

Universidad Central
"Marta Abreu" de Las Villas



Departamento: Ciencia de la Computación

TRABAJO DE DIPLOMA

Título: Aplicación web para el pronóstico y manejo de datos
relacionados con la incidencia de *Phyllachora maydis*

Autor: Alain Daniel Rodríguez Domínguez

Tutores: M. Cs. Alejandro Cespón

Dr.C. Orlando Miguel Saucedo Castillo

Santa Clara , noviembre de 2024

Copyright©UCLV

Universidad Central
"Marta Abreu" de Las Villas



Academic Department: Computer Science

DIPLOMA THESIS

Title: Web Application for the Forecast and Data Management
Related to the Incidence of *Phyllachora maydis*

Author: Alain Daniel Rodríguez Domínguez

Thesis Directors: M. S. Alejandro Cespón

Ph.D. Orlando Miguel Saucedo Castillo

Santa Clara, November 2024
Copyright©UCLV

Este documento es Propiedad Patrimonial de la Universidad Central “Marta Abreu” de Las Villas, y se encuentra depositado en los fondos de la Biblioteca Universitaria “Chiqui Gómez Lubian” subordinada a la Dirección de Información Científico Técnica de la mencionada casa de altos estudios.

Se autoriza su utilización bajo la licencia siguiente:

Atribución- No Comercial- Compartir Igual



Para cualquier información contacte con:

Dirección de Información Científico Técnica. Universidad Central “Marta Abreu” de Las Villas. Carretera a Camajuaní. Km 5½. Santa Clara. Villa Clara. Cuba. CP. 54 830

Teléfonos.: +53 01 42281503-14190



Hago constar que el presente trabajo fue realizado en la Universidad Central “Marta Abreu” de Las Villas como parte de la culminación de los estudios de la especialidad de Ciencia de la Computación, autorizando a que el mismo sea utilizado por la institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos ni publicado sin la autorización de la Universidad.

Firma del autor

Los abajo firmantes, certificamos que el presente trabajo ha sido realizado según acuerdos de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

Firma del tutor

Firma del tutor

Firma del jefe del Dpto

DEDICATORIA

A Cristo Jesús, mi Salvador,
muerto por mis pecados,
resucitado en vida eterna
y glorificado por los siglos de los siglos.

AGRADECIMIENTOS

A Dios, mi Creador y mi Padre, razón suprema de todo cuanto soy o hago.

A mis padres, mi hermana y mi amada prometida Yeni, apoyo indescriptible de mi vida e impulso de mi trabajo.

A toda mi familia.

A los hermanos de Antioquía, la feliz compañía de mis años universitarios.

A mis tutores Alejandro Cespón y Orlando Saucedo, por su ayuda efectiva, y por su tiempo y amabilidad.

A mis profesores y compañeros de aula, gracias a todos por su apoyo.

RESUMEN

La Facultad de Ciencias Agropecuarias de la Universidad Central de Las Villas “Marta Abreu” requiere de un sistema que apoye la investigación sobre la incidencia de la enfermedad fúngica mancha de asfalto (*Phyllachora maydis*) en el cultivo del maíz (*Zea mays*) mediante el registro y análisis de observaciones meteorológicas y el pronóstico de la afectación basado en dichas mediciones. Para ello se propone e implementa una aplicación web construida con los marcos de trabajo Next.js (que se sustenta en React, una biblioteca de JavaScript) para el contenido y la funcionalidad, y Tailwind (basado en el lenguaje CSS) para la estilización en el lado del cliente, y con Django y Django REST Framework (que usan el lenguaje de programación Python) en el lado del servidor, así como un servidor PostgreSQL para la gestión de la base de datos. Se implementó además un sistema de credenciales para la autenticación de los usuarios autorizados a modificar la información manejada. La aplicación web resultante permite la introducción, recuperación y eliminación de datos sobre las observaciones, las estaciones meteorológicas y las unidades de cultivo del maíz. Posibilita la emisión automática de pronósticos de advertencia cuando el riesgo de aparición y desarrollo de la enfermedad en los sembrados monitoreados lo haga necesario.

