UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingeniería

Aplicación web para el control de producción de Remu Apparel

Felipe Gabela Ramires

Ingeniería en Sistemas

Trabajo de fin de carrera presentado como requisito
para la obtención del título de
Ingeniero en Sistemas

Quito, 6 de Mayo de 2020

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingeniería

HOJA DE CALIFICACIÓN
DE TRABAJO DE FIN DE CARRERA

Aplicación web para el control de producción de Remu Apparel

Felipe Gabela Ramires

Nombre del profesor, Título académico Fausto Pasmay, MS

Quito, 6 de mayo de 2020

Derechos de Autor

Por medio del presente documento certifico que he leído todas las Políticas y Manuales de la Universidad San Francisco de Quito USFQ, incluyendo la Política de Propiedad Intelectual USFQ, y estoy de acuerdo con su contenido, por lo que los derechos de propiedad intelectual del presente trabajo quedan sujetos a lo dispuesto en esas Políticas.

Asimismo, autorizo a la USFQ para que realice la digitalización y publicación de este trabajo en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Nombres y apellidos: Felipe Gabela Ramires

Código: 124284

Cédula de identidad: 1715810279

Lugar y fecha: Quito, mayo de 2020

ACLARACIÓN PARA PUBLICACIÓN

Nota: El presente trabajo, en su totalidad o cualquiera de sus partes, no debe ser considerado como una publicación, incluso a pesar de estar disponible sin restricciones a través de un repositorio institucional. Esta declaración se alinea con las prácticas y recomendaciones presentadas por el Committee on Publication Ethics COPE descritas por Barbour et al. (2017) Discussion document on best practice for issues around theses publishing, disponible en http://bit.ly/COPETheses.

UNPUBLISHED DOCUMENT

Note: The following capstone project is available through Universidad San Francisco de Quito USFQ institutional repository. Nonetheless, this project – in whole or in part – should not be considered a publication. This statement follows the recommendations presented by the Committee on Publication Ethics COPE described by Barbour et al. (2017) Discussion document on best practice for issues around theses publishing available on http://bit.ly/COPETheses.

RESUMEN

En la última década la industria del comercio ha cambiado drásticamente de tiendas físicas a tiendas virtuales. La oportunidad de llegar a más mercados y la ventaja de vender 24 horas al día, 7 días a la semana, sin los elevados costos de alquiler y personal, hacen que las tiendas virtuales, o eCommerce, ganen actualmente fuerte tracción. Para abastecer la creciente demanda de tiendas virtuales, muchas empresas ofrecen tiendas virtuales *as-a-service.* La mayoría de las plataformas *as-a-service* son económicas y fáciles de usar, pero carecen de funcionalidades que se adapten al giro de negocio único de cada empresa. Este trabajo de titulación propone diseñar, desarrollar e implementar una aplicación cliente-servidor que extiende las funcionalidades de una tienda online en la popular plataforma de eCommerce *as-a-service* Shopify, de la empresa de ropa Remu Apparel. La aplicación extenderá funcionalidades de control de producción de la empresa que no son satisfechas por Shopify. La aplicación simplificará el proceso de producción de la empresa y evitará errores humanos de transmisión de información. Para el diseño de la aplicación se utilizarán varias herramientas como el diagrama de caso de uso y el diagrama de flujo de datos. El desarrollo de la aplicación se realizará en Python empleando el framework Django para desarrollo rápido. El despliegue de la aplicación se hará en Heroku con el fin simplificar el proceso de despliegue y enfocarse más bien en la aplicación en si. La aplicación se conectará con una tienda de pruebas de la compañía y en el futuro la empresa deberá evaluar la conexión con la tienda virtual real de la compañía.

Palabras clave: eCommerce, Shopify, aplicación cliente-servidor, control producción, Django, Heroku, PostgreSQL

ABSTRACT

During the last decade retail has experienced a dramatic shift from traditional brick and mortar stores to online stores. The opportunity to reach more markets and to sale 24/7 without high rent and staff costs, is the main driver the for the growth of ecommerce. To fulfill the growing demand for online stores, many companies have developed ecommerce solutions as-a-service. Most of those platforms are reasonable priced and easy to use. However, they lack the ability to adapt to unique business requirements. This project aims to design, develop, and implement a client-server web application to extend the functionalities of the Remu Apparel's online store on the popular ecommerce as-a-service platform, Shopify. The application will add production control functionalities that are not currently fulfilled by Shopify. The app will simplify the production process of the Company while avoiding information transfer human errors. Several tool including the Use Case Diagram and the Data Flow Diagram, will be used to design the application. The application will be written in Python using the Django framework of rapid development. The application will be deployed on the cloud using Heroku to simplify the deployment process. The app will be connected with a testing online store own by Remu Apparel and in the future the Company will have to evaluate a connection with its live online store.

Key words: ecommerce, Shopify, client-server application, production control, Django, Heroku, PostgreSQL

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

1.1 Descripción del problema

Actualmente existen muchas soluciones SaaS de eCommerce en el mercado. Entre las cuales se destacan plataformas cómo Shopify, Volusion y BigCommerce. Las principales ventajas de usar opciones SaaS de eCommerce versus alternativas open source o desarrollar el código desde cero incluyen tanto el costo y el tiempo inicial para montar la tienda en línea, cómo el mantenimiento del eCommerce (Oxatis, n.d.). Las plataformas SaaS no requieren mucha inversión inicial, simplemente un valor mensual fijo y costos iniciales bajos para la adquisición de dominio y templates. Los templates permiten personalizar la tienda en línea muy rápidamente ya que los dueños de la tienda no necesitan escribir código, simplemente cargar fotos, textos y diseñar menús y páginas a través de la plataforma del proveedor del eCommerce. Adicionalmente, el proveedor del servicio se encarga del mantenimiento y seguridad del software.

Por estas razones, las plataformas SaaS de eCommerce son muy accesibles y populares en la actualidad. Sin embargo, la mayoría de las soluciones de SaaS de eCommerce no rinden buenos resultados en un área: la adaptación al modelo de negocio y requerimientos únicos de cada empresa.

Remu Apparel, una empresa ecuatoriana de ropa sustentable usa Shopify para su eCommerce desde hace más de dos años. En el transcurso de ese tiempo ha detectado que Shopify no satisface una necesidad clave de su negocio: la coordinación de la producción. Shopify funciona asumiendo que el eCommerce maneja inventario, lo cual no corresponde al modelo de producción bajo pedido que lleva Remu. Cuando alguien compra un producto en el eCommerce de Remu, la orden debe ser enviada a los distintos talleres de la empresa para

comenzar a fabricar los productos. Dada la falta de funcionalidad de Shopify para realizar esa parte de su proceso, la compañía ha estructurado un mecanismo manual que incluye diversas hojas de Excel y mensajes por WhatsApp para coordinar la producción de los pedidos que se reciben. Este mecanismo, al ser manual, es vulnerable a errores de transmisión en muchas partes. Muchas veces la orden del cliente llega alterada al taller y no se detecta el error hasta el momento de enviar el producto al cliente, lo cual representa pérdidas de tiempo, recursos y dinero para la empresa.

