultado 1 noviembre de 2016.
url:https://www.djangoproject.com/

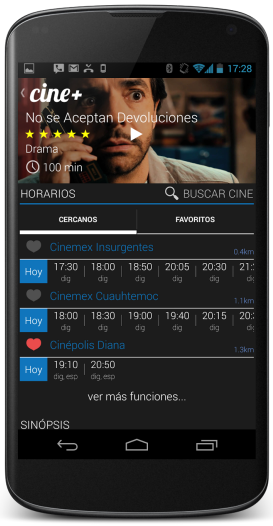
Anexos

Anexo A. Flujo principal de la aplicación v1.0

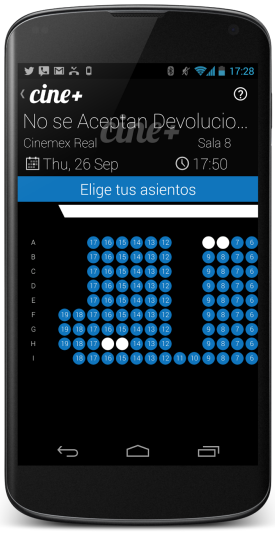
Paso 1. Cartelera



Paso 2. Horarios



Paso 3. Selección de asientos



Paso 4. Checkout

Paso 5. Confirmación

Opcional. Inicio de sesión

Anexo B. Ejemplo de documentación de un web service

GET /api/funcion/

Devuelve un array en formato json de objetos tipo función de la película indicada en los cines próximos al radio de la búsqueda.

Request Example

```
$ curl https://www.example.com/api/funcion?\
      lat=19.432576&\
      lon=-99.133577&\
      rango_busqueda_km=4&\
      id_película=3ddb350ba890ff897ae001cd&\
      oauth_consumer_key=xvz1evFS4wEEPTGEFPHBog&\
      oauth_nonce=kYjzVBB8Y0ZFabxSWbWovY3uYSQ2pTgmZeNu2VS4cg&\
      oauth_timestamp=1318622958&\
      oauth_token=370773112-GmHxMAgYyLbNEtIKZeRNFsMKPR9EyMZeS9weJAEb
```

Nodo	Descripción
lat	**Float**. Latitud geográfica en la que se realizará la búsqueda

lon	**Float**. Longitud geográfica en la que se realizará la búsqueda
rango_busqueda_km	**Integer**. Radio de búsqueda de funciones en kilómetros
id_película	**String**. Identificador único de una película
oauth_consumer_key	**String**. Llave única asociada a un usuario
oauth_nonce	**String.** Hash derivado del contenido del request
oauth_timestamp	**Integer**. Representación de la fecha y hora de creación del request
oauth_token	**String.** Token de sesión generado por el servidor

Anexo C. Ejemplo del código de pruebas de unitarias en Django

```
from django.test import TestCase
from myapp.models import Transaction, Cadena, Cinema, Usuario, Funcion

class TransactionTestCase(TestCase):
    def setUp(self):
        """Creando entidades de prueba"""
        self.cadena = Cadena.objects.create(nombre="Cadena prueba")
        self.cinema =Cinema.objects.create(nombre="Cinema prueba",
            cadena=self.cadena,  coordenada=Geo(latitude=0.0,
                longitude=0.0))
        self.pelicula = Pelicula.objects.create(nombre="Pelicula prueba",
            poster="http://example.com/images/placeholder.jpg", duracion=90)
        self.usuario = Usuario.objects.create(nombre="Usuario Prueba",
            email="test@example.com")
        self.funcion = Funcion.objects.create(cinema=self.cinema,
            pelicula=self.pelicula, fecha_hora=datetime.now())

    def test_transaction(self):
        """Realizando transacción y validando su entrada en la db"""
        url = reverse('transaction')
        data = {usuario: json.dumps(self.usuario), 'funcion':
            json.dumps(self.funcion)}
        response = self.client.post(url, data, format='json')
        self.assertEqual(response.status_code, status.HTTP_201_CREATED)
        self.assertEqual(Transaccion.objects.count(), 1)
        self.assertEqual(User.objects.get().nombre, self.user.nombre)
        self.assertEqual(Cinema.objects.get().nombre, self.cinema.nombre)
        self.transaccion = Transaccion.objects.get()

def clean(self):
        """Limpiando las entidades creadas"""
```

```
        self.transaccion.delete()
        self.funcion.delete()
        self.cinema.delete()
        self.cadena.delete()
        self.usuario.delete()
        self.pelicula.delete()
```

Test unitario de el endpoint de transaccion (Python)

Anexo D. Estadísticas del software Siege

```
$ siege -u www.example.com/api/funcion/f2532ab7ce3b5b2 -d1 -r10 -c25
..Siege 2.65 2016/15/11 23:42:16
..Preparing 25 concurrent users for battle.
The server is now under siege...done
Transactions: 250 hits
Elapsed time: 14.67 secs
Data transferred: 448000 bytes
Response time: 0.43 secs
Transaction rate: 17.04 trans/sec
Throughput: 30538.51 bytes/sec
Concurrency: 7.38
Status code 200: 250
Successful transactions: 250
Failed transactions: 0
```

Resultados en consola de Siege. Ejemplo de 25 hilos de 10 pruebas cada uno