Palabras clave: maíz, *Phyllachora maydis*, aplicación web, Django, Next.js

ABSTRACT

The Faculty of Agropecuary Sciences of the Central University of Las Villas “Marta Abreu” lacks a system to support the investigation of the incidence of the tar spot fungal disease (*Phyllachora maydis*) in the growing of corn (*Zea mays*) through the recording and analysis of weather observations and the forecast of possible affectations on the basis of those records. To that effect, the author implements and proposes a web application built with the frameworks Next.js (which is built on the JavaScript library React) for content and functionality, and Tailwind (based on the CSS language) for styling in the front-end, and with Django and Django REST Framework (which use the programming language Python) in the back-end. Furthermore, a PostgreSQL server was used for database management and a system of credentials was implemented for the authentication of the users that are allowed to make changes to the information in the system. The resulting web application makes it possible to input, retrieve and delete data corresponding to weather observations, meteorological stations and corn growing units. It is able to issue warnings when the estimated risk of presence and development of the fungal disease in the investigated corn fields becomes high.

Key words: corn, *Phyllachora maydis*, web application, Django, Next.js

CONTENIDOS

INTRODUCCIÓN	1
CAPÍTULO 1. MARCO TEÓRICO Y CONCEPTUAL	5
1.1 <i>Aspectos técnicos del negocio y requisitos de la aplicación web</i>	<i>5</i>
1.2 <i>Aplicaciones web</i>	<i>6</i>
1.2.1 <i>Ventajas y clasificación</i>	<i>6</i>
1.2.2 <i>Arquitectura cliente-servidor</i>	<i>8</i>
1.2.3 <i>API y API REST</i>	<i>8</i>
1.3 <i>Conceptos de lenguaje de programación y marco de trabajo</i>	<i>9</i>
1.4 <i>Tecnologías de desarrollo del lado del cliente</i>	<i>10</i>
1.4.1 <i>HTML, CSS y JavaScript</i>	<i>10</i>
1.4.2 <i>JSON</i>	<i>11</i>
1.4.3 <i>React y Next.js</i>	<i>11</i>
1.4.4 <i>Tailwind</i>	<i>12</i>
1.5 <i>Tecnologías de desarrollo del lado del servidor</i>	<i>12</i>
1.5.1 <i>Python</i>	<i>12</i>
1.5.2 <i>Django</i>	<i>13</i>
1.5.3 <i>Django REST Framework</i>	<i>13</i>
1.6 <i>Bases de datos</i>	<i>13</i>
1.6.1 <i>Sistema gestor de bases de datos</i>	<i>14</i>
1.6.2 <i>PostgreSQL</i>	<i>14</i>
1.7 <i>Autenticación y autorización</i>	<i>14</i>
1.7.1 <i>NextAuth y Django REST Framework</i>	<i>15</i>
1.8 <i>Conclusiones parciales del capítulo</i>	<i>16</i>
CAPÍTULO 2. DESCRIPCIÓN DE LA PROPUESTA TÉCNICA IMPLEMENTADA	17
2.1 <i>Modelación UML de aspectos generales de la aplicación web</i>	<i>17</i>
2.2 <i>Estructura e implementación del lado del servidor</i>	<i>20</i>
2.2.1 <i>Diseño de la base de datos, acoplamiento con Django e implementación en PostgreSQL</i>	<i>20</i>
2.2.2 <i>Implementación del servidor Django</i>	<i>24</i>
2.3 <i>Estructura e implementación del lado del cliente</i>	<i>28</i>
2.3.1 <i>Rutas y páginas</i>	<i>28</i>
2.3.2 <i>Componentes</i>	<i>30</i>
2.3.3 <i>Implementación de la autenticación</i>	<i>34</i>
2.4 <i>Conclusiones parciales del capítulo</i>	<i>35</i>
CAPÍTULO 3. PRUEBAS REALIZADAS AL SOFTWARE	36
3.1 <i>Aspectos generales de las pruebas de software</i>	<i>36</i>
3.1.1 <i>Propósito</i>	<i>36</i>
3.1.2 <i>Metodologías</i>	<i>36</i>
3.2 <i>Pruebas realizadas a la aplicación web propuesta</i>	<i>38</i>
3.2.1 <i>Pruebas concernientes al lado del servidor</i>	<i>38</i>
3.2.2 <i>Pruebas concernientes al lado del cliente</i>	<i>40</i>
3.2.3 <i>Pruebas de la aplicación como un todo</i>	<i>40</i>
3.3 <i>Conclusiones parciales del capítulo</i>	<i>41</i>
CONCLUSIONES	42
RECOMENDACIONES	43
REFERENCIAS BIBLIOGRÁFICAS	44