1.2 Objetivo general

El objetivo principal de este trabajo de titulación es diseñar, desarrollar e implementar una aplicación web para el control de la producción de la empresa Remu Apparel. Una aplicación que automatice el proceso actual implementado en la compañía, con el fin reducir la complejidad del proceso y evitar errores humanos al momento de transmitir la orden del cliente al taller de producción.

1.3 Objetivos específicos

- Diseño e implementación de la base de datos.
- Conexión con Shopify para sincronizar las órdenes.
- Diseño e implementación de un sistema de autenticación con dos perfiles de usuario: coordinador y costurera.
- Construcción de un panel que permita visualizar las nuevas órdenes de pedido y las órdenes asignadas a cada costurera.

- Diseño e implementación de un mecanismo para mantener comunicación entre las costureras y los coordinadores
- Diseño e implementación de un mecanismo para monitorear el estado de cada orden

2. DESARROLLO DE LA APLICACIÓN

2.1 Justificación del proyecto

Para entender a profundidad el objetivo y funcionamiento de la aplicación web desarrollada es importante comprender el modelo de negocio de Remu Apparel. Remu es una empresa ecuatoriana, que fabrica ropa sustentable, producida éticamente en pequeños talleres en zonas rurales del Ecuador. La empresa se encarga tanto de la producción en Ecuador, a través de su red de costureras independientes, como de la comercialización en línea de dichos productos a nivel internacional. La plataforma de eCommerce que utiliza la empresa se llama Shopify. Esta plataforma permite contar con una tienda en línea, segura y profesional, sin preocuparse por el mantenimiento o la seguridad de esta.

Sin embargo, la plataforma Shopify está diseñada principalmente para empresas que manejan inventario de producto terminado. Remu, al ser una empresa con objetivos de cuidado ambiental fuertes, busca evitar al máximo el desperdicio de recursos, lo cual le ha llevado a adoptar un modelo de producción bajo pedido, sin necesidad de llevar inventario de producto terminado. Este modelo de producción entra en conflicto con la manera que opera Shopify. Por este motivo Remu busca desarrollar una aplicación web para el control de producción que se ajuste a sus necesidades específicas.

Al trabajar con talleres y costureras independientes, Remu necesita una aplicación que permita a cada uno de ellos acceder al sistema y visualizar los productos que deben producir.

2.2 Requisitos funcionales de la aplicación

Se definieron los siguientes requisitos funcionales para la aplicación:

- Contar con un sistema de autenticación con dos perfiles de usuario: coordinador y costurera.
- Extraer las nuevas órdenes de la página de Shopify de la empresa.
- El coordinador debe visualizar cada producto de cada orden por separado en el panel de nuevas órdenes.
- El coordinador debe asignar para producción un producto específico de una orden a una costurera.
- Las costureras deben visualizar las órdenes que les fueron asignadas para producción.
- El coordinador debe visualizar las órdenes que han sido asignadas a cada costurera.
- Disponer de un sistema de comunicación entre las costureras y los coordinadores.
- Contar con un sistema para monitorear los distintos estados en el ciclo de producción.

2.3 Ciclo de producción de Remu

En el ciclo de producción de Remu, el producto puede tener los siguientes estados: ‘nuevo,’ ‘asignado’, ‘cortando’, ‘armando’, ‘terminado’, ‘entregado’, ‘error’, y ‘corrigiendo’. Cada estado representa una fase distinta del proceso, por lo cual es vital llevar un control del estado.

Estado	Descripción
Nuevo	Cuando un cliente compra un producto en la tienda en línea. Los productos registrados en la orden todavía no son asignados a una costurera para producción. Este es un estado obligatorio por el cual todos los productos pasan.
Asignado	Cuando el producto fue asignado por el coordinador a una costurera especifica para producción. Este es un estado obligatorio por el cual todos los productos pasan.
Cortando	Cuando la costurera comienza a trabajar en el producto y procede a cortar las piezas de tela requeridas. Este es un estado obligatorio por el cual todos los productos pasan.
Armando	Una vez que la mayoría de las piezas fueron cortadas y la costurera comienza a armar y coser las piezas. Este es un estado obligatorio por el cual todos los productos pasan.
Terminado	Este estado indica que el producto está totalmente terminado y se encuentra en el taller de la costurera. Este es un estado obligatorio por el cual todos los productos pasan.
Entregado	Este estado señala que el producto fue entregado por la costurera y se encuentra en las bodegas de la empresa. Este estado es muy importante ya que indica que el producto está listo para despachar al cliente final. Este es un estado obligatorio por el cual todos los productos pasan.
Error	Este estado indica que el producto presenta algún defecto. No todos los productos pasan por este estado.

Corrigiendo	Este estado indica que se está corrigiendo un error previamente detectado en el producto. No todos los productos pasan por este estado.

Tabla #1. Estados de producción

2.4 Términos importantes

Orden: documento que registra los productos adquiridos por un cliente en una única transacción de compra. Una orden tiene un número que la identifica y puede incluir uno o más productos. Para consistencia con la nomenclatura de Shopify, los productos dentro de la orden son denominados *line items*. Por ende, una orden puede tener uno o más *line items* y cada *line item* tiene asociado un SKU.

Line item: Un producto dentro de una orden.

SKU: Número/código de referencia (en ingles *stock keeping unit*). Es un código asignado a un producto para poder identificarlo. Por ejemplo, la empresa Remu vende un modelo de abrigo de lana llamado Tambo, pero lo vende en distintos colores y tallas. Cada variante del abrigo Tambo tiene un SKU único para poder identificarle.

2.5 Diagrama de Casos de Uso

Una vez comprendido el flujo de producción y los requisitos funcionales de la aplicación se procedió a elaborar el diagrama de casos de uso de la aplicación con el objetivo de delimitar el alcance de la aplicación.

