"El saber de mis hijos hará mi grandeza"

Alumno: Juan Daniel Grijalva Soto

Expediente: 214201526

Tutor: René Navarro

Proyecto: "Módulos de Odoo"

Empresa: Prodigia Procesos Digitales Administrativos

Índice

Introducción	3
Descripción del área donde se realizaron las prácticas	4
Justificación del proyecto realizado	5
Objetivos del proyecto	6
Problemas planteados para resolverlos	7
Alcances y limitaciones en la solución de problemas	8
Fundamento teórico aplicado	9
Procedimientos empleados y actividades desarrolladas	11
Resultados obtenidos	22
Conclusiones y recomendaciones	32
Retroalimentación	33
Referencias bibliográficas y virtuales	35

Introducción

La carrera de Ingeniería en Sistemas de Información del Departamento de Ingeniería Industrial incluye en su plan de estudios un requisito especial: las prácticas profesionales. Estas tienen un valor de 20 créditos, que son equivalentes a 340 horas.

Para cumplir con tal requisito se investigó sobre empresas de desarrollo de software en Hermosillo. Se contactó a varias de estas y solo una respondió con buenas noticias: Prodigia. Prodigia es una empresa que desarrolla software administrativo y financiero cuyo propósito es apoyar a pequeñas empresas en el ámbito organizacional ofreciendo productos digitales que hagan de sus procesos internos una tarea sencilla. Se ubican en la Torre N2, primer piso, en el boulevard Antonio Quiroga #21 de la colonia Quinta Amalia. Actualmente son un *PAC* (Proveedor Autorizado para la certificación de CFDI) y ofrecen timbrado fiscal de facturas electrónicas a través de su producto estrella *Pade*. Además, a sus clientes les ofrecen una implementación personalizada de *Odoo*, una plataforma de manejo de recursos y herramientas administrativas.

El período de prácticas se llevó a cabo durante junio y agosto del año 2018, en el cual se desarrollaron módulos para ampliar las funcionalidades de *Odoo*. Uno de ellos constaba de la adaptación de un proyecto anteriormente desarrollado en Django/Python, el cual generaba rutas óptimas entre 2 o más ubicaciones utilizando la API de Google Maps, esto con el fin de brindar una estrategia de ahorro de recursos a los clientes de Prodigia.

Este documento es una memoria de tal trabajo, en el cual se detallarán los objetivos del proyecto, los fundamentos teóricos requeridos, alcances y limitaciones, procesos y actividades que se realizaron y finalmente están las conclusiones y recomendaciones que resultaron de esta experiencia.

Descripción del área donde se realizaron las prácticas

Prodigia, al igual que muchas empresas, consta de varios departamentos de trabajo para poder funcionar, entre ellos el área de soporte técnico, ventas, administración… pero yo estuve en el departamento principal: el área de desarrollo. Es un espacio de tamaño mediano en donde normalmente trabajan 17 personas, unas enfocadas en mantener los productos ya existentes de Prodigia, otras dedican su tiempo a los proyectos en desarrollo (como *Odoo)* y un par trabajan dando soporte técnico a los clientes. Todos tienen un escritorio asignado, con computadoras potentes y dos monitores. Se vive un ambiente divertido y relajado, pero con el trabajo siempre en mente.

Se utiliza la metodología SCRUM, y semanalmente realizan pequeñas juntas (llamadas *Sprints*) para hablar de los avances, problemas y experiencias que han tenido durante la semana. Además, se mantiene contacto directo con el departamento de ventas y la gerencia para obtener retroalimentación de los clientes, avances, enfoque de objetivos y actualizaciones de los productos.

Figura 1. Edificio donde se encuentra Prodigia.

Justificación del proyecto realizado

Se implementará un módulo dentro de la plataforma ERP de código abierto *Odoo* para apoyar a los usuarios a generar rutas óptimas entre sus clientes, ayudándoles a pasar menos tiempo en el tráfico y así ahorrar gasolina. Por otro lado, también se extenderá la funcionalidad del módulo de facturación para permitir la importación de archivos XML, las cuales son facturas de proveedores y así tener control de ellas en la plataforma.

La misión de Prodigia es simplificar los procesos administrativos y contables de las empresas. Siguiendo esa idea, siempre se buscar mejorar los productos de la empresa para alcanzar la máxima satisfacción de los clientes. El módulo de generación de rutas óptimas se planteó para apoyar a las empresas cuyas actividades requieren desplazarse de un lado a otro en su automóvil. También se añadirá la posibilidad de crear mapas con la ubicación de todos los clientes en forma de puntos para visualizar la distribución de estos dentro de la ciudad. De esta forma, los clientes de Prodigia que usen *Odoo* podrán administrar mejor su tiempo en el tráfico, planear rutas, conocer las zonas en donde dominan el mercado y ahorrar gasolina.

Por otro lado, los usuarios que administran facturas se han quejado de que no se pueden importar facturas en la sección de compras (también conocida como la de proveedores), cuando esa funcionalidad ya existe en la sección de ventas. Su queja principal era esa, “¿por qué aquí no puedo y allá sí?”. Al abordar estas quejas y darles una solución sencilla y eficaz, los clientes de Prodigia estarán más satisfechos y serán más propensos a recomendar las soluciones digitales de esta empresa.

Objetivos del proyecto

El objetivo general de ambos desarrollos es ampliar la cantidad de soluciones empresariales que Prodigia ofrece a sus clientes a través de la plataforma Odoo. Algunas de estas soluciones surgen de las opiniones y necesidades de los clientes de la empresa; otras, son propuestas propias que la empresa aceptó.

Entre los objetivos principales destacan:

- Ofrecer una nueva solución de navegación al generar rutas optimizadas entre dos o más puntos, con información relevante en tiempo real.
- Ayudar a las pequeñas y medianas empresas a ahorrar recursos valiosos como gasolina y tiempo, reduciendo gastos innecesarios y esfuerzo de sus empleados.
- Mejorar el control sobre los gastos de la empresa al permitir importar facturas de compras a Odoo.

Problemas planteados

El problema principal al desarrollar estos proyectos tiene que ver directamente con los procesos y la estructura de la plataforma Odoo. Debido a que es un ERP ya establecido y un producto grande el cual miles de empresas emplean, sus funciones, diseño, estilos, convenciones y demás ya están preestablecidos. Esto puede ocasionar problemas a la hora de querer implementar nuestro propio módulo, ya que las limitantes son muy tajantes y te obliga a hacer todo a su manera. Debido a que el módulo de rutas es un proyecto ya existente hecho con el framework *Django*, habrá que ver si no se pierde funcionalidad al adaptarlo e implementarlo dentro de *Odoo*.

En el caso de extender la funcionalidad del módulo de facturación, es posible que el código no sea lo suficientemente abierto para permitir agregar botones y más código a algo que ya funciona en su totalidad.

Al ser de código abierto, existe bastante documentación de Odoo y de cómo resolver este tipo de problemas. Sin embargo, al leer y explorar dicha documentación es evidente que la lengua nativa de la comunidad que ha colaborado en este proyecto no es el inglés. Hay muchos *typos*, errores de gramática, enunciados que no tienen sentido, etcétera. Además, han habido 11 versiones de Odoo hasta la fecha y tristemente hay muy poca información sobre la última versión, que es la que utiliza Prodigia. Es evidente fue imposible mantener una buena documentación del proyecto por más de 10 versiones, por ende gran parte de la documentación está escondida dentro del código fuente. Esto puede retrasar mucho las actividades de desarrollo.

Alcances y limitaciones

Se tienen alrededor de 2 meses (8 semanas y media) para desarrollar ambos proyectos. El enfoque estará en completar al 100% ambos proyectos para que funcionen en su totalidad, tal como fue solicitado.

Para el caso del módulo de generación de rutas óptimas, este verá su funcionalidad reducida a la ciudad de Hermosillo ya que allí residen todos los clientes de Prodigia que utilizan *Odoo*. Sin embargo, se dejará listo el código y la documentación para poder extender estas barreras hacia todo el país de México, en caso de que la empresa crezca. Esta limitante no existe en el módulo de facturación, ya que Prodigia tiene la posibilidad de timbrar facturas y validarlas ante el SAT sin importar el lugar de origen. De todos modos, el desarrollo de este proyecto no tiene nada que ver con lugares o barreras físicas.

Ambos proyectos se desarrollarán para una plataforma web, por lo que estar actualizado e informado sobre tecnologías web como JavaScript y HTML será obligatorio. También se utilizaran otras herramientas, como *PaaS* (*Platform as a Service*), terminal de Bash, Git, Python, entre otras… de tal forma que conocer con anticipación cómo funciona el flujo de trabajo de estas tecnologías será de gran ayuda.

Se requerirá mucha paciencia al adentrarse en la documentación de *Odoo* debido a los problemas dictados en la sección anterior. Conocer y participar en sitios como *Stack Overflow* o el foro oficial de *Odoo* puede apaciguar el camino.

Fundamento teórico aplicado

La tecnología principal utilizada fue Odoo, un ERP open source que funciona con el lenguaje de programación Python, y PostgreSQL como manejador de bases de datos. Odoo funciona a través de un ORM (Object-Relational Mapping) para poder trabajar con SQL a través del paradigma orientado a objetos. El ORM se controla con Python, y se utiliza para la programación de *backend* (código en servidor), pero para el *frontend* (interfaces, lo que ve el usuario) el ORM va muy de la mano con el sistema de *views* de Odoo, el cual funciona con archivos XML declarando campos y tags específicos, similar a HTML.

Las interfaces y diseño visual de Odoo ya están preestablecidos gracias a que es un framework, pero este también pueden ser editado a través de XML, HTML o CSS. Es necesario utilizar HTML y CSS para customizar ciertos aspectos visuales de los módulos. Con Python se programa la lógica de los módulos, se hacen llamadas a la base de datos a través del ORM, se hacen cálculos y llamadas a APIs externas, y los datos obtenidos son pasados al *frontend* en donde se ordenan y manipulan con código XML. Así es el workflow de Odoo visto de forma general.