LISTA DE FIGURAS

<i>Figura 1: Diagrama de casos de uso.....</i>	<i>18</i>
<i>Figura 2: Diagrama de actividad del caso de uso "agregar observación meteorológica"</i>	<i>18</i>
<i>Figura 3: Diagrama de despliegue de alto nivel de la aplicación web.....</i>	<i>19</i>
<i>Figura 4: Esquema entidad-interrelación de la base de datos implicada por las reglas del negocio (modelo conceptual).....</i>	<i>20</i>
<i>Figura 5: Configuración de conexión a la base de datos en Django.....</i>	<i>21</i>
<i>Figura 6: Diagrama de entidad-interrelación lógico de las relaciones creadas.....</i>	<i>22</i>
<i>Figura 7: Tablas creadas por las migraciones de Django en la base de datos PostgreSQL. .</i>	<i>23</i>
<i>Figura 8: Tablas adicionales requeridas para el funcionamiento de la autenticación, permisos, migraciones y administración.....</i>	<i>23</i>
<i>Figura 9: Árbol de archivos de la implementación del servidor Django.....</i>	<i>24</i>
<i>Figura 10: Diagrama de clases de los modelos de Django de datosmaiz.....</i>	<i>25</i>
<i>Figura 11: Implementación del modelo Pronostico en models.py.....</i>	<i>26</i>
<i>Figura 12: Implementación de la función emitir_pronosticos().</i>	<i>27</i>
<i>Figura 13: Árbol de rutas de páginas del lado del cliente.....</i>	<i>29</i>
<i>Figura 14: Captura de la página inicial.</i>	<i>29</i>
<i>Figura 15: Captura de la página de registros.....</i>	<i>30</i>
<i>Figura 16: Árbol de archivos de la carpeta components.....</i>	<i>30</i>
<i>Figura 17: Fragmento de código del componente <Table/>.....</i>	<i>31</i>
<i>Figura 18: Fragmento de código del componente <TableSelector/>.....</i>	<i>31</i>
<i>Figura 19: Fragmento de código del componente <InputForm/></i>	<i>32</i>
<i>Figura 20: Vista del formulario generado por el componente <InputForm/> en la interfaz gráfica.....</i>	<i>32</i>
<i>Figura 21: Uso del componente <InputForm/> en la página de datos de unidades de cultivo</i>	<i>33</i>
<i>Figura 22: Código JSX producido por el componente <Statistics/></i>	<i>33</i>

<i>Figura 23: Sección de la interfaz gráfica producida por el componente <Statistics/></i>	<i>.33</i>
<i>Figura 24: Pestaña de la barra de navegación para la autenticación .</i>	<i>..... 34</i>
<i>Figura 25: Fragmento de código del componente <Navbar/> de lógica de autenticación .</i>	<i>35</i>
<i>Figura 26: Prueba escrita con Pytest para la vista estaciones_list(request).....</i>	<i>38</i>
<i>Figura 27: Resultado de la ejecución de las pruebas realizadas con Pytest.</i>	<i>39</i>
<i>Figura 28: Captura de la API navegable de Django REST Framework que muestra pronósticos emitidos en las pruebas.....</i>	<i>39</i>

INTRODUCCIÓN

El maíz (de nombre científico *Zea mays*) se destaca como un cultivo de interés para pequeños, medianos y grandes productores debido a su versatilidad y al aprovechamiento de su cosecha en diversos ámbitos, incluyendo su creciente uso como cobertura vegetal de suelo en países tropicales y subtropicales. Esta tendencia refleja la importancia que el maíz ha adquirido en la agricultura moderna, no solo como fuente de alimento, sino también como recurso para mejorar la salud del suelo y promover prácticas agrícolas sostenibles. Su capacidad para adaptarse a diferentes condiciones climáticas y su considerable papel en la seguridad alimentaria lo hacen un cultivo esencial en la producción agrícola a nivel mundial (Valdez Ocampo et al., 2024).

Estos hechos justifican el interés de los investigadores en la protección de la salud de los cultivos de maíz, pues esta planta no está exenta de amenazas en el ámbito fitosanitario. Las plagas tienen el potencial de disminuir sustancialmente el volumen y/o la calidad del maíz producido, así como la capacidad de la planta para realizar su función de cobertura vegetal; por tanto, los estudios dirigidos a entender mejor las enfermedades que afectan al maíz, para poder así combatirlos mejor, son una necesidad para la correcta explotación de estos cultivos. Entre las más importantes de ellas están el carbón común (*Ustilago maydis*) que afecta principalmente la mazorca; el carbón de la espiga (*Sporisorium reilianum*), que afecta la espiga y la mazorca; la roya común (*Puccinia sorghi*) y la enfermedad fúngica mancha de asfalto (Díaz-Morales et al., 2018).