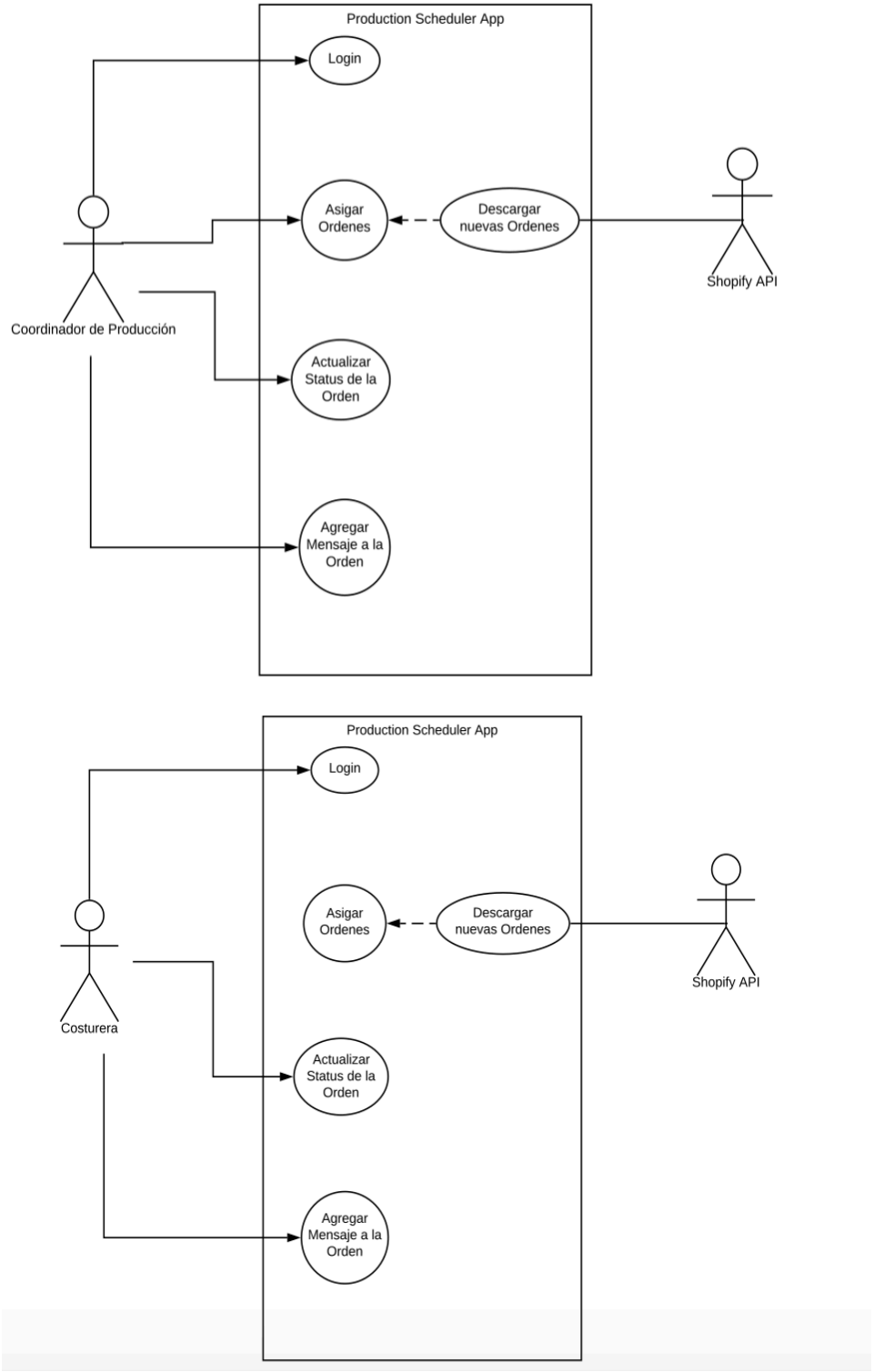


Figura #1. Diagrama de casos de uso

2.6 Diagrama de flujo de datos

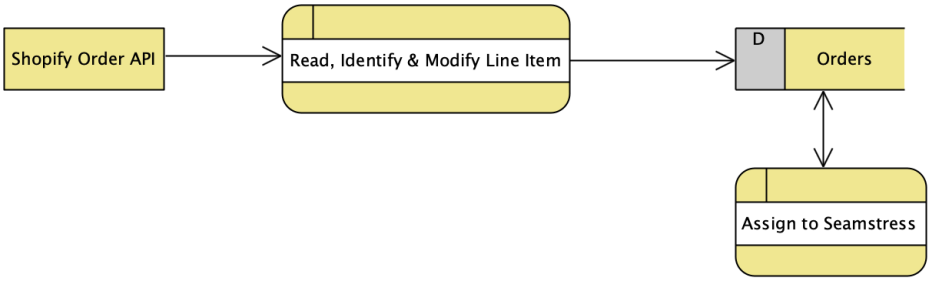


Figura #2. Diagrama de flujo de datos

La primera parte del flujo de datos de la aplicación consiste en modificar datos provenientes de la tienda en línea con los siguientes pasos secuenciales:

- La aplicación se conecta, a través de un web API, con la tienda de la empresa en Shopify y obtiene toda la información de las órdenes que aún no han sido atendidas hasta ese momento en formato JSON (JavaScript Object Notation). Se presenta a continuación la trama JSON generada por el web API.

```
,,
- line_items: [
   - {
        id: 4667850129546,
        variant_id: 32615115554954,
        title: "Loma Jacket - Dark",
        quantity: 1,
        sku: "LJD-WLRW",
        variant_title: "L / Raw Wool",
        vendor: "REMU APPAREL",
        fulfillment_service: "manual",
        product_id: 4637591011466,
        requires_shipping: true,
        taxable: true,
        gift_card: false,
        name: "Loma Jacket - Dark - L / Raw Wool",
        variant_inventory_management: "shopify",
        properties: [ ],
        product_exists: true,
        fulfillable_quantity: 1,
        grams: 0,
        price: "155.00",
        total_discount: "0.00",
        fulfillment_status: null,
      - price_set: {
          - shop_money: {
              amount: "155.00",
              currency_code: "USD"
            },
          - presentment_money: {
              amount: "155.00",
              currency_code: "USD"
            }
        },
      - total_discount_set: {
          - shop_money: {
              amount: "0.00",
              currency_code: "USD"
            },
```


Figura #3. Ejemplo trama JSON web API

- La aplicación procesa esa información e identifica cuales de esas órdenes son nuevas y aún no están registradas en la base de datos, procediendo a registrar todos los *line items* en la base de datos, identificándoles con el código SKU.

La segunda parte del flujo de datos es generada por los usuarios:

- El coordinador asigna un *line item* a una costurera, quien procede a actualizar el estado del *line item* hasta marcarlo como terminado.
- Una vez que la costurera marca un *line item* con el estado ‘terminado’, el coordinador procede a marcarlo con el estado ‘entregado’ si no hay ninguna observación y se almacenan el *line item* y sus hitos para fines históricos.

Adicionalmente, la aplicación cuenta con un sistema de log por cada *line item,* donde se registra automáticamente todas las acciones que algún usuario realiza sobre ese *line item*. Tanto el coordinador, como la costurera, pueden acceder a este log, visualizar los cambios y agregar mensajes manualmente para comunicarse entre ellos.

2.7 Módulos

Para cumplir con los requisitos funcionales la aplicación incluye tres módulos: el módulo de usuarios, el módulo de órdenes y el módulo de log.

El módulo usuarios se encarga de la creación de usuarios, la segmentación de usuario por grupos, la autenticación de usuarios y la recuperación de contraseñas. Este módulo cumple con las funcionalidades más comunes de la mayoría de los sistemas para manejo de usuarios. Dentro de este módulo se implementa los dos distintos niveles de acceso que maneja la aplicación: coordinador y costurera. Dependiendo a que grupo pertenece un usuario, coordinador o costurera, se le concede los permisos dentro de la aplicación y se determina que contenido recibe del servidor.

