


Desarrollo del sistema SUNT		Versión: 3.0
Equipo de trabajo: Tomas marin Aristizábal, simón Cárdenas Villada, Juan pablo Yepes, Juan Andrés vera		

Tabla de contenido

Sección 1: Aspectos generales de la entrega	2
Introducción.....	2
Sección 2: Evaluación Sprint anterior	2
Sección 3: Planificación Sprint actual	3
Sección 4: Aspectos estructurales y arquitectónicos de la solución	5
Arquitectura propuesta	5
Vista Lógica - Diagrama de Clases de Implementación	5
Vista Física - Diagrama de Despliegue	7
Sección 5: Principios de diseño	7
Principios de Diseño.....	7
Clean Code	9
Sección 6: Funcionalidad y demostración	10
.....	10
Conclusiones y lecciones aprendidas	11
Referencias	11

Sección 1: Aspectos generales de la entrega

Introducción

Este documento tiene como finalidad describir y planear la elaboración de un software para la realización de trámites de tránsito. Se aborda este tema porque al ver las opciones que existen en la actualidad, es evidente que, trámites como la actualización de datos, ingreso de nuevos registros y consulta de estos no es sencillo de realizar para el usuario común porque en algunos casos requiere el desplazamiento del usuario a una sucursal física. De esta manera surge la necesidad de crear una nueva herramienta web que tenga la accesibilidad y facilidad de uso como pilares de diseño.

Sección 2: Evaluación Sprint anterior

En el sprint anterior desarrollamos varias historias de usuario entre las cuales se encontraban toda la parte de consulta tanto de vehículos, personas y licencias.

Técnica de las 4L:

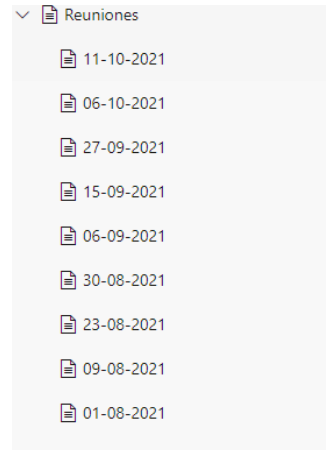
Linked: Durante el desarrollo de la segunda entrega del proyecto SUNT el equipo se desenvolvió bien a la hora de hacer tanto el código como la entrega, siempre se asistió a las reuniones que se plantearon y cada uno realizó sus historias de usuario correspondientes.

Learned: Aprendimos a relacionarnos de una buena manera, a entender y respetar las opiniones de los demás, realizamos un proyecto en DJANGO y aprendimos en si como se desenvuelve este framework, como también todo lo relacionado a los estilos arquitectónicos que usamos y su impacto en el código.

Lacked: En la toma de decisiones muchas veces no nos poníamos de acuerdo ya que algunos querían unas cosas u otros otras cosas, por esto es por lo que muchas veces requerimos votaciones o preguntarle a la profesora para que determinara que le parecía mejor y podernos poner de acuerdo lo cual generaba un gasto de tiempo innecesario.

Longed for: Hubiéramos preferido tener unas reuniones más estructuradas y organizadas, ya que en algunas ocasiones no se avanzaba en todos los ítems requeridos o no se sabía por dónde empezar, lo que alargó el proceso de toma de decisiones en el código.

Link reuniones que realizamos de manera semanal :
<https://dev.azure.com/tmarina0307/SUNT%20%20PI>