Esta última es provocada por el hongo ascomyceto *Phyllachora maydis*, que fue descrito inicialmente en 1904 en México e incide en diversos países de América Central y Suramérica. En 2015 fue registrado por primera vez en los Estados Unidos, en Illinois e Indiana, y después se propagó a otros estados (European and Mediterranean Plant Protection Organization, 2024).

La presencia de este reduce el rendimiento y en algunos casos puede llegar a causar la muerte de las plantas de maíz. En América latina se han documentado pérdidas de rendimiento de hasta el 46%. La mancha de asfalto se caracteriza por la presencia de estromas negros en las hojas. Estas manchas son producto de los cuerpos fructíferos del hongo, un biótrofo obligado (debido a su incapacidad para obtener nutrientes de células muertas y su dependencia de plantas de maíz) que comienza infectando la parte baja de la planta, aunque puede infectar vainas o mazorcas. Una vez establecido, se distribuye a través de la hoja provocando manchas que llegan a unirse,

causando posteriormente necrosis, senescencia prematura y muerte (Góngora-Canul et al., 2022; Vines-Tachong et al., 2022).

La afectación de las plantas de maíz por la mancha de asfalto está muy relacionada con las condiciones climáticas en las que se encuentre. La velocidad del viento que dispersa las ascosporas, los valores entre los cuales varían la temperatura y la humedad relativa, etcétera, son factores que influyen en el riesgo de que la enfermedad incida en los cultivos, razón por la cual el registro diario de observaciones meteorológicas es una parte clave del estudio de la misma.

Esta enfermedad del maíz resulta de interés a la Facultad de Ciencias Agropecuarias de la Universidad Central de Las Villas “Marta Abreu”. Sin embargo, todo estudio moderno, en cualquier campo de investigación, debe auxiliarse de las tecnologías de la información recientes, y los estudios fitosanitarios del maíz no son una excepción, sino que ya existen precedentes para ello, como el trabajo de (Vines-Tachong et al., 2022), que describe la implementación del procesamiento de imágenes para la evaluación de la presencia de la misma enfermedad que concierne a la presente investigación. El uso de soportes digitales aventaja notablemente a los métodos tradicionales de manejo de datos basados en papel manuscrito o impreso, ya que disminuye los riesgos de redundancia e inconsistencia de la información, aumenta abrumadoramente la facilidad de copiado y modificación de la misma, reduce notablemente el volumen del material requerido para almacenarla, prescinde de los costos continuos de nuevas hojas y tinta y, además de todo lo anterior, contribuye al prestigio de la universidad como centro interesado por la modernidad tecnológica y la digitalización.

El desarrollo de un sistema informático para el estudio y pronóstico de la incidencia de *Phyllachora maydis* es una novedad en Cuba, y la propia investigación sobre la enfermedad en el país resulta reciente, de acuerdo con las indicaciones del cliente experto, el Dr.C. Orlando Miguel Saucedo Castillo, investigador titular del Centro de Investigaciones Agropecuarias (CIAP) de la Universidad. Actualmente en la Facultad de Ciencias Agropuecuarias se maneja los datos sobre las observaciones meteorológicas relevantes a la misma mediante hojas de cálculo de Microsoft Excel y ficheros en formato `.txt`; sin embargo, el trabajo basado solo en herramientas como las anteriores, aunque superior al soporte en papel, es susceptible a problemas de redundancia de la información, carece de plena integración como sistema y no se vale de facilidades más recientes del panorama tecnológico. Además, la automatización del procesamiento de datos

para realizar predicciones sobre la incidencia de la enfermedad ahorraría esfuerzo humano experto en el pronóstico de las afectaciones de la misma; y la Facultad carece de tal facilidad.

Por otra parte, de estar asentado en la red de la universidad, el sistema que dé respuesta a esta necesidad podrá eliminar las distancias entre oficinas de investigadores, ya que sería accesible desde cualquier computadora o dispositivo móvil dentro del campus. Las técnicas basadas en el trabajo en línea se perfilan tecnológicamente como un frente de innovación; el desarrollo de soluciones web avanza a un ritmo impresionante, impulsado por una alta demanda y sustentado por multitud de opciones y herramientas en constante evolución que compiten entre sí y se ofrecen como alternativas efectivas para el desarrollador y le permiten satisfacer cada vez mejor las necesidades de los usuarios finales (Kralina and Popova, 2024).