El módulo de órdenes es el módulo principal ya que contiene la mayoría de la lógica de la aplicación. Este módulo permite la extracción de los datos de Shopify, maneja los estados de los *line items,* permite asignar los *line items* a las costureras y registra el histórico de los *line items*.

El módulo de log es encargado llevar un registro de todos los eventos de cada *line item*. Los eventos incluyen la fecha de creación, fecha de asignación, cambios de estado, etc. Este registro se crea automáticamente cuando se ejecuta alguna acción que afecta a un *line item*. El módulo también permite, tanto al coordinador como a la costurera asignada a ese *line item,*

agregar mensajes manualmente en el log. Por lo tanto, este módulo funciona tanto como registro, como medio de comunicación entre el coordinador y la costurera.

2.8 Elección de tecnología

Existen dos opciones más apropiadas para el desarrollo de esta aplicación: web y móvil. En base a que la aplicación no requiere utilizar los sensores y capacidades internas de los dispositivos móviles, se optó por desarrollar una aplicación web cliente-servidor. La principal ventaja de una aplicación web cliente-servidor en este contexto es la compatibilidad con una gran variedad de dispositivos (Stanley, n.d.). A una aplicación web, a diferencia de una aplicación móvil, se puede acceder a través de cualquier dispositivo que soporte un navegador web como Chrome o Safari.

Inicialmente se contempló la opción de desarrollar el backend de la aplicación utilizando el framework Laravel de PHP, pero luego se decidió emplear el framework Django de Python. Las principales razones por las cuales se escogió Django sobre Laravel son la antigüedad del framework y las capacidades de desarrollo rápido (Django Project, n.d.). En la actualidad Django es un framework más moderno y popular por lo cual tiene una comunidad más activa que sirve de ayuda para desarrolladores, para resolver inquietudes y cuando surgen problemas. Adicionalmente, Django se caracteriza por promover el desarrollo de software rápido (Django Project, n.d.). Comenzar a escribir código utilizando Django es simple, se requiere escribir un par de comandos en la terminal y Django genera la estructura completa del proyecto, crea un servidor de desarrollo y una base de datos para desarrollo, SQLite3.

Otra ventaja de utilizar Django es el sistema de autenticación y manejo de usuarios. Django viene con un sistema de autenticación por default bastante robusto, el cual simplemente debe modificarse para adaptarse a cada proyecto. Adicionalmente, Django incluye un panel de

administración completamente listo para visualizar tablas en la base de datos. Con relación al manejo de la base de datos, Django presenta otra ventaja ya que utiliza un modelo en el cual las tablas se definen como clases en Python y se accede a ella a través de código Python sin necesidad de escribir comandos SQL (Django Project, n.d.). Luego Django migra eso a la base de datos que se desee utilizar y Django se encarga de generar los Queries correspondientes a los comandos en Python.

Para el frontend de la aplicación se utilizó Bootstrap 4, una librería desarrollada inicialmente en Twitter para obtener consistencia en los desarrollos internos. Bootstrap es una librería open source para desarrollo web que utiliza HTML, CSS y JS. La librería se enfoca en hacer aplicaciones y páginas web *responsive*, es decir, que se adaptan a la pantalla del usuario (Bootstrap, n.d.). Bootstrap utiliza una combinación de código CSS y otras librerías de JavaScript como jQuery y Popper.

Como se mencionó anteriormente, durante el desarrollo de la aplicación se empleo la base de datos sqlite3 por su facilidad de instalación. Pero esta base de datos es básicamente un archivo local que genera Django cuando realizamos la primera migración de las tablas. Para el ambiente de producción se trabajó con una base de datos más robusta y segura: PostgreSQL. La elección de PostgreSQL para la base de datos de producción fue tomada en base a dos factores: las ventajas intrínsecas de la misma y la compatibilidad absoluta con Heroku.

PostgreSQL es una base de datos relacional open source que utiliza el lenguaje SQL para garantizar integridad y seguridad de los datos (PostgreSQL, n.d.). La segunda razón por la que se eligió este sistema es debido a su compatibilidad con Heroku. Heroku es una plataforma en la nube basada en contenedores que permite desplegar y administrar una aplicación web de manera muy sencilla (Heroku, n.d.). Heroku se basa en la premisa de tornar extremadamente simple el mantenimiento y despliegue de aplicaciones para los desarrolladores. Es una plataforma que permite a los desarrolladores de aplicaciones olvidarse

por completo de administrar los servidores y asignar recursos, pero manteniendo una aplicación en un ambiente de producción seguro y escalable. Heroku permite instalar y configurar con pocos clicks una base de datos PostgreSQL.

Adicionalmente, para mayor seguridad Heroku cambia periódicamente las credenciales de acceso a la base de datos. Por estos motivos se escogió usar Heroku para desplegar la aplicación en la nube.

2.9 Diseño base de datos

La aplicación cuenta con una base de datos de 6 tablas, adicionales a las tablas propias de Django. Todas las tablas son visibles desde el panel de administración que viene por default instalado en Django. Para el diagrama entidad relación de la aplicación ver Anexo A.

3. CONCLUSIONES

3.1 Conclusiones

Los objetivos planteados para el proyecto se cumplieron a cabalidad, pues la aplicación satisface plenamente la necesidad especifica para la cual fue diseñada. La aplicación simplifica el proceso de control producción de los productos de la empresa y evita errores manuales de transmisión de especificaciones al momento de la asignación a las costureras de los productos que deben ser producidos. Se presenta a continuación una tabla en la que se evidencia el cumplimiento de los requerimientos establecidos en el proyecto.

Requerimiento funcional	Módulo que satisface el requerimiento
Contar con un sistema de autenticación con dos perfiles de usuario: coordinador y costurera.	Módulo de usuario.
Extraer las nuevas órdenes de la página de Shopify de la empresa.	Módulo de órdenes.
El coordinador debe asignar para producción un producto específico de una orden a una costurera.	Módulo de órdenes.
Las costureras deben visualizar las órdenes que les fueron asignadas para producción.	Módulo de órdenes.
El coordinador debe visualizar las órdenes que han sido asignadas a cada costurera.	Módulo de órdenes.
Disponer de un sistema de comunicación entre las costureras y los coordinadores.	Módulo de log.
Contar con un sistema para monitorear los distintos estados en el ciclo de producción.	Módulo de órdenes.

Tabla #2. Cumplimiento de requerimientos funcionales

Se puede concluir que la tecnología seleccionada para el desarrollo de la aplicación fue adecuada. Django permitió un desarrollo rápido, debido a que es un framework que genera código por default lo cual facilitó la tarea de desarrollo de la presente aplicación. Adicionalmente, Django es un framework seguro, pues implementa por default distintos mecanismos de seguridad, como por ejemplo los tokens CSRF (Cross-Site Request Forgery) y el almacenamiento de contraseñas encriptadas de los usuarios en la base de datos.