Sección 3: Planificación Sprint actual

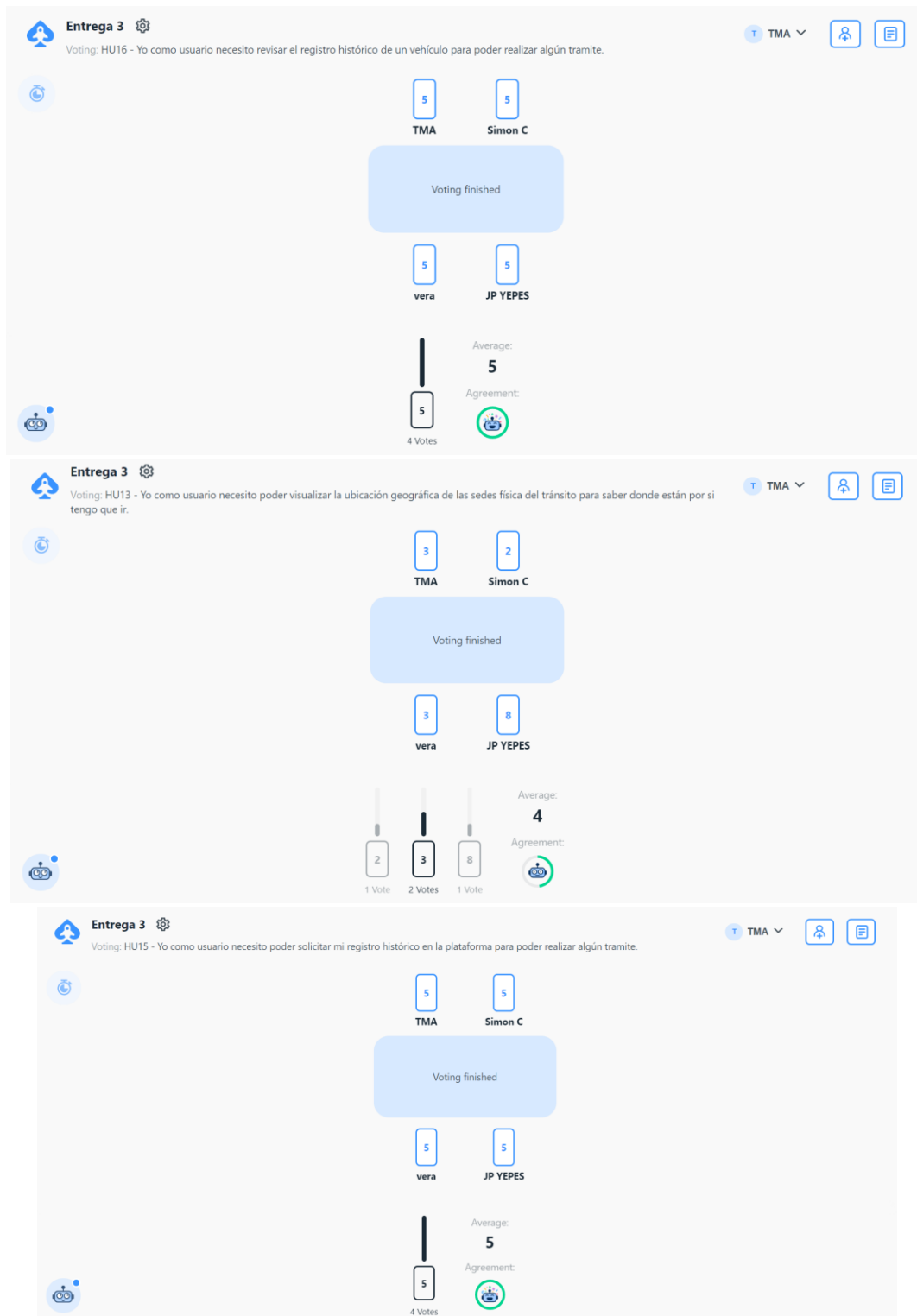
Durante este sprint vamos a realizar las siguientes historias de usuario HU05, HU08, HU09, HU13, HU15, HU16 y HU17 en los cuales se encuentran las consultas que puede realizar el usuario sobre sus vehículos , licencia e información personal.

Tabla de historias de usuario aplicando planing poker.