Todo ello constituye la **justificación de la investigación** realizada.

De las consideraciones anteriores se deduce el siguiente **problema de investigación**: ¿Cómo desarrollar un sistema informático que auxilie a los investigadores asociados a la Facultad de Ciencias Agropecuarias de la mencionada Universidad en el manejo de datos relevantes para el estudio de la incidencia de la mancha de asfalto en los cultivos de maíz y el pronóstico de la misma?

Para abordar de manera efectiva esta problemática se plantea cuatro **preguntas de investigación** esenciales:

1. ¿Qué requisitos debe cumplir el sistema a desarrollar?
2. ¿Cuáles prácticas y tecnologías actuales para el desarrollo de sistemas informáticos basados en actividad en línea (aplicaciones web) son adecuadas para la solución del problema?
3. ¿Cómo implementar una aplicación web que responda a las necesidades planteadas por el problema que motiva su realización?
4. ¿Cómo probar la propuesta técnica resultante?

Considerando el problema y las preguntas de investigación formuladas, se proponen los siguientes objetivos generales y específicos:

Objetivo general: desarrollar una aplicación web que permita a los investigadores asociados a la Facultad de Ciencias Agropecuarias el manejo de datos de observaciones relacionadas con la

incidencia de la enfermedad fúngica mancha de asfalto en el cultivo de maíz en las unidades estudiadas, y emita pronósticos automáticos de riesgo de afectación por dicha enfermedad.

Objetivos específicos:

1. Definir los requisitos que debe cumplir el sistema.
2. Justificar la selección de prácticas y tecnologías actuales para el desarrollo de aplicaciones web.
3. Implementar la aplicación web que cumpla con los requisitos determinados.
4. Realizar pruebas de software a la aplicación desarrollada.

Estructura del trabajo: el presente informe se organiza en una introducción, tres capítulos, conclusiones, recomendaciones y referencias bibliográficas.

En el capítulo 1 se aborda los requisitos de la aplicación web basados en los aspectos técnicos del negocio, así como las tecnologías modernas de desarrollo de web escogidas para ser empleadas en la solución del problema.

En el capítulo 2 se describe el análisis técnico y los elementos más relevantes de la implementación.

En el capítulo 3 se describen las técnicas de pruebas y las pruebas realizadas al software.

CAPÍTULO 1. MARCO TEÓRICO Y CONCEPTUAL

El presente capítulo cubre los fundamentos técnicos del problema cuya solución se persigue que resultan relevantes al desarrollo de la aplicación web propuesta, así como los conceptos, enfoques, tecnologías y herramientas empleadas para la implementación de este.

1.1 Aspectos técnicos del negocio y requisitos de la aplicación web

La aplicación web a desarrollar debe ser capaz de almacenar, recuperar y eliminar información relevante a la investigación sobre la incidencia de *Phyllachora maydis*; en particular, de los registros diarios de la información meteorológica dados por las estaciones correspondientes, con la fecha de observación, la temperatura máxima, mínima y media en grados Celsius, la humedad máxima, mínima y media en porcentaje, la cantidad de horas con humedad relativa mayor que el 90% (y dentro de ese período, los valores de temperatura máxima, media y mínima), la precipitación del día anterior en milímetros y la velocidad del viento en metros por segundo. De los terrenos de cultivo se debe guardar, con información provista por las entidades de producción, el nombre de la forma productiva correspondiente, la denominación del cultivar, el tipo de suelo, la fecha de siembra, si la semilla tiene tratamiento químico o no, el área sembrada en hectáreas, el tipo de fertilizante y su dosis (en kilogramos por hectárea), el marco de siembra y el sistema de riego. De cada estación meteorológica se tendrá su código, nombre y municipio donde está ubicada.

Otra funcionalidad del software será emitir pronósticos sobre la incidencia de la enfermedad fúngica mancha de asfalto sobre la base de condiciones climáticas. La información especificada por este pronóstico incluye la unidad de producción y el cultivar en riesgo de afectación, el período climático que resultó favorable para la aparición de la enfermedad y que por tanto motivó la emisión del pronóstico, el plazo de predicción de los primeros síntomas de la mancha de asfalto, el tipo de mensaje de predicción, así como el valor total de “grados días” o suma térmica en grados Celsius.