Desplegar en la nube la aplicación utilizando Heroku fue una decisión acertada, pues simplificó el proceso de despliegue y evitó tener que realizar la tarea de aprovisionamiento de servidores. Al hacer el despliegue en la nube con Heroku la aplicación se vuelve fácil de escalar ya que se puede instanciar más contenedores y Heroku se encarga de balancear el tráfico hacia la aplicación.

El desarrollo web es extremadamente importante en la época actual, ya que las empresas necesitan herramientas que les permitan operar de forma remota, compartiendo recursos y

centralizando la información para la toma de decisiones. Más aun en la crisis que atraviesa actualmente el mundo. Dadas las restricciones de movilidad y medidas de aislamiento para contener el COVID-19 el trabajo se debe hacer desde las casas y para eso se requieren las herramientas, que pueden ser aplicaciones web cliente-servidor, que permitan a las personas realizar sus tareas laborales desde el hogar de una manera eficaz y plena.

Finalmente, en el plano personal y profesional, el proyecto me permitió explorar a profundidad como funciona Django, obtener bases del funcionamiento de los frameworks, y herramientas de desarrollo web, tanto para el frontend como el backend. El aprendizaje y aplicación de estas tecnologías me proyectará en el futuro en este importante campo del conocimiento de la Ingeniería en Sistemas.

3.2 Trabajo futuro

Actualmente la aplicación, está conectada con una tienda de prueba de Remu en Shopify. Para implementar esta aplicación en la tienda en vivo de Remu en Shopify se debería generar nuevas credenciales para la conexión con el web API de la tienda y registrarlas en las variables de ambiente de Heroku.

A pesar de que se cumplió el objetivo planteado para el proyecto, para una versión futura se podrían considerar algunos cambios que mejoren la experiencia del usuario. La versión actual se enfoca fundamentalmente en aspectos funcionales, sin embargo, la parte gráfica y visual de la aplicación podría mejorar en una versión futura.

Otro aspecto que se podría mejorar en una versión futura son las notificaciones. La versión actual solo notifica al usuario de eventos y modificaciones si este se encuentra dentro de la aplicación. Sería interesante usar notificaciones por correo para que el usuario pueda recibir alertas en cualquier momento.

REFERENCIAS BIBLIOGRÁFICAS

Bootstrap. (n.d.). *About*. Obtenido el 3 de mayo 2020 de https://getbootstrap.com/docs/4.4/about/overview/

Django Project. (n.d.) *Meet Django*. Obtenido el 3 de mayo 2020 de https://www.djangoproject.com

Gunicorn. (n.d.). *Running Gunicorn*. Obtenido el 6 de mayo de 2020 de https://docs.gunicorn.org/en/latest/run.html

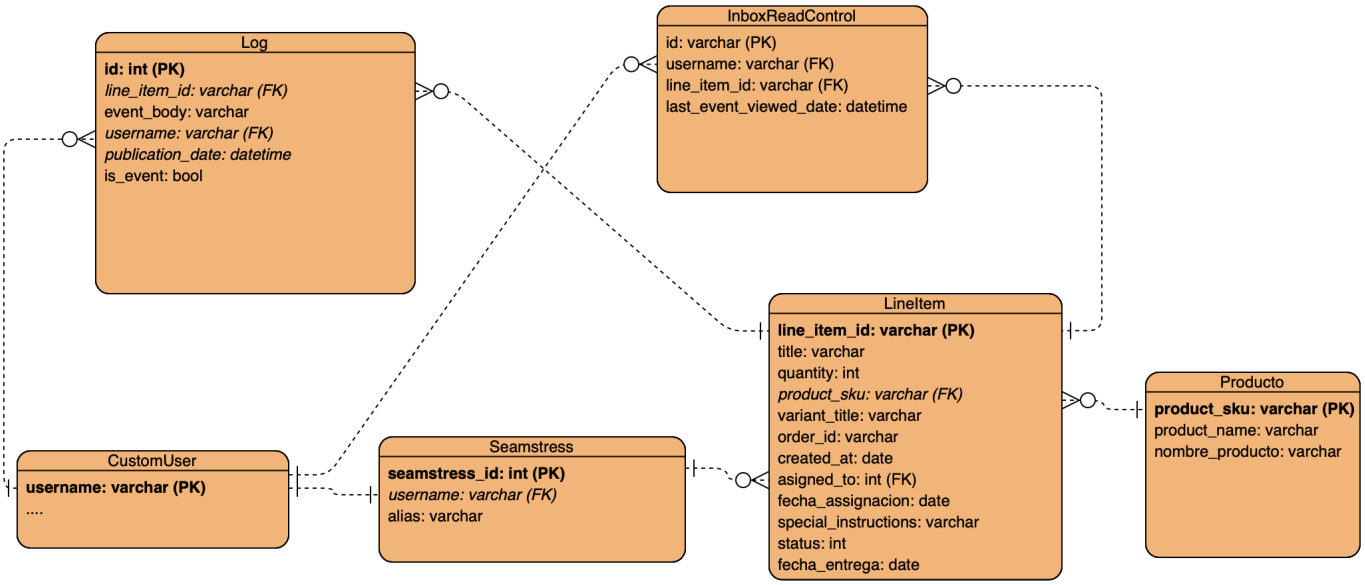
Heroku. (n.d.). *What is Heroku*. Obtenido el 3 de mayo de 2020 de https://www.heroku.com/about

PostgreSQL. (n.d.). *About.* Obtenido el 3 de mayo 2020 de https://www.postgresql.org/about/

Oxatis. (n.d.). *Solucion ecommerce SaaS*. Obtenido el 6 de mayo de 2020 de https://www.oxatis.es/solucion-ecommerce-saas.htm

Stanley, Paul. (n.d.). *Advantages of Web Applications*. Obtenido el 6 de mayo de 2020 de https://www.pssuk.com/AdvantagesWebApplications.aspx

ANEXO A: DIAGRAMA ENTIDAD RELACIÓN



ANEXO B: DETALLES DESPLIEGUE A HEROKU

Para poder hacer el despliegue de una aplicación a Heroku es necesario que el código se encuentre en Git ya que Heroku carga los archivos directamente de un repositorio de Git. Es por eso por lo que todo el código de la aplicación se encuentra en el siguiente repositorio: https://github.com/felipegabela/jupiter.git. Para hacer más fácil el deployment a Heroku se utilizó Heroku CLI o *Command Line Interface*, una librería para poder crear una aplicación en Heroku y poder cargar el código del repositorio de Git a Heroku directamente desde la terminal o línea de comandos.