HU5	6.6
HU8	4
HU9	5.5
HU13	4
HU15	5
HU16	5
HU17	3

Azure Backlog: <https://dev.azure.com/tmarina0307/SUNT%20IS>

Evidencia aplicación de técnica de planing poker:

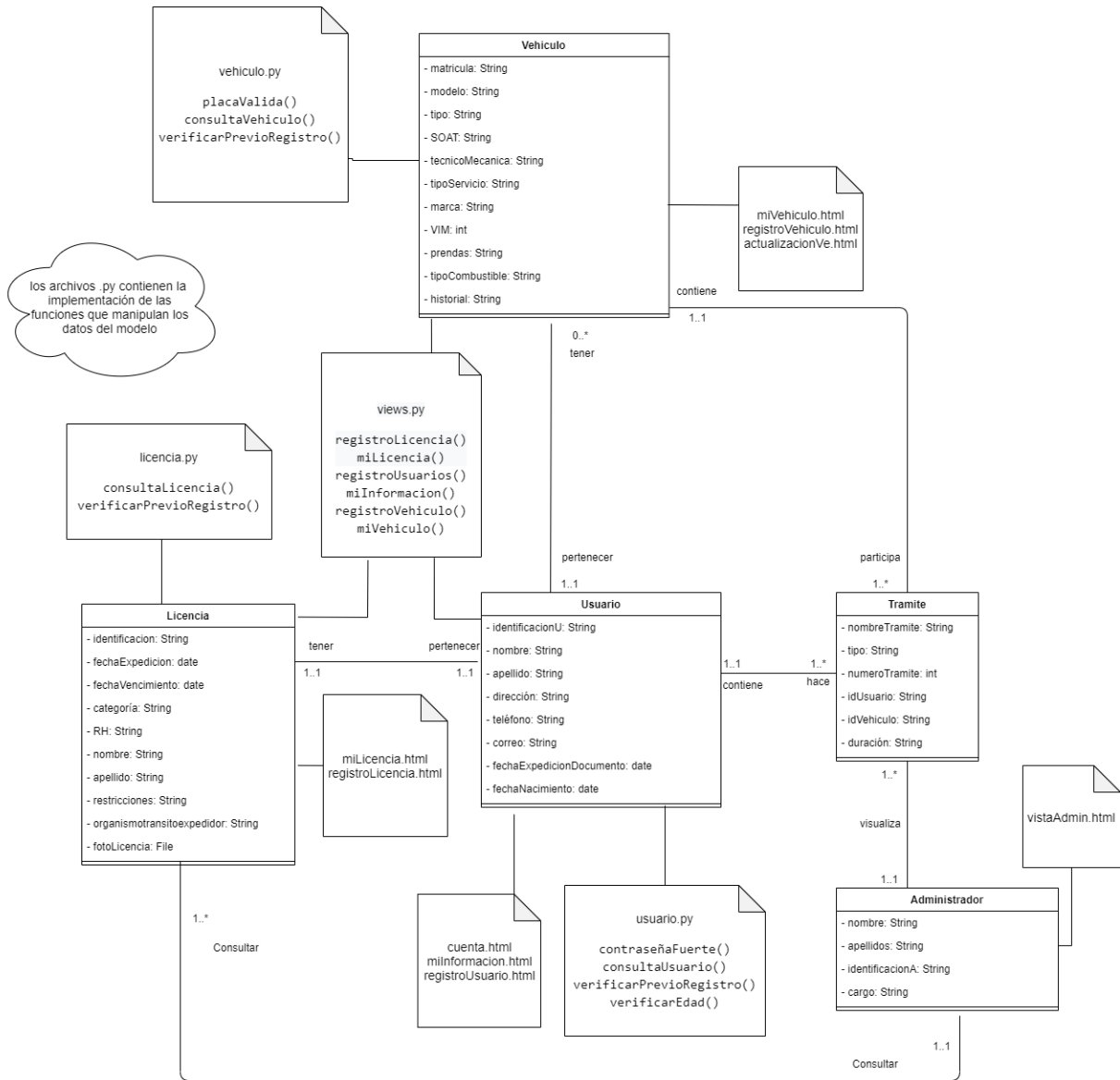


Sección 4: Aspectos estructurales y arquitectónicos de la solución

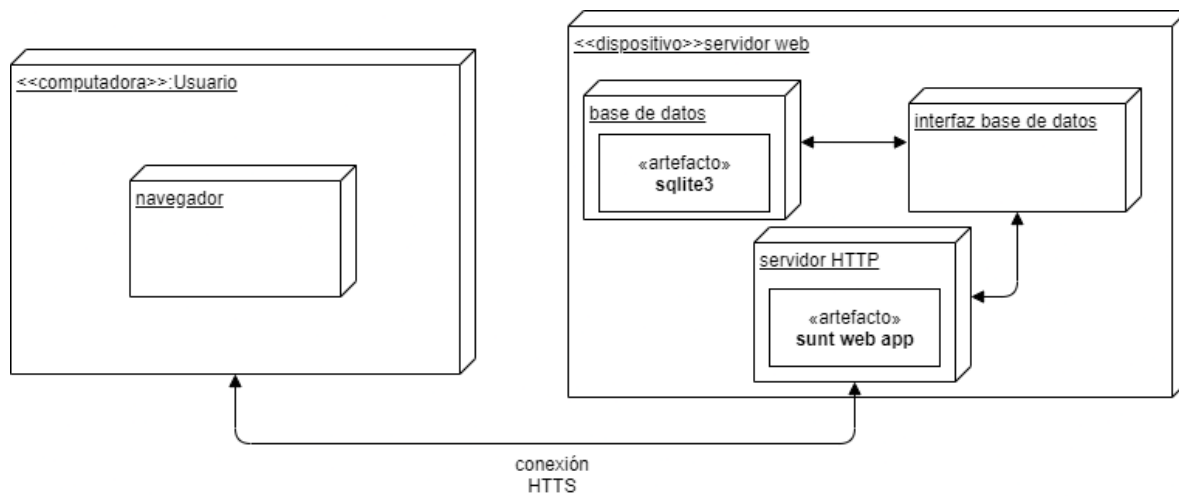
Arquitectura propuesta

Tipo Aplicación:	Web
Estilo Arquitectónico:	<ul style="list-style-type: none">● Client/Server (Implementación): Porque queremos hacer modificaciones fácilmente sin que este todo compacto, como también al tener los recursos centralizados hace que el manejo de datos sea más eficiente y esta arquitectura nos da beneficios en la escalabilidad y encapsulamiento.● Layered (Estructura): Layered porque usamos una arquitectura donde se diferencian las capas de presentación, persistencia y lógica de negocio haciendo más fácil el desarrollo y mantenimiento de la aplicación en el tiempo.
Lenguaje programación	Python
Aspectos técnicos	Usaremos una base de datos relacional que en concreto es SQLite3
Frameworks	Usaremos como framework DJANGO

Vista Lógica - Diagrama de Clases de Implementación



Vista Física - Diagrama de Despliegue



Sección 5: Principios de diseño

Principios de Diseño

Principios SOLID	