Según las indicaciones directas del cliente experto, el Dr.C. Orlando Miguel Saucedo Castillo, a partir del día de siembra se comienza el cálculo diario de la suma térmica. Cuando el acumulado alcance 700 °C, se comienza a tomar un período de 7 días. Si en esos días se registra una temperatura media diaria entre 17 °C y 22 °C, una máxima menor que 30 °C, mínima superior

a 15 °C, humedad relativa diaria mayor o igual a 90% por más de 7 horas, y precipitación mayor que 25 mm, se da lugar a que el día siguiente sea crítico. Es decir, se toma los días 1 al 7 después de la suma térmica de 700 °C, y si se cumplen los parámetros meteorológicos anteriores el día 8 es crítico y después se va analizando los siguientes 7 días corriendo un día hacia adelante, del 2 al 8, y así sucesivamente. Cuando se obtenga 5 días críticos se emite un pronóstico de alerta, y cuando se llegue a 6 días críticos se da uno de peligro máximo (no tienen que ser estrictamente consecutivos, pues si uno o dos días subsiguientes no resultan críticos, se saltan; pero si se pasan de dos días seguidos que no sean críticos, se reinicia la cuenta). La acumulación de calor (grados-días) se calcula restando 10 °C del valor medio de las temperaturas máxima y mínima; pero si la temperatura máxima es mayor que 30 °C, se reemplaza por 30 °C; y si la mínima es menor que 10 °C, se reemplaza por 10 °C.

1.2 Aplicaciones web

Las aplicaciones web son sistemas accesibles mediante un navegador a través de una red local o Internet, utilizados para la obtención y el procesamiento de datos, que dependen de un servidor físico o virtual en línea para el cumplimiento de sus funcionalidades (Hernández Cruz et al., 2023; Llamuca-Quinaloa et al., 2021).

El campo de desarrollo de aplicaciones web ha experimentado una evolución pronunciadamente rápida desde su nacimiento. En contraste con los sitios web o *websites* estáticos de los días pasados del Internet, que ofrecían contenido sin características interactivas, las aplicaciones web de hoy son integrales en experiencia digital, brindando servicios dinámicos e inmersivos que compiten con los tradicionales programas de escritorio (Panwar, 2024).

1.2.1 Ventajas y clasificación

Una aplicación web disfruta de ciertas ventajas sobre otras clases de sistemas. El mismo hecho de residir en un servidor remoto alivia al dispositivo del usuario de las cargas de almacenamiento y procesamiento que tendría que asumir el ordenador en el caso de una aplicación de escritorio ubicada localmente. Además, ofrece una flexibilidad geográfica y facilidades de sincronización superiores, pues un usuario puede acceder a sus servicios desde cualquier lugar con acceso libre al Internet, y si emplea diferentes dispositivos no necesita ocuparse de mantener la consistencia de la información guardada en ambos, pues el servidor al que accede mantiene sus datos de

aplicación en un respaldo unificado. Por la parte del desarrollador, las aplicaciones web son altamente compatibles y su mantenimiento y actualización son simples, ya que requieren de un único desarrollo para funcionar en diferentes sistemas operativos y se reducen los conflictos de versiones. Todas estas características favorables respaldan la elección de una aplicación web como el tipo de software a implementar para dar solución al problema abordado por el presente trabajo.

Independientemente de las características esenciales que comparten, las aplicaciones web pueden ser asociadas a diferentes clases. Las tres principales son *Single Page Application (SPA)* o aplicaciones de una sola página, *Multi-Page Application (MPA)* o aplicaciones de múltiples páginas, y *Progressive Web Application (PWA)* o aplicaciones web progresivas.

En las SPA la transición entre vistas de la aplicación web no provoca una nueva carga de la página completa, lo cual elimina la necesidad de cargar cada subpágina desde el servidor. Esto se debe a que la SPA carga un único archivo HTML (véase el epígrafe 1.4.1) y después actualiza el contenido de este documento según se requiera. Todo el contenido de la página es cargado simultáneamente, mientras que los datos necesarios son renderizados del lado del cliente (véase el epígrafe 1.2.2), lo cual favorece el rendimiento de la página y ofrece una mejor experiencia de usuario a lo largo de su interacción con la aplicación; sin embargo, la carga inicial toma más tiempo, pues se debe tomar toda la página de la primera vez.