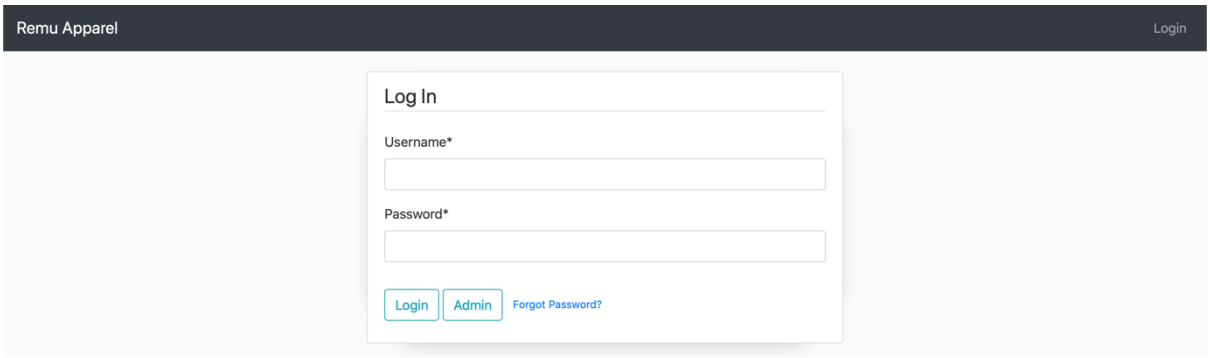
Se utilizó una versión gratuita de Heroku con 1 "Dyno" para producción. Los "Dynos" son los componentes principales de Heroku. Básicamente los "Dynos" son contenedores virtuales Linux independientes, con su propia capacidad de procesamiento, memoria, etc. Estos contenedores son quienes ejecutan la aplicación. El plan gratuito incluye 1000 horas de Dynos con 512 MB de RAM. Al exceder esta cantidad de horas hay que cambiarse a un plan pagado para continuar utilizando Heroku. Es importante mencionar que este tipo de Dyno se duerme después de 30 minutos de inactividad.

El Dyno que utiliza la aplicación corre un proceso web con Gunicorn, un servidor WSGI ligero de poco consumo de recursos para Python (Gunicorn, n.d.) Gunicorn es bastante versátil, rápido y se integra muy bien a Django.

Heroku utiliza el concepto de "addons" para incluir funcionalidades. La aplicación tiene instalados dos "addons". El primero, una base de datos PostgreSQL que es administrada y mantenida totalmente por el equipo Heroku. El segundo, Sendgrid para enviar los correos de recuperación de contraseña.

ANEXO C: MANUAL DE USUARIO

1. Entrada al sistema: para acceder a la aplicación usuario deberá contar con credenciales de acceso (usuario y contraseña). En caso de no tener debe solicitar la creación del usuario al administrador del sistema.
2. Portal de acceso:
    a. Para acceder a la aplicación se utilizará el siguiente enlace: https://jupiter-production-scheduler.herokuapp.com
    b. Inicio de sesión: ingresar las credenciales de acceso y dar click en “Login” para ingresar a la aplicación.



    c. Recuperar la contraseña: para recuperar la contraseña dar click en “Forgot Password?” El sistema desplegará una pantalla donde podrá ingresar el correo electrónico asociado a su cuenta y recibirá un correo con un link para cambiar de clave.
    d. Acceder al panel de administración: para ingresar al panel de administración dar click en “Admin”, luego ingresar las credenciales y dar click en “Login”. Recuerde que solo el administrado de la aplicación puede acceder a este panel.

3. Asignar *line item* a una costurera:

   a. Primero seleccionar la costurera a quien desea asignar y hacer click en "Asignar". Esto asigna el line item a la costurera pero todavía no cambia el estado de "Nuevo" a "Asignado", por lo cual la costurera todavía no puede ver ese *line item*.

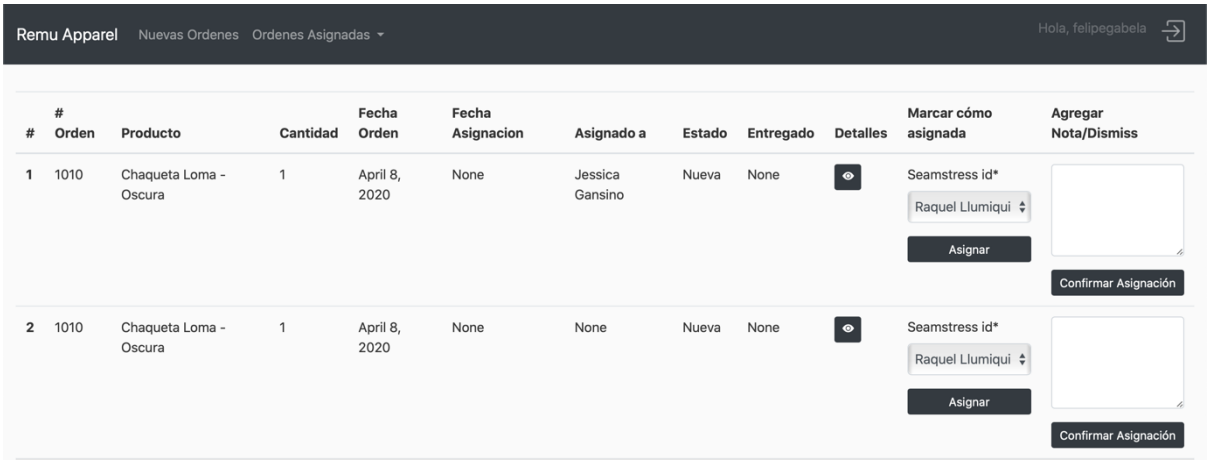
   b. Para confirmar la asignación y cambiar el estado de "Nuevo" a "Asignado" dar click en "Confirmar Asignación".

   c. Antes de hace click en "Confirmar Asignación" se puede agregar una nota con indicaciones especiales para la costurera. Al hacer click la nota será almacenada y será visible tanto para los coordinadores como para la costurera a la cual el *line item* fue asignado.

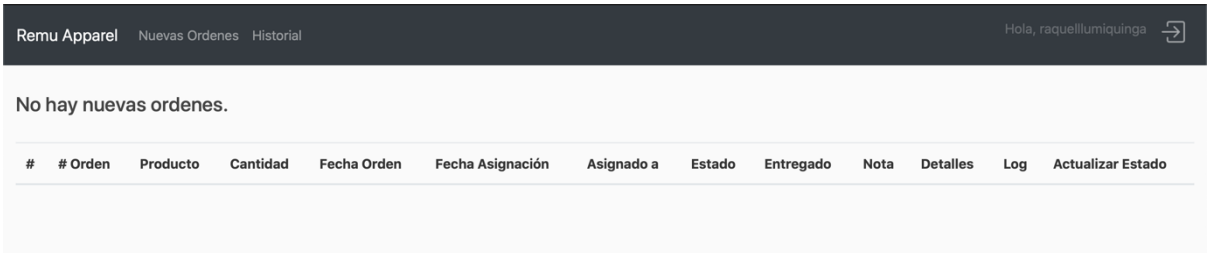
   Nota: Es necesario tener permiso de coordinador para poder asignar órdenes a las costureras.

4. Ver nuevas órdenes: por defecto cuando un usuario inicia sesión accede a la pantalla donde visualiza los nuevos line items. Sin embargo, los coordinadores visualizan todos los *line items* de las nuevas órdenes que aun no están asignadas a una costurera para producción mientras que las costureras visualizan los *line items* asignados a ellas en producción. También se puede acceder a esta vista haciendo click en el menú principal donde indica "Nuevas Ordenes".