Python es un lenguaje orientado a objetos. Durante el desarrollo se utilizaron diferentes tipos de clases para definir y guardar objetos en la base de datos. De igual manera diferentes tipos de estructuras de datos y métodos ofrecidos por Python fueron utilizados, como listas, diccionarios, tuplas, *list comprehensions*, métodos HTTP, manejo de timezones, entre otros. Este lenguaje de programación fue utilizado de la mano con una terminal de Bash, para correr comandos de ejecución, pruebas en base de datos, debugging, manejo de archivos y demás. El software utilizado para programar fue VS Code (editor de text), pgAdmin 4 (manejador de base de datos) y Git Bash (terminal).

A pesar de no utilizar SQL directamente debido al uso de un ORM, es útil conocer cómo funcionan las queries para resolver ciertos problemas. Se utilizan iltros, relaciones one-to-many o many-to-many, joins y agregaciones. Las funciones de agregación son muy útiles ya que se utilizan para resumir y totalizar datos de los clientes, y además, la posibilidad de usar campos de tipo arreglo/lista se agradece.

El uso de Git y GitHub es muy requerido durante el desarrollo de estos proyectos. De entrada, para poder instalar Odoo se necesita clonar el repositorio de GitHub a través de Git. Para el módulo de facturación existente de Prodigia fue necesaria la configuración de llaves SSH, ya que el repositorio estaba privado debido a datos sensibles en el código. Y, evidentemente, es útil y seguro usar un sistema de control de versiones para tener un control sobre los cambios que se hacen en el día a día. Obviamente el conocimiento básico de comandos de Git era indispensable: *add*, *commit* y *push*. Otros como *status*, *diff* o *checkout* también son utilizados. Saber cómo funcionan las ramas, qué hacer durante conflictos, los *forks* y otras utilidades son de gran ayuda para mejorar y resolver situaciones durante el desarrollo.

Además, Git era necesario para utilizar la plataforma odoo.sh, la cual es una *PaaS* (*Platform as a Service*) con una instancia de Odoo preinstalada, y sirve para desarrollar, ejecutar y probar módulos de Odoo en la nube. Esta plataforma es de muchísima utilidad ya que elimina la complejidad de preparar o configurar un servidor, y puedes asegurarte de que tu código está estandarizado y funcionará en cualquier servidor que tenga Odoo. Se requiere aprender el flujo de trabajo de esta plataforma y aprender a configurarla. En este tipo de servicios simplemente se crea un repositorio en GitHub y la plataforma hace deploy por tí cada vez que se *pushea* un cambio al repositorio. En caso de haber conflictos se tiene que usar una terminal remota o checar logs, por lo que la habilidad de debuggear con rapidez y utilizar comandos de bash o Linux son necesarios.

Procedimientos empleados y actividades desarrolladas

Durante la primer semana recibí un programa de capacitación, tanto en línea como presencial, dentro de la empresa. La capacitación en línea fue a través de un curso masivo (*MOOC,* por sus siglas en inglés) en la plataforma *Thinkific*, el cual me fue proporcionado por la empresa ya que tenía un costo. Este curso me sirvió para conocer Odoo, sus funciones, alcance, sintaxis y sobre todo, cómo crear un módulo personalizado e implementarlo en Odoo. El contenido del curso era fácil de seguir; estaba basado directamente en la documentación oficial de Odoo, y hacían uso de videos complementarios para explicar más a fondo algunos detalles. Sin embargo, al comenzar a desarrollar unas semanas después, me di cuenta que el curso (al igual que la documentación oficial) sólo tocaba partes superficiales de este framework y falló en proporcionar detalles específicos de muchas de las funciones internas de Odoo. Para solucionar esto tuve que recurrir a muchísimos threads en el foro de Odoo y Stack Overflow, lo cual ocasionó otro problema del que hablaré más adelante.

Además del curso en línea, la dirección de Prodigia se encargó de incorporarme en la empresa al conducir una serie de pláticas (como la de seguridad informática para enterarme de procesos y sanciones) para conocer los procesos internos de la empresa de tal forma que me sintiera cómodo durante mi estadía. El equipo de desarrollo también me ayudó a conocer los productos de Prodigia, cómo funcionan y especialmente, qué datos manejan dentro de Odoo. Una vez incorporado a la empresa, tuve una junta con los líderes del proyecto en donde me indicaron las actividade que estaría realizando, los requerimientos detallados de ambos proyectos y las necesidades que tenía la empresa. Al final de la junta mostré mi proyecto de generación de rutas óptimas, el cual utiliza la API de Google Maps, y me dijeron que eso era exactamente lo que necesitaban, ya que se ajustaba a las

necesidades de algunos de sus clientes. Entonces me preguntaron si podía implementar el mismo proyecto en su plataforma Odoo, a lo cual respondí que sí, ya que mi proyecto y Odoo usan el mismo lenguaje de programación, *Python.* Obviamente habría que hacer varios ajustes, pero la idea principal y el código serían casi el mismo. La primer diferencia que se me vino a la mente es que Odoo no utiliza HTML directamente, sino XML. Pero hablaré de eso más adelante.

Continué con mi investigación acerca de Odoo y cómo desarrollar módulos personalizados, y un par de días después, Rafael Carrillo (mi asesor) me dió una plática introductoria a odoo.sh, la plataforma para despegar Odoo en la nube (*Platform as a Service)*. Primero, me enseñó un módulo que él hizo, el cual agrega la facturación de Prodigia al módulo default de facturación. Este módulo que Rafael desarrolló se encontraba en un repositorio privado de GitHub, ya que contiene información sensible. Rafael me mostró me mostró el módulo que él hizo debido a que yo agregaría más funcionalidad a este (importar facturas de proveedor). Como el repositorio era privado, me tuvo que agregar como colaborador para poder descargar el proyecto y poder agregar cosas en un futuro. Además, Mailyn Cabrera (otra asesora) me generó una llave SSH para poder hacer *commits* y *pushear* cambios al repositorio de forma segura desde mi computadora. Después, Rafael me explicó cómo trabajar con este tipo de plataformas, usando Git y repositorios de GItHub. Me pareció muy claro debido a que yo ya he trabajado con plataformas similares (Heroku), por lo tanto simplemente creé mi cuenta en odoo.sh, hice un repositorio nuevo en GitHub, le añadí como submódulo (a través de `git submodule`) el repositorio de facturación de prodigia que hizo Rafael, y al final creé un proyecto en odoo.sh, indicando la URL de mi repositorio. Cada vez que yo subo cambios a mi repositorio, odoo.sh se encarga de actualizar los cambios y hacer tests.

Ya terminado mi *setup*, lo que quedaba era ponerme a trabajar en los proyectos. Comencé a estudiar el módulo de facturación que desarrolló Rafael, tratando de aplicar y recordar lo que había aprendido con el curso y la

documentación de Odoo. Me sentí intimidado por tanto código que no conocía. Funciones y statements que no había visto en el curso ni en la documentación oficial, y tristemente nada estaba comentado. Rápidamente me di cuenta de que no tenía la experiencia suficiente para comenzar con el proyecto de facturación, así que decidí primero “aprender” Odoo ensuciándome las manos con el proyecto de rutas, ya que la lógica principal y el manejo de la API de Google Maps ya la había hecho, y sólo tenía que adaptarlo a Odoo. Pensé que así, sobre la marcha, aprendería más fácil y rápido.

Así pues, decargué el repositorio open source de Odoo con `git clone` para correrlo en mi computadora. Instalé las librerías de las que depende Odoo, creé una base de datos vacía en PostgreSQL y corrí el proyecto localmente para asegurarme de que todo estaba bien. Después, para agregar mi propio módulo y desarrollarlo, utilicé el comando `odoo-bin scaffold`, el cual simplemente crea una subcarpeta dentro de la carpeta `addons` con todos los archivos necesarios para un nuevo módulo. Entonces, usando la lógica MVC (Modelo Vista Controlador) para el desarrollo web, definí unas rutas de prueba (URLs) en el archivo `controllers.py` (el cual guarda las URLs que están disponibles en el módulo, y la lógica que se debe seguir cuando se visiten dichas direcciones). También creé unas vistas XML para mostrar los datos de prueba. Así, cuando yo visitaba la dirección `127.0.0.1:8000/rutas` la aplicación me mostraba una serie de datos (los datos constaban de una lista de rutas extraídas de Google Maps e información sobre ellas). Me sentía satisfecho pues al igual que Django (el framework que anteriormente usé para el proyecto de rutas), Odoo funcionaba de la misma manera (o al menos eso parecía), y sólo tenía que traducir las vistas HTML a XML para que el módulo de rutas corriera en Odoo. Sin embargo, me di cuenta de que, aunque Odoo puede funcionar de la misma manera que Django a través de un MVC, realmente debe funcionar como un sistema operativo: la pantalla principal es un menú con la lista de módulos (llamados aplicaciones) que están instaladas, parecido a Android cuando presionamos el botón de aplicaciones. Todos los módulos tienen el mismo diseño, mismos componentes, menús, tamaños y tipografía.

Esto significaba que lo que realmente tenía que hacer era integrar el módulo de rutas de forma *nativa*, al igual que los otros módulos. Deshice lo que había hecho y eché un vistazo a la configuración de otros módulos para saber cómo hacer que mi módulo esté en el mismo ambiente que los demás.
Resulta que sólo tenía que cambiar unas lineas en el `manifest`, el archivo de configuración del módulo, y agregar *elementos* de menú (etiquetas como `<record />`) a las vistas XML (cabe mencionar que las etiquetas XML sirven para decirle a Odoo qué HTML debe renderizar; es una manera más funcional y programática para generar vistas). El módulo aparecía en la sección de instalar aplicaciones, sólo tenía que instalarlo y así aparecería en el menú principal de apps. Ya instalado, podía acceder al módulo dando click en su ícono, tal como si fuera un acceso directo de Android o Windows.