Single Responsibility	Django maneja urls que funcionan como un controlador a la hora de usar la página ya que gestionan los accesos que debe de tener la página. Cada función que pusimos en los archivos cumple con acciones específicas enfocadas a lo que busca este archivo en concreto.
Open/Closed	Por la estructura de los archivos los métodos están separados y no dependen de otros, si se agregan nuevos métodos estos no afectaran la estructura del sistema, ni su comportamiento.
Liskov substitution	No lo implementamos en nuestro proyecto ya que no usamos la herencia debido a que nos orientamos a un paradigma más funcional que de clases y por ende no tenemos la posibilidad

	de aplicarlo.
Interface segregation	Lo implementamos separando lo que se le muestra al usuario con lo que se le muestra al administrador, de esta manera el usuario no vera opciones que no pueda usar y tendrá una mejor interfaz gráfica sin información irrelevante para él.
Dependency inversion	<p>Lo aplicamos con la parte del controlador que tiene como principio DJANGO y en la manera en que podemos modificar la base de datos que usemos por otra sin mayores cambios en la estructura del código gracias al ORM que proporciona el framework.</p> <p>Los requests de la capa de presentación no dependen directamente de lo que esté contenido en los HTML o en el CSS, sino que se comunican con el controlador de las vistas.</p>