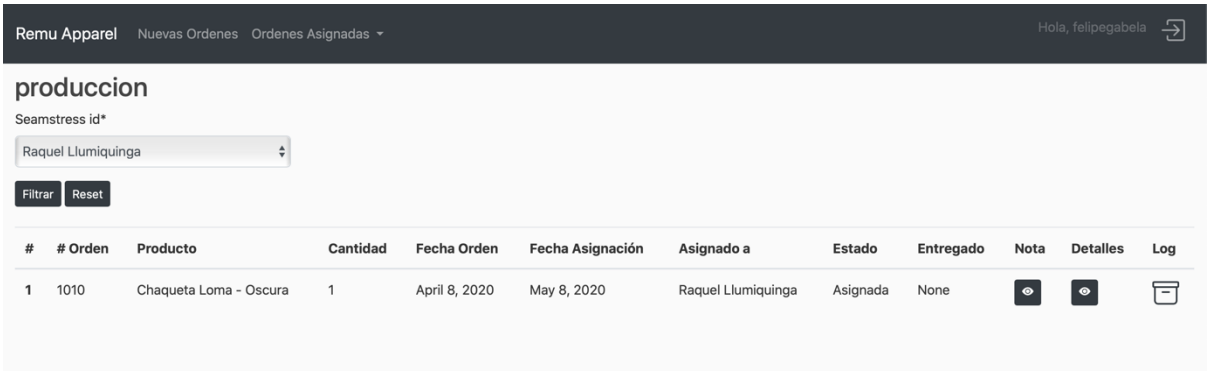
Vista del coordinador: donde visualiza todos los nuevos *line items* sin asignar a producción



Vista de la costurera: en este caso no hay nuevos *line items* asignados a ella para producción

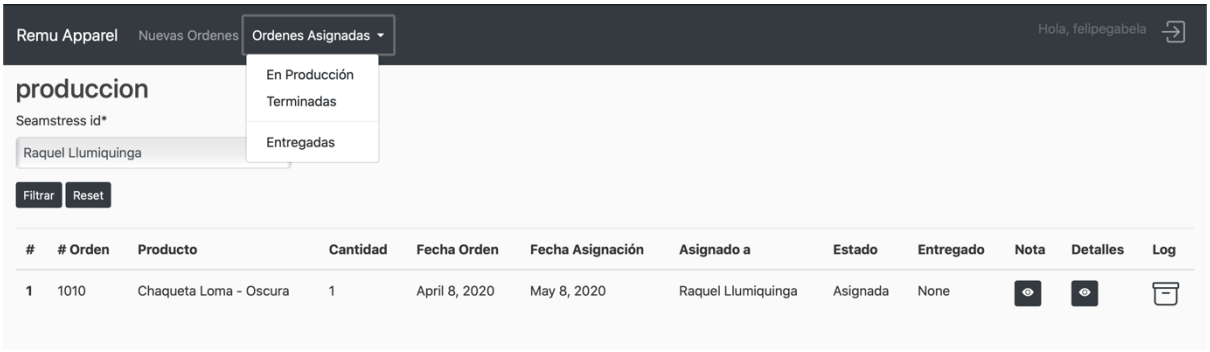


5. Ordenes Asignadas: solo los coordinadores tienen acceso a esta vista. En esta vista están todos los *line items* asignados a todas las costureras.



   a. Para acceder a esta vista dar click en “Ordenes Asignadas”

   b. Se desplegara el siguiente menú donde deberá elegir si desea visualizar los *line items* asignados actualmente en producción, los *line items* asignados con estado “Terminado”, o los *line items* que ya fueron entregados

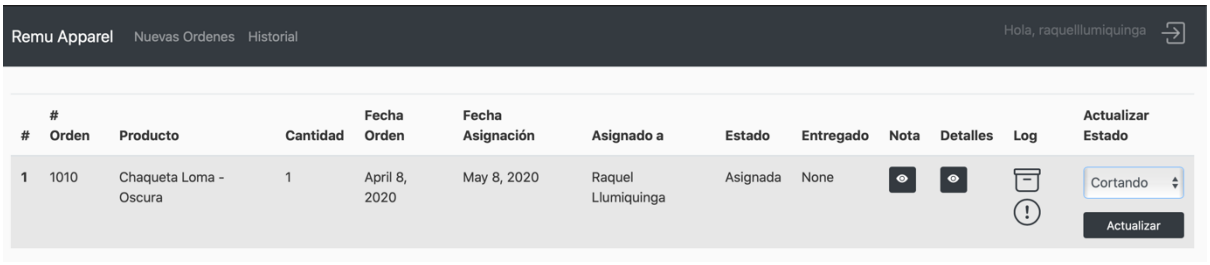


c. En la vista de ordenes asignadas se puede filtrar por costurera especifica o ver todos. Para filtrar elegir la costurera en el selector y presionar “Filtar”. Para eliminar el filtro presionar “Reset”.

6. Cambiar de estado de un line item:

a. Coordinador: los coordinadores solo pueden cambiar de estado un *line item* cuando la costurera lo marca como “Terminado”. Para hacerlo deben acceder a la vista de “Ordenes Asignadas” mencionada arriba, en la opción “Terminadas” del submenú. Una vez allí deben seleccionar el estado deseado en el selector de cambio de estado y presionar “Actualizar”.

b. Costureras: las costureras en cambio solo pueden cambiar el estado de un *line item* asignado a ellas en la vista de “Nuevas Ordenes”. Una vez allí seleccionar el estado deseado en el selector y presionar “Actualizar”.



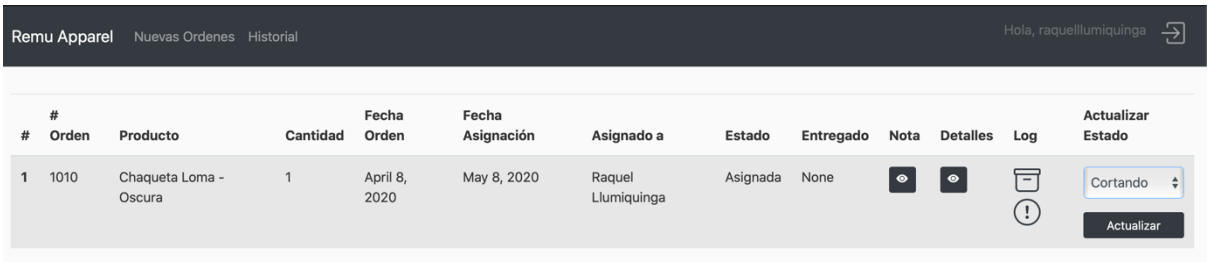
7. Acceder al log del *line item*:

   a. Coordinadores: los coordiandores pueden acceder a cualquier log de cualquier line item en la vista “Ordenes Asignadas”. Una vez allí presionar el icono de mensajes debajo de la columna “Log”.

   b. Costureras: las costureras solo pueden acceder a los logs de *line items* asignados a ellas. Lo pueden hacer tanto desde la vista “Nuevas Ordenes”, como de la vista “Historial”. Una vez allí presionar el icono de mensajes debajo de la columna “Log”.