Sin embargo, por el momento sólo había menús vacíos, así que siguiendo la guía oficial de cómo crear tu módulo, primero creé los modelos de la base de datos, es decir, las tablas y los campos que necesita el módulo para poder guardar datos dentro de Odoo. Estos campos se crean en el archivo `models.py`, y se crean al estilo orientado a objetos ya que Odoo, al igual que muchos frameworks, maneja la base de datos a través de un *ORM* (Object-Relational Mapping). Así, las tablas y sus campos pueden ser tratados como objetos y las queries como métodos para facilitar el uso de los datos. Después de haber definido los campos (entre ellos, la fecha, ids de los clientes que se visitarán, origen, distancia y duración de la ruta), habría que definir algunos métodos básicos para el ORM, como `create`, el cual crea un nuevo registro al ser llamado. En este modelo también agregué todos los métodos que utilicé en mi generador de rutas, para poder generar una ruta a partir de los datos que sean guardados en Odoo. Entre estos métodos estaban: calcular ruta, el cual tomaba una lista de clientes y punto de origen, obtenía la dirección de estos puntos y llama a la API de Google Maps en el endpoint `directions` con el parámetro `optimize=true` (para obtener la ruta más rápida entre los puntos). La API regresaba una serie de datos como la URL (que se usó para mostrar la ruta en un iframe HTML) y la distancia y tiempo entre cada tramo de

la ruta. Otro de los métodos era el de calcular el tiempo y distancia total de la ruta, el cual itera sobre cada tramo de la ruta, sumando los valores y regresa el total. Además, tuve que crear un archivo `requirements.txt` en donde se definen las librerías *externas* de las cuales depende el módulo. En este caso, la librería de Google Maps. Este archivo le dice al intérprete de Python qué librerías instalar al momento de despegar la aplicación en odoo.sh para que la aplicación funcione correctamente en la nube.

Ya que la lógica *backend* estaba lista, recurrí a hacer las vistas y formularios del módulo de rutas, como el formulario de creación de rutas, el de modificar rutas, ver lista de rutas generadas, entre otras. Para el formulario de creación y la lista de rutas generadas fue bastante sencillo: simplemente declarar qué campos son visibles y/o editables dentro de etiquetas XML `<form />` para el formulario y `<tree />` para ver las rutas ya guardadas. El formulario de creación consta de 2 campos: seleccionar clientes (máximo 23, debido a limitantes por parte de Google) y punto de origen. Los clientes disponibles se obtienen del módulo Contactos, y los puntos de origen del módulo de Inventarios (ahí están guardados los almacenes y direcciones de la empresa).
Una vez creada una ruta, los campos distancia, duración se hacen visibles, al igual que el mapa interactivo que muestra la ruta generada.
En la lista de rutas guardadas se muestran los siguientes datos: número de ruta, fecha, duración, distancia y cantidad de clientes. Por default, Odoo agrega el botón de editar y borrar automáticamente por lo que no hay que preocuparse por eso. Sin embargo, tuve que agregar un poco de lógica en el modelo Python para editar una ruta. Por ejemplo al editar una ruta, si se elimina un cliente de la lista de clientes seleccionados, la ruta se recalcula y el mapa que muestra la ruta tiene que actualizarse, eliminando el punto en donde se encontraba el cliente eliminado. De igual manera, si se cambia el punto de origen de una ruta, la ruta se tiene que recalcular y el mapa se actualiza. Esos cambios suceden en *tiempo real*, y es posible hacerlo gracias al decorador `onchange` de Odoo, el cual indica que, si el valor de un campo cambia, se deben de recalcular otros campos que dependan de él.

Estas acciones (ver, crear y modificar) se tenían que acceder desde algún lado, por lo que proseguí a crear los menus *dropdown* en la barra de navegación de Odoo. También agregué una barra de búsqueda, la cual sirve para filtrar rutas por su fecha, distancia, duración, número y/o por los clientes que aparecen en la ruta.

Otra de las funciones que debía hacer este módulo era crear un mapa estático (no interactivo, sólo una imagen) con la ubicación de todos los clientes registrados mostrada como puntos. Los puntos debían ser un pin rojo y debían mostrarse en la dirección de cada uno de los clientes. Esto es útil ya que muestra la distribución de los clientes en la ciudad, y puede proporcionar una ayuda estratégica a la empresa al saber en dónde se concentra su base de usuarios. Naturalmente, se agregó una opción en el menú de la barra de navegación para acceder a esta función. El formulario para crear este mapa solo requería que el usuario seleccionará a los clientes que el mapa debía mostrar y listo. Una vez creado el mapa este aparecía como imagen en el formulario. El elemento `<img />` de HTML toma como `src` (fuente) una URL generada por la API de Google, la cual tiene como límite 8192 caracteres. Si la longitud de la URL supera eso, se le indica al usuario que elimine clientes de la selección para que la URL se acorte. Si los datos del mapa se modifican (el usuario quita o agrega clientes), la imagen del mapa se recalcula y actualiza en tiempo real, al igual que el formulario de generar rutas.

Las funciones detalladas hasta ahora son las funciones principales que me fueron requeridas. Por lo tanto presenté el módulo ante mis supervisores, para hacerles saber de mis avances y obtener un poco de retroalimentación. Sentí un sentimiento de alivio al escuchar que todo estaba perfecto, pero no me libré de la retroalimentación. Mailyn me indicó que estaría bien agregar una función que muestre las veces que se han visitado a cada cliente en una gráfica de barras. Dicho en otras palabras: cuántas veces se ha seleccionado a cada cliente a la hora de hacer una ruta. Esta funcionalidad puede ser muy útil para el usuario, supongamos el siguiente ejemplo: la empresa que usa el módulo de

rutas es una empresa de soporte técnico de computadoras y arregla las computadoras de diferentes cibercafés por toda la ciudad. A través de la gráfica de barras, la empresa puede ver cuáles cibercafés visitan más y, por ende, en cuál cibercafé se descomponen más las computadoras y así ofrecerles ayuda especial u ofertas.

Después de ver la utilidad que sería tal herramienta en el módulo de rutas, acepté la oferta de Mailyn y me puse a trabajar en esta. Primeramente, se debía agregar un campo contador a la tabla de clientes, el cual aumente en 1 (uno) cada vez que se genere una ruta. El modelo de base de datos al que pertenecen los clientes es nativo de Odoo, y se llama `res.partner`. El nombre `res` es el nombre del módulo *resources* de Odoo, y `partner` es el nombre del modelo de la base de datos. Así podemos entender que dicho modelo está dentro de la carpeta/módulo `res`; así es como funciona un poco el nombramiento de los modelos en Odoo. Entonces, al ser un modelo nativo de Odoo, yo no podía (más bien, no debía) modificarlo directamente. Lo que debía hacer era crear un modelo personalizado dentro del módulo de rutas, en el archivo ya existente `models/models.py`. El modelo a ser creado tenía algo especial: el campo `_inherit`, el cual estaba puesto a `'res.partner'`. De esta forma, el modelo creado "extiende" al modelo `res.partner`, y podemos agregar los campos que queramos (en este caso, un simple contador) y se verán reflejados en el modelo padre. Entonces procedí a agregar dos campos nuevos: `visits`, de tipo *Integer* y `route_ids`, el cual es un campo *Many to Many*. La razón de este último campo, es que `visits` debe tener el parámetro `compute='función X'`, el cual indica que el valor del campo será calculado automáticamente por `función X`. En este caso, la función que calcula las visitas se llama `_get_number_of_visits` y dicha función depende del campo mencionado, `route_ids`, el cual es una lista de las rutas en la relación cliente-ruta. Es en esta relación donde se registran las rutas en las que aparece cada cliente, y la función `_get_number_of_visits` simplemente regresa la longitud de esta lista haciendo uso de la función `len()` de Python. Otro detalle es que a la función para obtener el número de visitas le agregué el decorador

`@api.depends('route_ids')` para que cada vez que el campo `route_ids` cambie (por ejemplo, si se elimina una ruta o un cliente), la cantidad de visitas se actualice automáticamente y no haya incoherencias en los datos. Dicho todo esto, ya estaba lista la lógica backend para calcular la gráfica de barras. Ahora faltaban las vistas XML y menús para acceder a esta funcionalidad. Procedí a agregar los elementos XML correspondientes para el menú (`<menuitem />`), acciones que se activarán al hacer click en dicho menú (el cual se ve así: `<record model="ir.actions.act_window" />`) y finalmente las etiquetas de para hacer gráficas, las cuales vienen incluidas en Odoo. Para hacer una gráfica de barras basta con `<graph type="bar" />` y agregar los campos que quieres mostrar, en este caso `name` para mostrar el nombre de los clientes en X y `visits` para la cantidad de visitas en Y. Con esto ya terminaba esta funcionalidad.

Hasta ahorita todo lo había probado en una instancia local con una instancia de desarrollo descargada del GitHub de Odoo. Por eso subí los cambios a una instancia de odoo.sh para asegurarme que lo que había desarrollado estaba estandarizado y podría funcionar en cualquier entorno. Subí los cambios a mi repositorio de GitHub y odoo.sh se encargó de hacer deploy y tests. Por fortuna todo salió bien y funcionaba a la perfección, obviamente después de instalar y configurar el módulo de rutas. Decidí mostrar mis avances una vez más a mis superiores. Mailyn se alegró al ver funcionar perfectamente la gráfica de visitas. Rafael me indicó que todo iba bien, y me preguntó si había alguna forma de enviar o compartir rutas a un trabajador registrado en Odoo, por medio de e-mails. Esta funcionalidad existe en mi proyecto de Django, y no era algo tan complicado entonces con gusto comencé a adaptarlo para Odoo. Primeramente, creé las vistas XML y menú correspondiente. Para enviar/compartir una ruta, se tendría que hacer click en el menú “Enviar rutas”, el cual abre una ventana emergente, conocida en Odoo como *Wizard*. Un wizard es simplemente una serie de pasos para hacer algo, el cual progresa cuando el usuario clickea en “Siguiente” o “Terminar”. Entonces, para enviar una ruta, se inicializa el wizard, después el usuario hace click en las rutas que

quiere enviar, después se selecciona al trabajador a partir de la lista de empleados registrados (la cual se obtiene del módulo *Employee Directory* de Odoo), y finalmente se hace click en “Enviar” para terminar el proceso y enviar las rutas por correo electrónico a los empleados seleccionados. En el backend, se popula una plantilla de email, la cual le indica al empleado que alguien le compartió unas rutas, seguido de una lista de URLs de rutas de Google Maps. Estas URLs son públicas (sin parámetros de la API) y tienen la siguiente forma: `maps.google.com/<origen>/<dirección 1>/<dirección 2>/<dirección 3>/.../<dirección N>/`. Por cada ruta seleccionada en el wizard se obtienen las direcciones de los clientes correspondientes y así se genera la URL *pública* de la ruta, la cual puede ser accedida desde un navegador web o de la aplicación de Google Maps. Esta funcionalidad puede ser muy útil. Por ejemplo, si una ruta la recibe un chofer, simplemente abre la URL desde su celular y la aplicación de Google Maps le indicará direcciones para cubrir toda la ruta. Por último, para poder enviar emails con Odoo hay que usar la función `send()`, agregar `mail` como dependencia en el archivo `__manifest__.py` y configurar el usuario, contraseña y servidor que se usará (esto se hace manualmente en la interfaz de configuración). Subí de nuevo los cambios a odoo.sh y el módulo seguía funcionando perfectamente.