Las MPA representan un enfoque más tradicional, que se basa en la generación dinámica de múltiples archivos HTML del lado del servidor. Cada transición entre subpáginas requiere una nueva solicitud de página para tomar el archivo HTML del servidor, lo cual, al contrario de lo que sucede en las SPA, causa demoras al cargar y mostrar el contenido durante la interacción con el usuario (Kowalczyk and Szandala, 2024).

Las PWA ofrecen funcionalidades que combinan el acceso al servidor con el soporte fuera de línea; se pueden instalar en el dispositivo del usuario y, hasta cierto punto, usarse sin conexión a Internet. También incluyen potencialmente notificaciones emergentes y acceso al hardware del dispositivo. Este enfoque fue desarrollado por Google y otros actores del campo tecnológico para simplificar la creación de aplicaciones multiplataforma, y es una opción adecuada cuando se desee crear aplicaciones web que se sientan como aplicaciones nativas pero que sean más

fáciles de desarrollar y mantener que las aplicaciones nativas tradicionales (Ahyar Muawwal, 2024).

Para dar solución al problema de la presente investigación se desarrolla una aplicación SPA, con su moderno y eficiente enfoque; con ello se evita las dificultades de rendimiento de las MPA sin necesidad de entrar en las complicaciones de implementación de versiones del lado del cliente específicas para diferentes plataformas que implican las PWA.

1.2.2 Arquitectura cliente-servidor

El modelo abstracto cliente-servidor es una arquitectura de sistemas distribuidos en los que una o más computadoras clientes solicitan recursos de un servidor remoto (que es una computadora física o virtual, o un grupo de ellas) a través de una red. El cliente proporciona interfaces adecuadas a los usuarios mediante las cuales pueden pedir recursos del servidor, mientras que este recibe una o múltiples solicitudes (*requests* en inglés), las procesa, y devuelve una respuesta (*response*) al cliente. Este enfoque es parte existencial del Internet, los sistemas bancarios y las redes celulares (Mwamba Nyabuto et al., 2024).

Esta arquitectura cuenta con varias ventajas importantes: la centralización del control (el acceso y los recursos son controlados por el servidor asignado, lo cual reduce el riesgo de uso indebido de los datos y posibilita la actualización unificada de los mismos); el manejo organizado (todos los datos son almacenados en el mismo sitio, lo que facilita su administración); facilidad de respaldo y recuperación de los datos por parte de los usuarios; accesibilidad (el servidor puede brindar servicios a los clientes con independencia de la ubicación de estos); entre otros (Abirami et al., 2019).

El contexto de las tecnologías, herramientas y desarrollo de la parte del cliente se conoce en inglés como *front-end*; y el del lado del servidor, como *back-end*.

1.2.3 API y API REST

Una interfaz de programación de aplicaciones (API, del inglés *Application Programming Interface*) es un protocolo de comunicación entre aplicaciones, un convenio que especifica las formas mediante las cuales un sistema o un componente de un sistema puede acceder a funcionalidades de otro. Las APIs vinculan una amplia variedad de softwares y les proveen la comunicación entre sí, por lo que están presentes a lo largo del mundo digital, desde las redes sociales hasta

las transacciones de comercio electrónico. Para los desarrolladores de software es importante tener acceso a las APIs de los servicios relevantes a su trabajo, que pueden ser creadas por ellos mismos para regular la interacción entre partes de sus propios productos, o provistas por compañías que dominan el panorama tecnológico, como Meta, Microsoft y OpenAI (Nguyen, 2023). Una clase de APIs de singular importancia es conocida como REST (del inglés *REpresentational State Transfer*, transferencia de estado representacional). Fue propuesta por Roy Fielding en 2000 y se ha convertido en un estándar *de facto* para las interacciones de servicios web (Liu et al., 2022).

Las APIs de tipo REST se adhieren al intercambio de datos mediante protocolos estandarizados, e incluyen entre sus principios fundamentales la ausencia de estado (el servidor no almacena información sobre el estado del cliente), la cacheabilidad (las respuestas del servidor deben ser compatibles con el uso de caché), y una interfaz uniforme que simplifique las interacciones cliente-servidor. Usan extremos (*endpoints* en inglés), que son funcionalidades concretas implementadas de un proceso de negocios, y son invocadas con ayuda de una dirección URL (*Uniform Resource Locator*, localizador uniforme de recursos). Las solicitudes del cliente, mediante las cuales este puede acceder a recursos de datos a ser creados, eliminados, actualizados o recuperados, son enviadas a uno de los extremos con una ruta de recurso y un método, que comúnmente es POST (crear), GET (recuperar), PUT (actualizar) o DELETE (eliminar) (Ehsan et al., 2022; Kim et al., 2023).