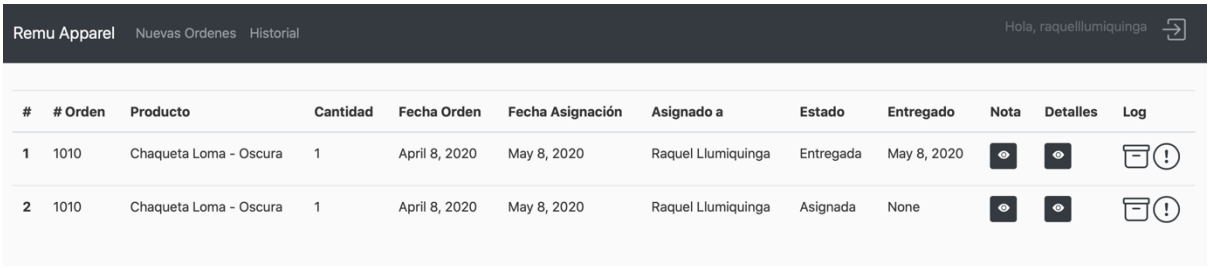
   c. Dentro del log podrán visualizar en el contenido, pero también añadir mensajes manualmente.

d. El símbolo de alerta a lado del icono de mensajes indica que existen mensajes o eventos nuevo en *line item* que el usuario todavía no los ha visto.



8. Historial:

a. Costureras: las costureras pueden acceder a la vista “Historial” para ver un histórico de todas las órdenes asignadas a ella. El histórico es muy importante ya que una vez que el line item es marcado “Entregado” por el coordinador ya no se muestra en la vista “Nuevas Ordenes” de la costurera.



9. Panel de administración: solo el administrador de la aplicación puede acceder a este panel.

En este panel se puede:

a. Crear nuevos usuarios: para crear un nuevo usuario hacer click en “users”. Luego presionar “Add User” e ingresar la información solicitada. Luego añadir a un grupo y presionar “Save”. Importante: un usuario no puede pertenecer a

dos grupos. Los nombres de usuario deben ser el nombre seguido por el apellido sin espacio, todo en minúsculas.

b. Para crear un usuario costurera se requiere un paso adicional. Luego de crear el usuario como indicado arriba regresar a la página principal del panel haciendo click en “Home” en la esquina superior izquierda. Luego hacer click en “Seamstress”. Presionar “Add Seamstress” y elegir el usuario de la costurera que acabamos de crear y asignarle un alias igual al nombre de usuario pero con mayúsculas y espacio entre el nombre y el apellido.

ANEXO D: CÓDIGO FUENTE

El código fuente tiene la estructura de carpetas detallada en este anexo. En este anexo se explicará brevemente que hace cada archivo del código fuente. Para ver el código fuente ingresar al siguiente repositorio de Github: https://github.com/felipegabela/jupiter.git.

Carpeta Raíz de la aplicación:

Nombre del archivo o carpeta (>)	Funcionalidad o descripción.
>jupiter	Carpeta de configuración de la aplicación.
>log	Módulo del log.
>production_scheduler	Módulo de órdenes.
>templates	Templates HTML para toda la aplicación.
>users	Módulo de usuarios.
.gitignore	Archivos que no queremos tener en GitHub.
Procfile	Archivo que indica que tipo de proceso correr en el contenedor de Heroku.
README.txt	Instrucciones para instalar la aplicación.
manage.py	Script de Django que permite realizar tareas como migraciones o creación de usuarios.
requirements.txt	Lista de dependencias de la aplicación.

A continuación, se mostrará la estructura de carpetas interna de cada carpeta en la carpeta raíz indicada arriba.

>jupiter:

__init__.py	Archivo vacío que indica que es una aplicación de Django.
asgi.py	Configuración ASGI creada por defecto por Django.
settings.py	Archivo principal de configuración de la aplicación.
urls.py	Archivo de mapeo de URL principal.
wsgi.py	Configuración WSGI creada por defecto por Django.

>log:

>migrations	Carpeta con las migraciones del modelo de datos del módulo.

>templates>blog	Carpeta que contiene los templates HTML del módulo.
init__.py	Archivo vacío que indica que es un módulo de Django.
admin.py	Archivo donde se registra que tablas del modelo de datos del módulo serán accesibles desde el panel de administración.
apps.py	Archivo de configuración del módulo.
db_manipulation.py	Archivo del módulo que interactúa con la base de datos.
forms.py	Archivo con los formularios del módulo.
models.py	Archivo con el modelo de datos del módulo.
tests.py	Archivo que almacena los *tests* del módulo.
urls.py	Archivo de mapeo de URL del módulo
views.py	Archivo con la lógica del módulo.

>production_scheduler

>migrations	Carpeta con las migraciones del modelo de datos del módulo.
>static>css	Carpeta que contiene el código CSS del módulo.
>templates>production_scheduler	Carpeta que contiene los templates HTML del módulo.
>templatetags	Carpeta que contiene funciones que añaden funcionalidades especificas en el módulo que pueden usarse en tanto en los templates como en las vistas (*views*).
admin.py	Archivo donde se registra que tablas del modelo de datos del módulo serán accesibles desde el panel de administración.
apps.py	Archivo de configuración del módulo.
init__.py	Archivo vacío que indica que es un módulo de Django.
db_manipulation.py	Archivo del módulo que interactúa con la base de datos.
forms.py	Archivo con los formularios del módulo.
historial_view.py	Archivo que contiene la lógica del historial de ordenes.
new_orders_view.py	Archivo que contiene la lógica del manejo de nuevas ordenes.
post_views.py	Archivo que contiene los métodos HTTP POST del módulo.
models.py	Archivo con el modelo de datos del módulo.
tests.py	Archivo que almacena los *tests* del módulo.
urls.py	Archivo de mapeo de URL del módulo

>templates

>registration	Carpeta que contiene los templates HTML para el sistema de autenticación.
base.html	Archivo con le código HTML básico que todos los templates extienden.
navbar.html	Archivo HTML de la barra de navegación de la aplicación.

>users

>migrations	Con las migraciones del modelo de datos del módulo.
>templatetags	Carpeta que contiene funciones que añaden funcionalidades especificas en el módulo que pueden usarse en tanto en los templates como en las vistas (*views*).
init__.py	Archivo vacío que indica que es un módulo de Django.
admin.py	Archivo donde se registra que tablas del modelo de datos del módulo serán accesibles desde el panel de administración.
apps.py	Archivo de configuración del módulo.
forms.py	Archivo con los formularios del módulo.
models.py	Archivo con el modelo de datos del módulo.
tests.py	Archivo que almacena los *tests* del módulo.
views.py	Archivo con la lógica del módulo.