Llegó Julio y ya era hora de hacer la presentación final del módulo ante el director general (Gustavo), mis supervisores y el equipo implementador de Odoo. Me hicieron varias preguntas sobre el módulo, hubo retroalimentación y propuestas nuevas, pero al final a todos les gustó y me felicitaron. Me preguntaron si estaba documentado para que la empresa pueda actualizar y mejorar el módulo cuando yo ya no esté, respondí que casi cada línea está comentada, con referencias a links externos de documentaciones y tutoriales. Una de las propuestas que obtuve para mejorar el proyecto era relacionar el módulo de rutas con el de inventarios para obtener de ahí las direcciones de almacén, ya que las direcciones de origen de la ruta se tenían que crear dentro del módulo de rutas, pero esas direcciones se supone que ya existían dentro del módulo de inventarios, en la sección de almacenes.

Esa propuesta me pareció demasiado buena para dejarla pasar, además de que era algo bastante sencillo de hacer. Para hacerlo, simplemente tenía que editar el campo `origen` y hacerlo de tipo `Many2one`, indicando que la relación se haría con el módulo `stock`, específicamente su modelo `warehouse` (`stock.warehouse`). De esta forma, cuando se quiera generar una ruta nueva, las direcciones de origen que aparecerán serán los almacenes registrados en el módulo de inventario. Así fue como terminé este módulo, así que lo entregué a la empresa creando un repositorio privado en su cuenta de GitHub y subiendo los archivos con `git push`.

Para comenzar a desarrollar el otro proyecto (importación de facturas XML de proveedores) cloné el repositorio de Rafael Carrillo, quien hizo el módulo de facturación de Prodigia. En ese módulo, tendría que trabajar en el archivo `models.py`, además de agregar un par de vistas nuevas. Para el aspecto visual y de interfaz, agregue una opción “Importar XMLs” dentro del menú *dropdown* de facturas de proveedor. Al hacer click en esta opción, se ejecuta un *Wizard* el cual guía al usuario con una serie de pasos para importar facturas. La idea general es la siguiente: al ejecutarse el wizard, el usuario hace click en el botón “escoger archivo”, el cual deberá ser un ZIP que contenga archivos XML. Después se verifica que sea un ZIP válido y que cada archivo dentro de éste sea una factura válida.

Para importar archivos, se tiene que crear un nuevo modelo y definir un campo de tipo `Base64`. Aquí se guarda el archivo de forma binaria. Para trabajar con este archivo binario (el cual es un realmente es un ZIP) se usa la función `b64decode` del módulo `base64` de Python. Posteriormente se transforma a bytes con la función `BytesIO` del módulo `io`. Finalmente, los bytes se convierten a un archivo ZIP con la clase `ZipFile` del módulo `zipfile`, el cual es nativo de Python. Si el archivo subido no es un ZIP, salta la excepción `BadZipFile`, la cual sirve para decirle al usuario que escoja un archivo válido.

Una vez que tenemos el ZIP en memoria, se puede iterar sobre su contenido a través de la función `zipfile.infolist()`. Así se obtiene cada archivo XML dentro del ZIP. Cada factura se convierte a diccionario de Python (JSON) con el módulo `xmltodict` por simple comodidad, ya que así se puede acceder a los atributos y elementos del XML más fácilmente. Ya que se tiene el XML en forma de diccionario es más fácil obtener los datos de la factura que se desean guardar en Odoo. Entonces se hace *scraping* sobre cada factura para obtener los datos requeridos (como subtotal, impuestos, productos, precio unitario, descuentos, fecha, proveedor). Si el proveedor de la factura no está registrado en Odoo, este se crea automáticamente con su nombre, RFC y estado fiscal. Lo mismo para los productos: se crean si no están registrados ya que se necesita un ID válido para agregarlos a la factura en Odoo. Una vez recolectados todos los datos necesarios, se utiliza un web service/API de Prodigia para verificar la validez de la factura ante el SAT. A grandes rasgos, esta API toma el certificado de la factura (el cual es un *hash* encriptado de 256 caracteres) y su llave, para posteriormente verificar que la llave es la que se utilizó para generar dicho certificado. Si es válido, se crea una factura en estado de borrador, utilizando la función `create` de Odoo sobre el modelo `account.invoice`. De lo contrario, se arroja un error y se le avisa al usuario que no es posible importar la factura.

El desarrollo que se hizo sobre este módulo fue bastante sencillo, el problema fue saber cómo implementarlo dentro del módulo existente de facturación ya que había muy poca documentación y en los foros la gente no se da a entender. Pero se logró y al mostrarselo a mis supervisores quedaron encantados. Al final, hice un commit y push al repositorio de Rafael en la rama `importar-proveedores` para que no hubiera conflictos y, posteriormente, hice un *pull request* para que Rafael pudiera combinar mi rama con dicho repositorio en un futuro y que mi desarrollo esté oficialmente dentro del módulo de facturación de Prodigia.

Resultados obtenidos

El módulo de rutas y la herramienta para importar archivos XML como facturas fueron implementadas en una instancia de odoo.sh. Ambos proyectos fueron entregados a la empresa a través de un repositorio privado de GitHub. A continuación se mostrarán capturas de ambos proyectos funcionando en odoo.sh.

Primero, el tablero de aplicaciones de Odoo (figura 2). Desde aquí se accede a todos los módulos instalados. Se puede observar el módulo de rutas con el logo de Prodigia. También está el módulo de facturación, que es donde se importan los archivos XML a la sección de compras/proveedores. Por último, se pueden ver otros módulos que se usan como el directorio de contactos y empleados, los ajustes generales y el manejo de inventario y almacenes.

Figura 2. Tablero de aplicaciones de Odoo. Se muestra el acceso al módulo de rutas.

Al entrar al módulo de rutas, se muestra la interfaz principal (figura 3). Aquí están 3 diferentes opciones en la barra de navegación: rutas, mapas y reportes. En rutas está la opción de crear y buscar rutas, como aparece en la figura 3.

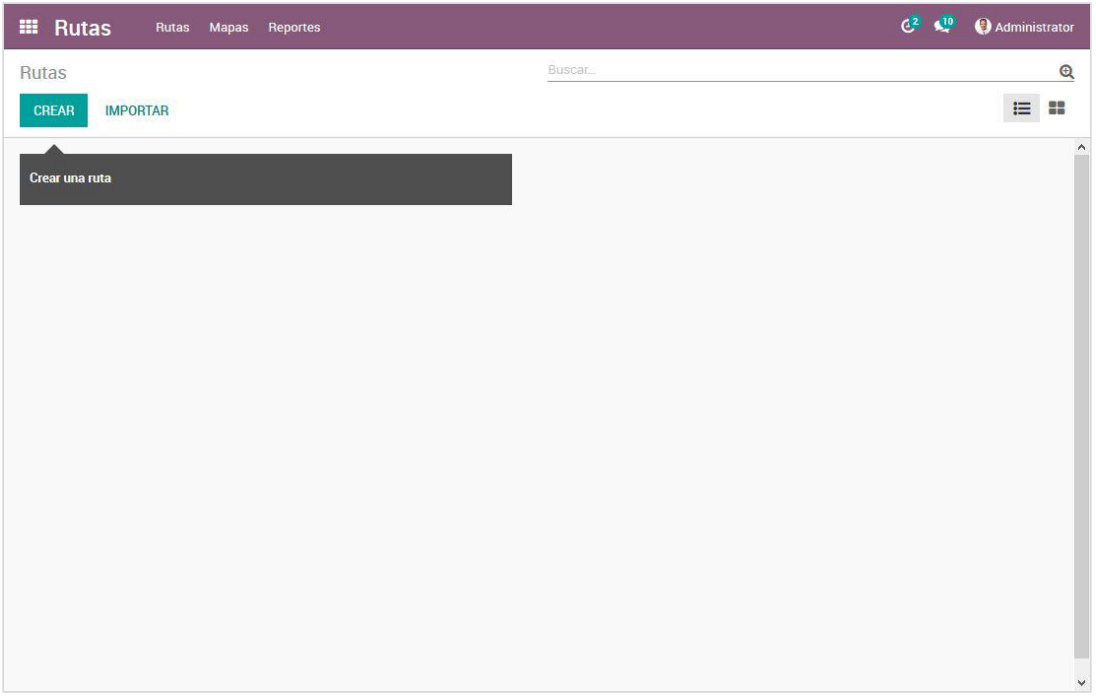

Figura 3. Interfaz general del módulo de generación de rutas óptimas.

En la figura 3.1 se muestra el formulario para crear una ruta. Se seleccionan los clientes (los cuales deben estar registrados en el módulo de contactos) y un punto de partida o almacén, cuya dirección se obtiene del módulo de inventario.