Principios GRASP	
Bajo Acoplamiento	Se puede evidenciar en la forma en que la lógica de negocio y los archivos que gestionan las urls y las vistas no están relacionadas todos entre sí, sino que las dependencias están separadas y asociadas únicamente con los módulos requeridos.
Alta Cohesión	Se puede ver en la separación de archivos relacionados con la lógica de cada una de las partes ósea usuarios, vehículos o licencias (usuarios.py, vehículos.py, licencias.py).
Experto	<p>Tenemos archivos especializados (usuarios.py, vehículos.py, licencias.py) en acciones específicas y que no contienen información que no sea de su competencia.</p> <p>Los renders también ya que solo se encargan de una plantilla en específico y no interactúan con las demás.</p>
Creador	Lo vemos cuando se ingresa o consulta algún dato en la base de datos DJANGO crea un objeto temporal que luego lo traduce a SQL para hacer las operaciones de CRUD.
Controlador	La parte del controlador la podemos evidenciar en

	DJANGO de por sí ya que este usa el modelo M(modelo)V(vista)T(template) donde la parte del controlador se ve en las distintas vistas que se encargan de gestionar los métodos de la lógica, el intercambio de información con la persistencia y la comunicación al front-end para que sea presentada al usuario.
Polimorfismo	No lo implementamos en nuestro proyecto ya que no usamos la herencia debido a que nos orientamos a un paradigma más funcional que de clases y por ende no tenemos la posibilidad de aplicarlo.
Fabricación Pura	No lo usamos porque al abordar el problema no encontramos la necesidad de usar APIs o clases que se salieran del dominio de nuestro problema.
Indirección	No importa si se cambia la base de datos no tenemos que cambiar todo, el ORM se encarga de gestionar el cambio a la hora de usar la base de datos.
No Hables con extraños	Por la manera en que se crearon los modelos y gracias al ORM de Django, para acceder a la información que gestionamos en SQLite solo es necesario realizar un query que devuelve un objeto con todos los atributos necesarios, y, por ende, no es necesario invocar métodos o clases extrañas para visualizar la información.

Clean Code

- Aplicamos clean code en la parte de los nombres de los archivos los cuales son precisos y nos dan una idea de lo que tienen adentro.
- Siempre realizamos el código con la mayor claridad posible y de manera que solo tenga lo que realmente se usa y necesita como la indentación, conservar el mismo estilo en todo el desarrollo del proyecto.
- En el código solo está comentado lo que se necesita, no tenemos comentarios sueltos o que no tengan una funcionalidad diferente a explicar el código.

Sección 6: Funcionalidad y demostración

Link GitHub: <https://github.com/tmarina1/PI.git>

The image displays three sequential screenshots of a web application interface, likely for vehicle management or registration. Each screenshot features a dark blue header with the SUNT logo and navigation links: Inicio, Sobre Nosotros, Cuenta, Normas, and Trámites. A search bar labeled 'Buscar' is positioned on the right side of the header.

Screenshot 1: Datos Vehículo
This screen shows a list of vehicle details in a light green area. The details include:
Placa: djo259
Modelo: 2021
Tipo: Camioneta
SOAT: [SOAT.pdf](#)
Tecnico mecanica: /media/Entrega2.pdf
Servicio: Particular
Marca: kia
VIM: 87598347859
Prendas: /media/SUNTuso.pdf
Tipo combustible: gasolina
Historial: /media/SUNTuso.pdf

Screenshot 2: User Profile
This screen displays a user profile in a light green area. It includes a circular profile picture placeholder, the name 'Tomas', the ID '1001016308', the email 'todostomas@hotmail.com', and a 'Cerrar sesión' (Log out) button.

Screenshot 3: Registro vehículo
This screen shows a vehicle registration form in a light green area. The form is titled 'Registro vehículo' and contains several input fields and buttons for data entry:
Matricula: Ingresar la matricula
Modelo: Ingresar el modelo
Tipo: Ingresar el tipo
SOAT: Seleccionar archivo N..o
Técnico mecánica: Seleccionar archivo N..o
Tipo de servicio: Ingresar el tipo de servici
Marca: Ingresar la marca

Conclusiones y lecciones aprendidas

- Lo que logramos aprender es identificar los diferentes patrones que existen en nuestro código.
- Aprendimos diferentes funcionalidades de DJANGO para hacer ciertas cosas en nuestro código como lo es el login.
- Es bueno observar como a través de los principios de diseño, es más ordenada la implementación, y puede servir para estar en constante actualización del software

Referencias