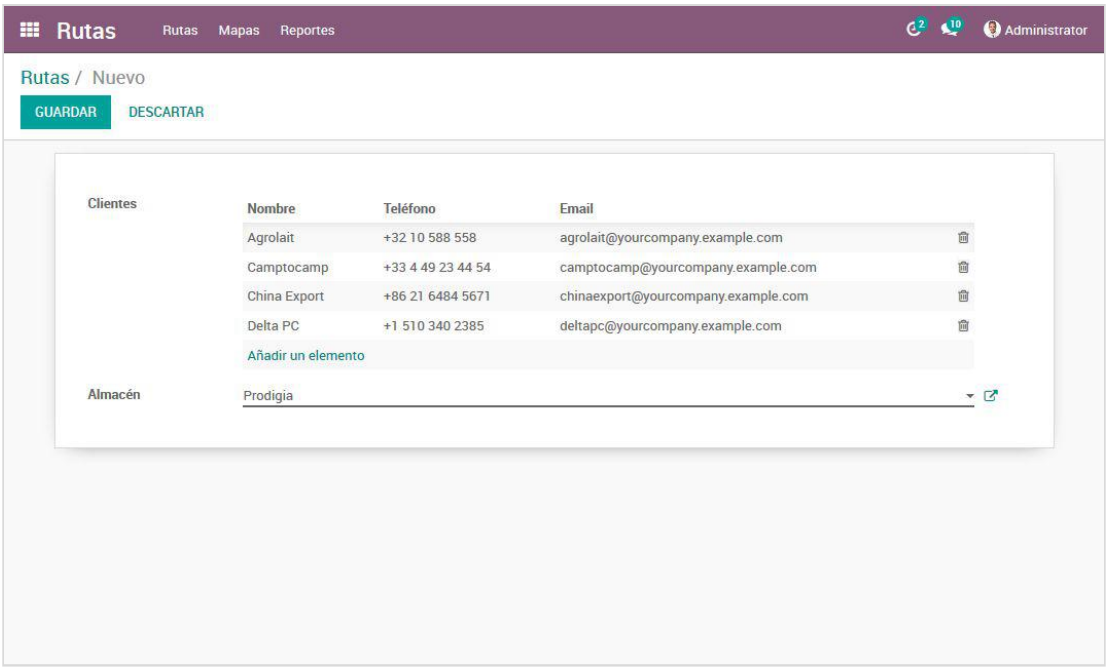

Figura 3.1. Creación de una ruta.

Una vez se hace click en guardar, se llama a la API de Google Maps para obtener una ruta óptima entre los clientes seleccionados. Se muestra el nombre de la ruta—el cual es generado automáticamente con un número autoincremental y la fecha de creación—la distancia total de la ruta y la duración de esta, además de un mapa interactivo que muestra el recorrido (figura 3.2).

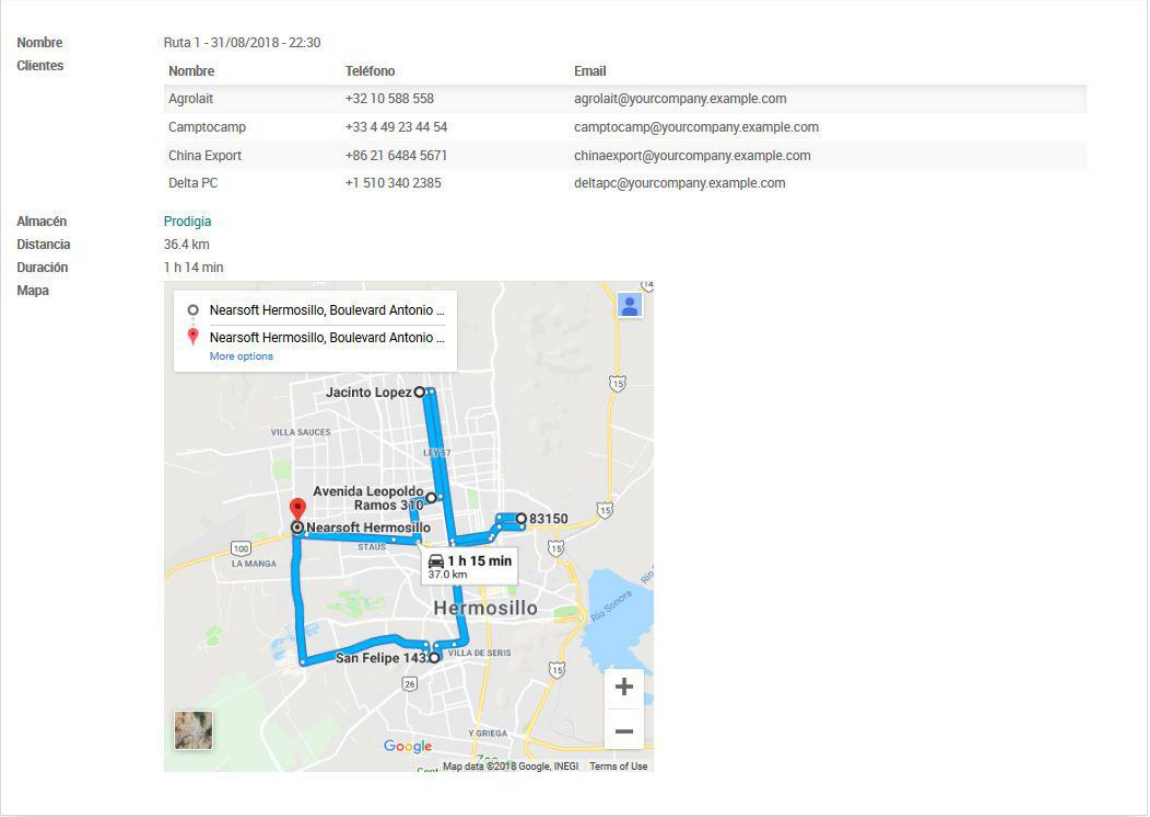

Figura 3.2. Una ruta generada.

Así, en la sección de rutas se muestra una lista de rutas guardadas, indicando el número de clientes seleccionados, la distancia y duración de estas. Al seleccionar una o más rutas a través de su respectivo *checkbox* se pueden borrar, archivar o exportar a archivo CSV o Excel. Además, el buscador permite filtrar por clientes, fecha, duración o distancia (figura 3.3).

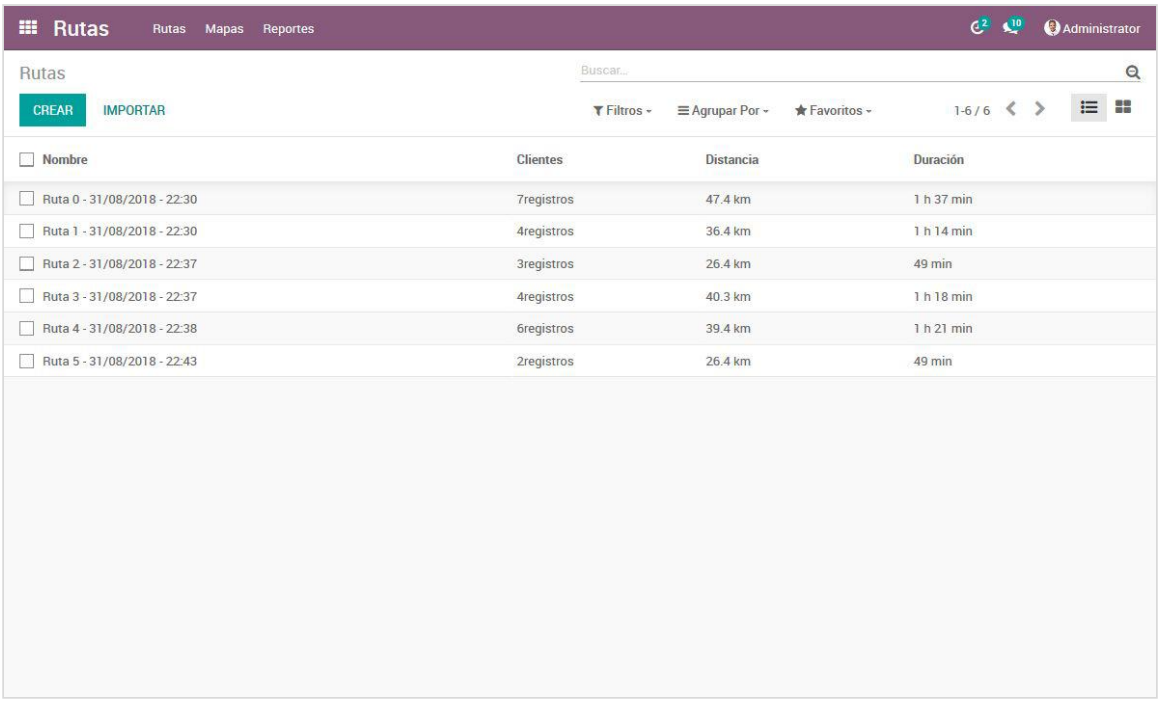

Figura 3.3. Lista de rutas guardadas.

Otra función de este módulo es la de generar mapas estáticos que muestre la ubicación de todos los clientes registrados. Para hacerlo, el formulario sólo necesita que seleccione a los clientes que se desean mostrar (figura 4).

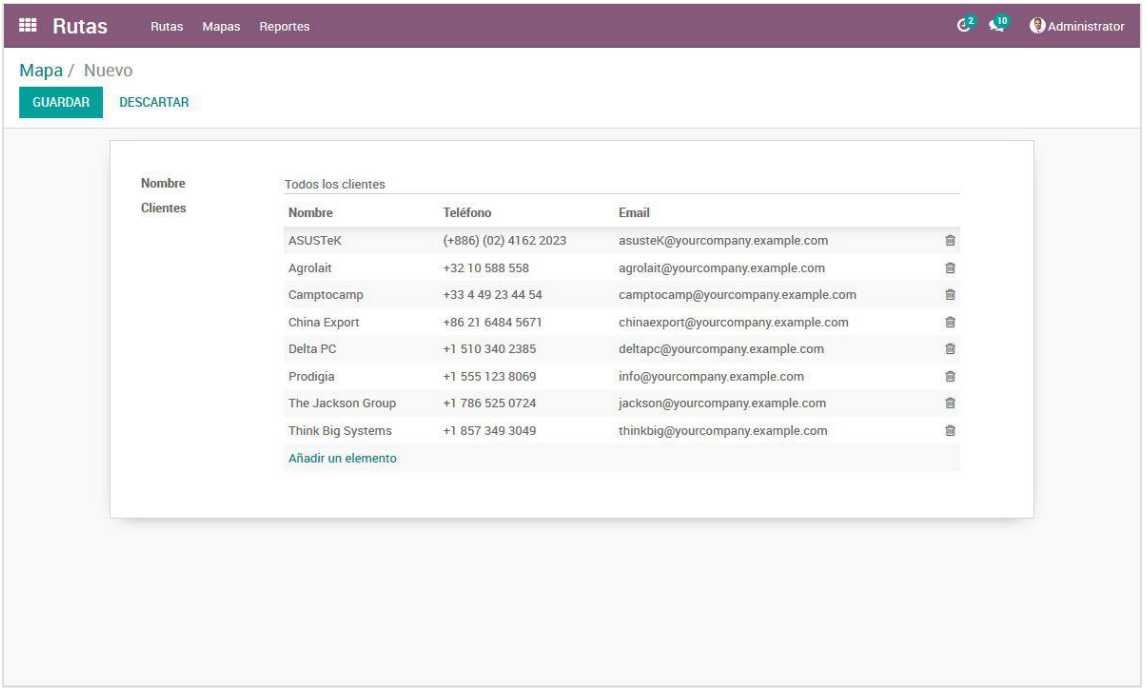

Figura 4. Creación de un mapa estático.

Rápidamente se generará un mapa como el de la figura 4.1. Está función la sugirió uno de los clientes de Prodigia ya que es útil para ver la distribución de los clientes de una empresa y así ver a qué zonas atacar con publicidad, por ejemplo. La idea es que un mapa más poblado se vea como en la figura 4.2.

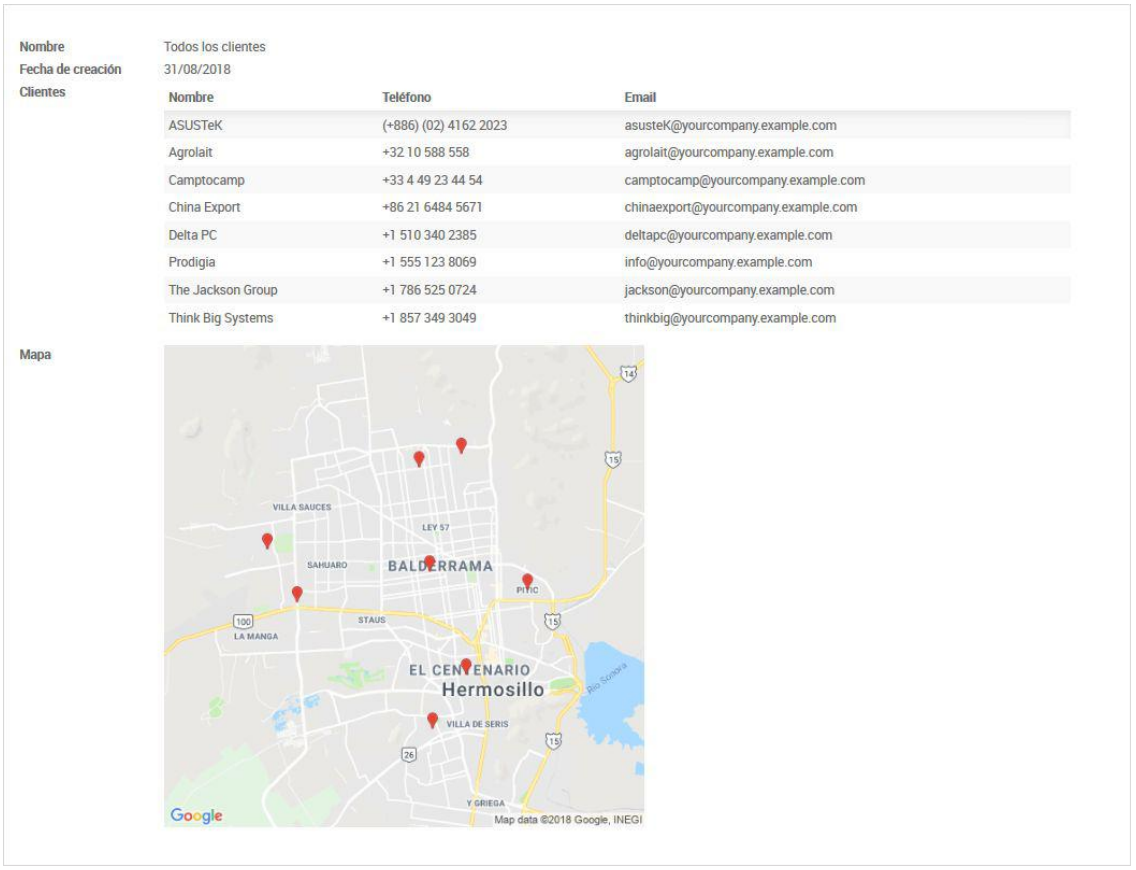

Figura 4.1. Creación de un mapa con la ubicación de clientes seleccionados.

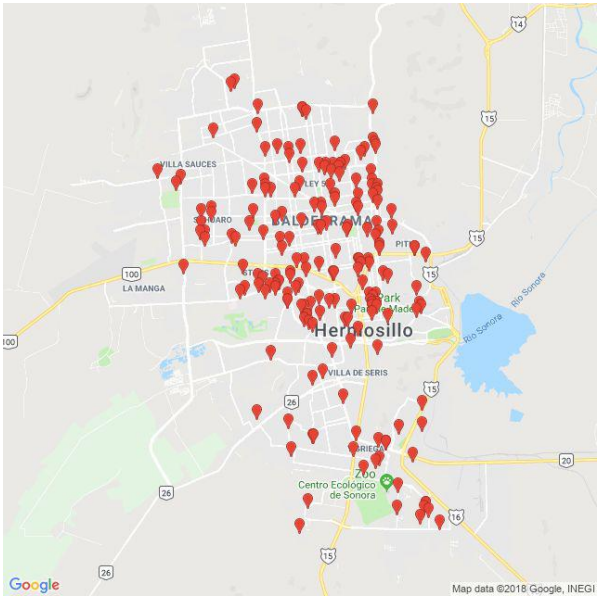

Figura 4.2. Un mapa altamente poblado resalta las zonas geográficas que una empresa domina.

Se implementó la funcionalidad de enviar rutas a los empleados que estén registrados en Odoo (haciendo uso del módulo de directorio de empleados). Para esto se hace click en la barra de navegación (Rutas > Enviar rutas). El formulario requiere que seleccione un empleado y una o más rutas (figura 5).

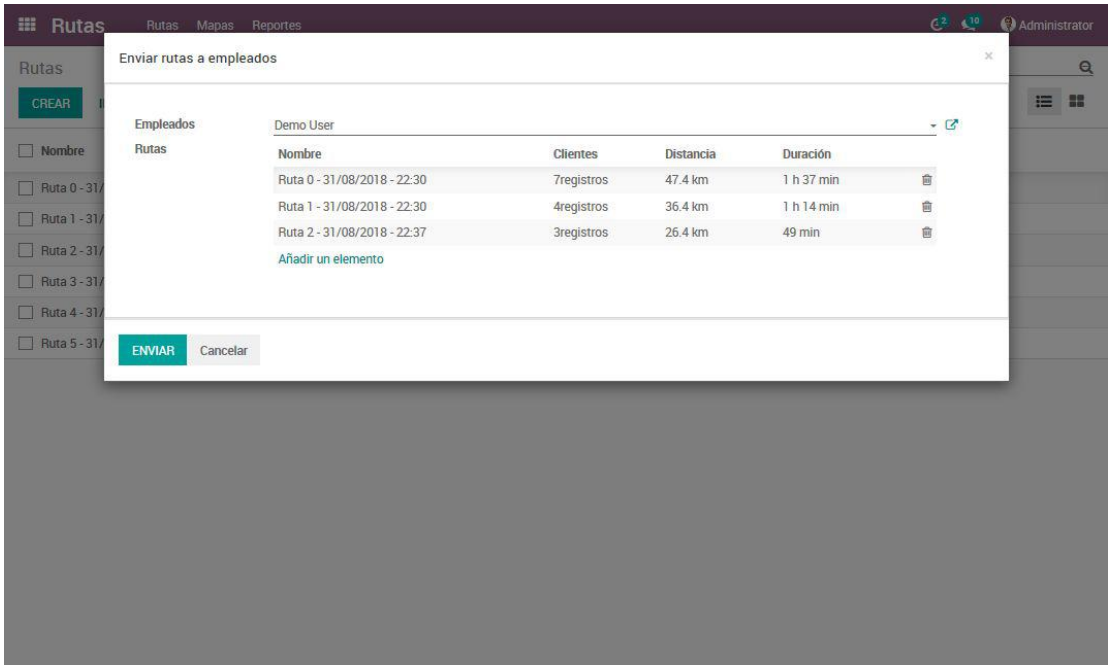

Figura 5. Enviar rutas a empleados por e-mail.

Figura 5.1. Plantilla de correo electrónico.

El correo electrónico usa una plantilla (figura 5.1) y anexa los links publicos (para que se puedan acceder fuera de Odoo) de las rutas que se seleccionaron en el formulario. Estas rutas se pueden

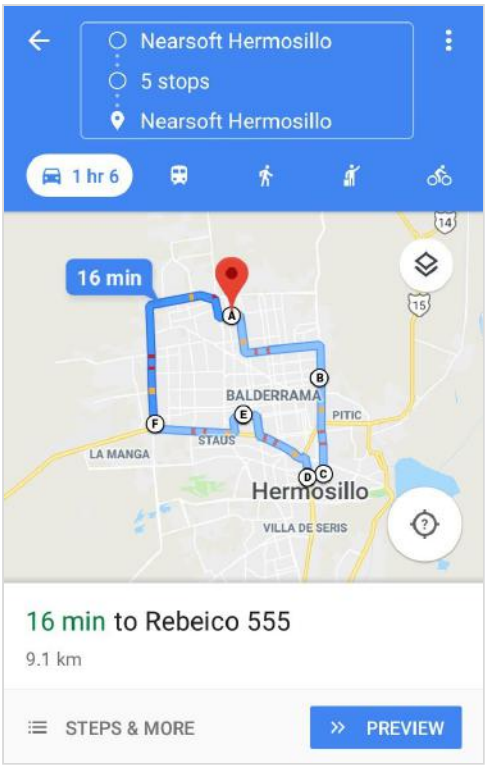

Figura 5.2. Ruta en app de Google Maps.

ver desde un navegador web o un celular, como se muestra en la figura 5.2. También se puso en marcha la sección de reportes para ver las visitas a cada cliente (figura 6). Esto sirve, por ejemplo, para ver a los clientes más frecuentes de una empresa de soporte técnico que arregla equipo de cómputo a domicilio. Así se pueden dar más atención a estos clientes o tomar otro tipo de decisiones, como de mercadotecnia.

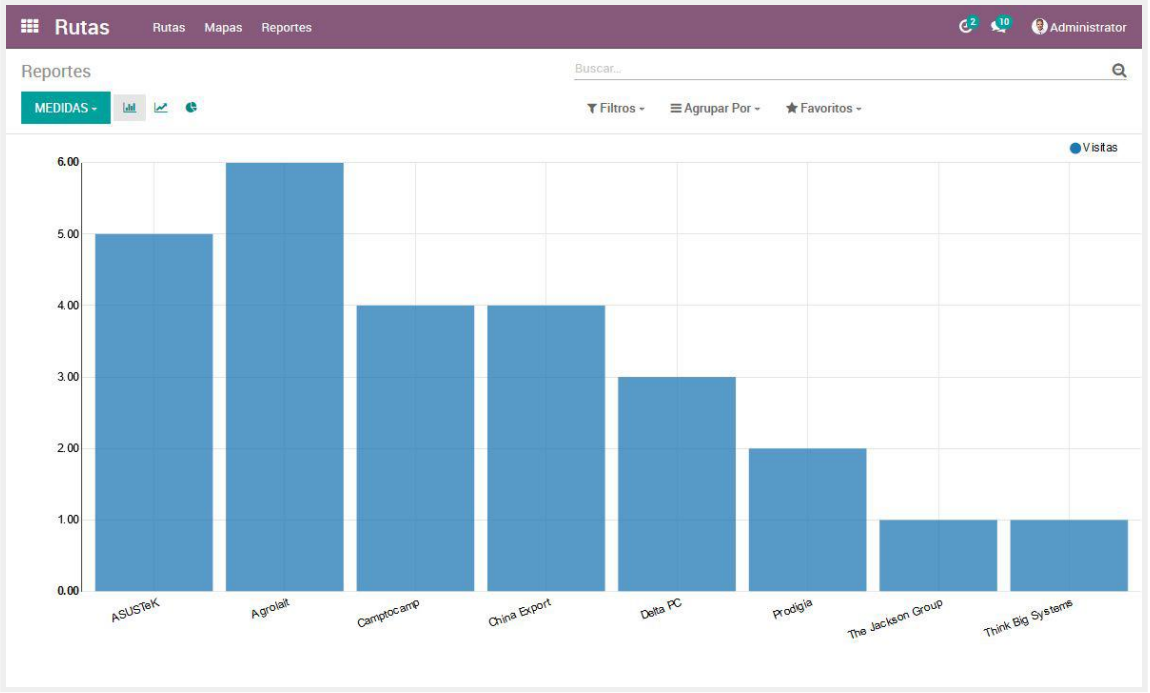

Figura 6. Gráfica de barras que muestra las veces que cada cliente fue seleccionado para ruta.

Respecto a la herramienta para importar archivos XML a la sección de compras, esta se accede desde el módulo de facturación que aparece en la figura 2. Al entrar, se debe explorar la barra de navegación hasta llegar a la opción “Importar XMLs” como se ve en la figura 7.

Figura 7. Opción de importar XMLs bajo el menú de compras.

Después se abrirá un formulario en donde se debe escoger un archivo ZIP que contenga una o más facturas dentro (figura 7.1). Una vez seleccionado el archivo, se presiona el botón validar e importar. Si el archivo es válido y las facturas son correctas, estas se importarán en estado de borrador (figura 7.3). Si hay algún error en los archivos, se le avisará al usuario por medio de una alerta, como en la figura 7.2.

Figura 7.1. Formulario para importar XMLs.

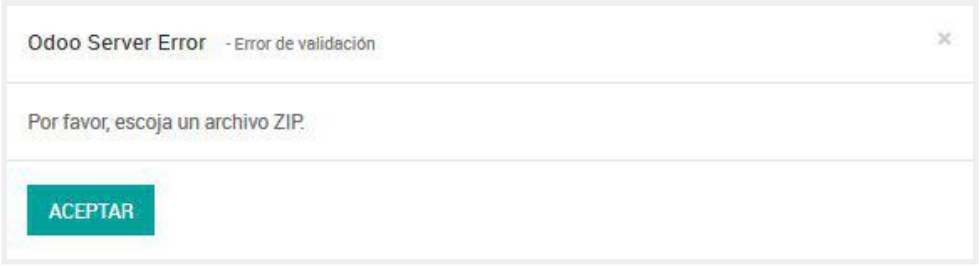

Figura 7.2. Error de validación al no seleccionar un archivo ZIP.

Finalmente, la factura se guarda en Odoo y muestra los datos contenidos en el XML como conceptos (productos), valor unitario, cantidades, impuestos, descuentos, retenciones, importes, proveedor, fecha, entre otros (figura 7.3).

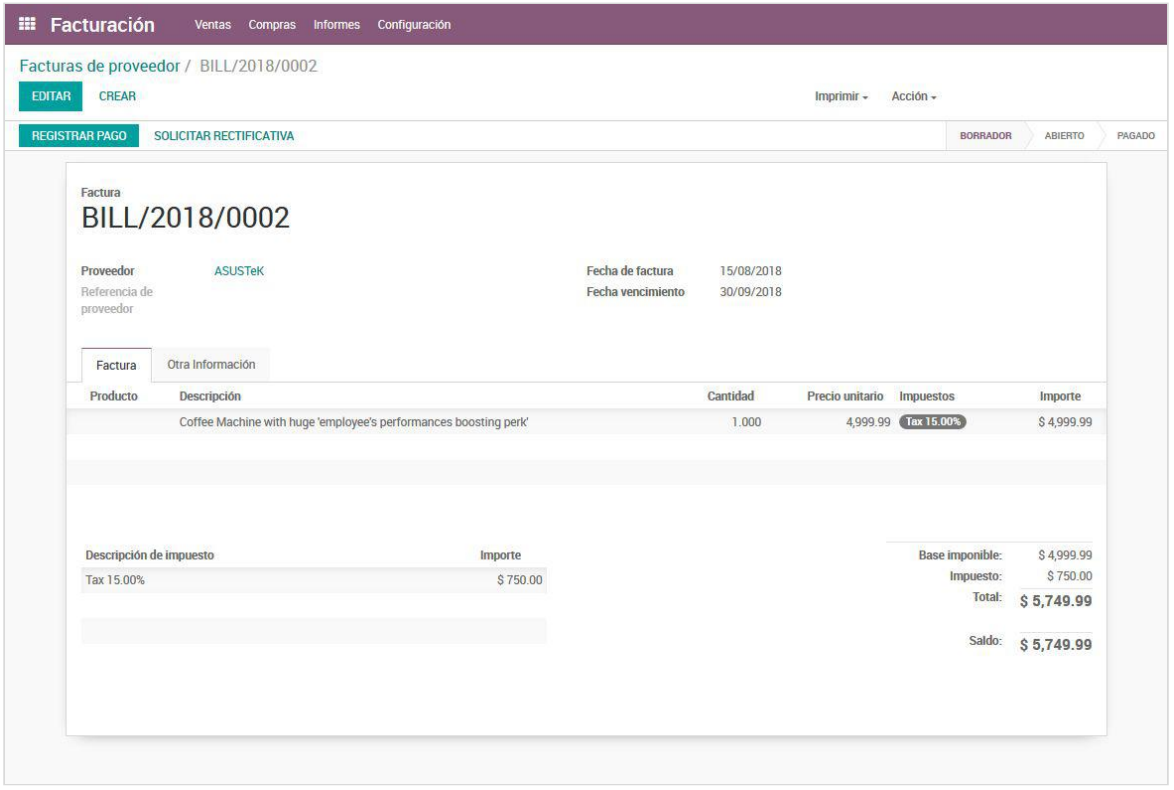

Figura 7.3. Una factura de proveedor que ha sido importada en estado de borrador.

Como referencia, anexo un ejemplo de una factura electrónica en formato XML. (figura 7.4). En esta se pueden observar datos del SAT como el sello (el cual se usa para validar la autenticidad de la factura a través de un web service de Prodigia), los conceptos con sus respectivos datos, impuestos y retenciones. Cada uno de estos datos se presenta con elementos y etiquetas XML, los cuales son transformados a JSON para su manipulación dentro de Odoo.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<cfdi:Comprobante xmlns:cfdi="http://www.sat.gob.mx/cfd/3" Version="3.3" Folio="16BC"
Fecha="2017-10-30T09:09:23" Sello="kqFJelLV199l8DunR98pn5k8uCwi94k" NoCertificado="200010000037"
Certificado="MIIGBzCCA+gAwIBuoaissUl0dpw" SubTotal="22694" Moneda="MXN" Total="1436" Descuento="0">
    <cfdi:Emisor Rfc="LAHH850905BZ4" Nombre="HORACIO LLANOS" RegimenFiscal="608" />
    <cfdi:Receptor Rfc="HEPR930322977" Nombre="RAFAEL ALEJANDRO HERNÁNDEZ PALACIOS" />
    <cfdi:Conceptos>
        <cfdi:Concepto Cantidad="2" Descripcion="ACERO" ValorUnitario="150" Importe="300">
            <cfdi:Impuestos>
                <cfdi:Traslados>
                    <cfdi:Traslado Base="300" TipoFactor="Tasa" TasaOCuota="0.16" Importe="36" />
                </cfdi:Traslados>
                <cfdi:Retenciones>
                    <cfdi:Retencion Base="300" ipoFactor="Tasa" TasaOCuota="0.53" Importe="160" />
                </cfdi:Retenciones>
            </cfdi:Impuestos>
        </cfdi:Concepto>
    </cfdi:Conceptos>
    <cfdi:Impuestos TotalImpuestosRetenidos="1196" TotalImpuestosTrasladados="363">
        <cfdi:Retenciones>
            <cfdi:Retencion Impuesto="002" Importe="272" />
            <cfdi:Retencion Impuesto="003" Importe="1193" />
        </cfdi:Retenciones>
        <cfdi:Traslados>
            <cfdi:Traslado Impuesto="002" TipoFactor="Tasa" TasaOCuota="0.16" Importe="363" />
        </cfdi:Traslados>
    </cfdi:Impuestos>
</cfdi:Comprobante>
```

Figura 7.4. Ejemplo de factura XML. Algunos datos, como el timbrado fiscal, fueron omitidos.

Conclusiones y recomendaciones

Las prácticas profesionales se desarrollaron en una empresa cuya área más fuerte y la de más enfoque es la de desarrollo. Debido a esto, recibe gran parte de los recursos y es muy agradable trabajar en un ambiente así. Sin embargo, hay detalles que pueden mejorar dentro de la empresa.

- Documentación de los procesos internos. Para alguien que es nuevo en una empresa, es muy útil saber exactamente qué hacen, cómo lo hacen y quién lo hace. De igual manera, el saber qué hacer y qué no hacer puede ayudar a los nuevos empleados a evitar futuros conflictos.
- Documentación de los proyectos. Las primeras semanas fue difícil comenzar a trabajar ya que no podía adaptarme con facilidad a los proyectos existentes, debido a que no había un lugar para leer sobre estos y cómo funcionan. El código de algunos proyectos hechos en Prodigia no está comentado y así es difícil contribuir a estos sin pedir ayuda constantemente.
- Minutas de reuniones. Estuve presente en varias juntas del equipo de desarrollo, en las cuales se llegaron a acuerdos importantes para los productos de la empresa. Sin embargo, nadie grabó y escribió nada acerca de los resultados y acuerdos obtenidos en las juntas, y eso puede afectar en un futuro al querer dar seguimiento o presentar resultados a la gerencia. Además, tener un escrito de lo sucedido en las reuniones sirve para informar a los que no estuvieron presentes y también para consultas futuras y reenfoque empresarial.
- Flexibilidad de instalar sistemas operativos diferentes a Windows en los equipos de cómputo. Sería bueno facilitar y documentar el proceso de instalar Linux en los equipos, ya que para los que trabajamos en Odoo/Python, Linux es mucho más amigable.

Por último, no cabe duda que Odoo es una excelente solución de manejo de recursos para las empresas. Es una herramienta muy poderosa, configurable al 100% y con cientos de módulos que pueden ajustarse a casi cualquier empresa. Sin embargo, al desarrollar módulos nuevos puede ser difícil encontrar buena documentación. Yo recomendaría a Prodigia que, si quieren construir herramientas digitales, no se queden solamente con Odoo, ya que existen otros frameworks de desarrollo (Django, por mencionar uno) más robustos y fáciles de implementar.

Retroalimentación

El programa de Ingeniería en Sistemas de Información de la Universidad de Sonora enseña como base de la programación el paradigma orientado a objetos, y no cabe duda que dicho paradigma es el rey a la hora de abstraer los procesos empresariales que desarrollan las empresas hoy en día. Mis conocimientos en ese tema fueron bien aplicados. Así mismo, la experiencia obtenida en la universidad acerca de las bases de datos fue de bastante utilidad, aunque se podría profundizar más en funciones de agregación, ya que es una de las maneras más rápidas para mostrar información relevante con datos. También los alumnos podrían sacar provecho al aprender a manejar otros sistemas de bases de datos como PostgreSQL.

Una de las cosas que aprendí en la escuela y que me ayudó mucho durante el desarrollo fue la habilidad para analizar los requerimientos de un software para resolver un problema, ya que puede ser difícil traducir el texto de un cliente a código, pero el nivel de atención que desarrollé como estudiante me ayudó a abstraer hasta el más mínimo detalle.

Sin embargo, Git y GitHub, una de las herramientas más utilizadas por un desarrollador hoy en día no se enseña en este programa educativo. El tener un sistema de control de versiones es muy útil para cualquier proyecto de toda empresa, y está entre las habilidades más solicitadas en el mercado laboral. Estaría bien que se enseñara la importancia de esto en el programa educativo de la carrera. Por fortuna, esto es algo que yo aprendí por mi cuenta y por pláticas/talleres elaborados por el laboratorio *CSI* de la Universidad de Sonora.

Dicho esto, el desarrollo de una actitud autodidacta es algo que muy pocas materias y profesores cultivan en los alumnos, sin embargo es algo que, en mi opinión, algo que te diferencía de la competencia. Es muy importante la habilidad de leer y comprender documentación y foros de ayuda de diferentes

lenguajes de programación y otras tecnologías. Además, el nivel inglés que la carrera requiere no es suficiente en mi opinión, y debería de ser más alto ya que absolutamente todo lo referente a programación se lee en inglés.

Recientemente ha habido un *boom* en el uso de Python, el lenguaje de programación. El crecimiento se debe a la cantidad de librerías que existen, la buena documentación, rapidez, pero sobre todo la facilidad de uso y comprensión de código. Además es utilizado extensivamente en el manejo de datos e inteligencia artificial, un área a la que los nuevos estudiantes de la carrera se enfrentarán mucho estos próximos años. Por estas razones opino que Python sería una mejor opción para enseñar programación a alumnos de primer ingreso, ya que la complejidad textual y la sintaxis de Java puede alejarlos e intimidarlos del bello arte de programar.

Por último, el trabajar en una empresa formal puede ser un poco intimidante para un estudiante. Conocer gente nueva, responder a superiores, responsabilidades reales con consecuencias, relacionarse con tus compañeros… todo eso puede ser difícil. Me hubiera gustado aprender técnicas para desarrollar las llamadas *soft skills*, las cuales potencian habilidades de comunicación, trabajo en equipo, gestión de tiempo, resolución de conflictos, liderazgo, confianza y seguridad en uno mismo, negociación y otras aptitudes que ayudan a desenvolverte mejor en el área profesional.

Referencias bibliográficas y virtuales

- Odoo. 2018. *Developer Documentation*. Disponible en: https://www.odoo.com/documentation/11.0/. (Ingresado el 05/06/2018)
- Thinkific. 2018. *Odoo training Center*. Disponible en https://odoo.thinkific.com/. (Ingresado el 05/06/2018)
- Odoo. 2018. *Installing Odoo*. Disponible en:https://www.odoo.com/documentation/11.0/setup/install.html. (Ingresado el 08/06/2018).
- Odoo. 2018. *Building a Module*. Disponible en:https://www.odoo.com/documentation/11.0/howtos/backend.html. (Ingresado el 14/06/2018).
- Odoo Cloud Platform. 2018. *Documentation*. Disponible en:https://www.odoo.com/documentation/user/11.0/odoo_sh/documentation.html. (Ingresado el 19/06/2018).
- Google Maps 2018. *Google Maps Platform Documentation*. Disponible en:https://developers.google.com/maps/documentation/. (Ingresado el 22/06/2018).
- GitHub. 2018. *Connecting to GitHub with SSH.* Disponible en: https://help.github.com/articles/connecting-to-github-with-ssh/. (Ingresado el 18/06/18).

UNIVERSIDAD DE SONORA

COORDINACIÓN DIVISIONAL DE INGENIERIA

PRÁCTICAS PROFESIONALES

FPP-4

DEPARTAMENTO: Ingeniería Industrial

UNIDAD REGIONAL CENTRO CAMPUS HERMOSILLO

REPORTE FINAL DE ACTIVIDADES

Periodo: Del 20 / Julio / 2018 al 2 / Agosto / 2018

Cantidad de 340 Horas de un total de 340 Avance: 100 %

Nombre del practicante: Juan Daniel Grijalva Soto

Expediente: 214201526 **Programa Educativo (Licenciatura):** Ing. en Sistemas

Nombre del Programa/Proyecto: Módulos de Odoo

Datos de la Unidad Receptora (Razón Social): Prodigia Procesos Digitales Administrativos S. A. de C.V.

Responsable de la Unidad Receptora (Nombre/Puesto): Rafael Carrillo / Desarrollador

Contacto: Teléfono/UR: 2 10 4963 Ext. Celular:

DESCRIPCIÓN GENERAL DE ACTIVIDADES
Se terminó de desarrollar la funcionalidad de importar facturas XML de proveedores en el módulo de facturación de Odoo. Se agregó la funcionalidad de agregar impuestos a la factura, y se arregló un bug al momento de importar una factura cuyo proveedor no existe. Por último, se hizo entrega de los proyectos, subiendolos en repositorios privados de GitHub en la cuenta de Prodigia.

RETROALIMENTACIÓN (Comentarios del tutor)

En caso de requerirse, anexar reportes, formatos, diagramas que apoyen las actividades realizadas.

Para las Ingenierías deberá anexar ***reporte técnico*** en archivo electrónico ≤ 2 MB y carta de terminación de prácticas firmada por el responsable de la empresa.

Observaciones Generales:

Daniel G. Juan Daniel Grijalva Soto	Prodigia PRODIGIA PROCESOS DIGITALES ADMINISTRATIVOS SA DE CV PPD 1011 E A3	Rafael Carrillo Rosario
Nombre y firma del alumno	**Nombre y firma del tutor de prácticas profesionales UniSon.**	**Nombre y firma del responsable de la unidad receptora Sello de la UR**

Original entregar en físico al *Coordinador o Responsable de Prácticas Profesionales de la carrera.*
Copia *para Tutor de Prácticas Profesionales y* ***Copia*** *alumno.*
Enviar en PDF los documentos *al coordinador/responsable de prácticas profesionales de la carrera.*

(25/04/2018)

Prodigia®

Prodigia Procesos Digitales Administrativos S.A. de C.V.

Hermosillo, Sonora a 03 de agosto de 2018

A QUIEN CORRESPONDA:

Por medio de la presente hago de su constar que **Juan Daniel Grijalva Soto** con número de expediente **214201526** de la carrera de Ing. Sistemas De la Información en la Universidad de Sonora llevo a cabo sus prácticas profesionales en nuestra empresa **Prodigia Procesos Digitales Administrativos SA de CV** ubicada en Antonio Quiroga 21 int 3 Col. Quinta Emilia, Hermosillo, Sonora México, C.P. 83214 en el periodo de *05 de junio al 02 de Agosto* del presente año, el área de desarrollo, en el proyecto "Módulos de Odoo" el cual consto de una duración de ***340 horas***.

Sin más por el momento quedo a sus órdenes.

Atentamente.

Prodigia
PRODIGIA PROCESOS DIGITALES ADMINISTRATIVOS SA DE CV
PPD 101129 EA3

Ing. Rafael Misai Carrillo Rosario
Responsable Prodigia Procesos Digitales Administrativos SA de CV