Una API REST es, por tanto, una opción adecuada para la comunicación entre el servidor y el cliente en la aplicación web propuesta como solución al problema.

1.3 Conceptos de lenguaje de programación y marco de trabajo

Un lenguaje de programación es un lenguaje formal diseñado para especificar instrucciones ejecutables por una computadora digital. Los desarrolladores de hoy cuentan con una variedad enorme de lenguajes de programación para seleccionar, los cuales se distinguen entre sí por sus propósitos o subdominios a los que están dedicados, los paradigmas que siguen para construir programas, el nivel de abstracción que presentan con respecto al código directamente ejecutable por un procesador, entre otros.

En el contexto del desarrollo de aplicaciones web, un marco de trabajo (*framework* en inglés) es una plataforma de software que provee a los desarrolladores herramientas preconstruidas para la implementación de sus proyectos. Un marco de trabajo extiende las características normalmente presentes en el lenguaje sobre el cual se basan y añaden sus propias clases, funciones y capacidades, con el objetivo de simplificar el trabajo de producción y mantenimiento de código (Mendez, 2014).

En el desarrollo de la aplicación web propuesta como solución al problema de esta investigación, se hace uso de varios lenguajes de programación y marcos de trabajo, los cuales serán especificados en los epígrafes subsiguientes.

1.4 Tecnologías de desarrollo del lado del cliente

1.4.1 HTML, CSS y JavaScript

Todo el desarrollo web moderno del lado del cliente está construido sobre tres lenguajes fundamentales: HTML, CSS y JavaScript.

HTML (del inglés *HyperText Markup Language*, lenguaje de marcado de hipertexto en inglés) constituye el esqueleto del desarrollo web, pues provee la estructura básica y el contenido de una página web, definiendo elementos como tablas, formularios, listas y títulos, e identifica dónde las secciones empiezan y terminan; todo ello mediante etiquetas (*tags* en inglés).

CSS (*Cascading Style Sheets*, hojas de estilo en cascada) permite dar estilos y aspectos diferentes a los elementos definidos por HTML, creando reglas sobre el color, la fuente y la forma de mostrar las páginas web. También determina cuándo dichas reglas deben ser usadas, sobre la base de información como el tipo de dispositivo conectado, o en respuesta a una acción del usuario (Mendez, 2014).

JavaScript fue concebido originalmente como un lenguaje simple del lado del cliente para especificar instrucciones que añadieran funcionalidad a las páginas web. Sin embargo, su evolución gradual lo ha transformado en un lenguaje más robusto y versátil, capaz de abordar desafíos de programación complejos, y el surgimiento de marcos de trabajo actuales para JavaScript ha facilitado su transformación en un lenguaje de programación que puede emplearse tanto en el desarrollo del lado del cliente como del servidor (Shukla, 2023).

HTML, CSS y JavaScript son esenciales para el desarrollo del lado del cliente de cualquier aplicación web, y por tanto están plenamente integrados en la aplicación web propuesta por la presente investigación.

1.4.2 JSON

JSON, o notación de objetos de JavaScript (*JavaScript Object Notation* en inglés) es un formato de codificación de datos estandarizado para su transmisión entre sistemas, que se ha hecho muy popular en la industria del desarrollo de software para la lectura y escritura de datos en disco, así como en la comunicación a través de una red. JSON usa la sintaxis de JavaScript para representar datos estructurados de forma concisa, legible por máquinas y humanos, con especificaciones de lenguaje e implementación bastante simples (Stanek and Killough, 2023).

Por las razones anteriores, JSON es escogido como el formato de intercambio de datos entre el cliente y el servidor en la aplicación web desarrollada.

1.4.3 React y Next.js

React es una biblioteca de JavaScript desarrollada y mantenida por Facebook (Meta) para la construcción de interfaces de usuario (UI, siglas de *User Interface* en inglés) de aplicaciones web. Con ella, los desarrolladores pueden adoptar un enfoque declarativo y eficiente para la creación de elementos de UI capaces de ser actualizados dinámicamente sobre la base de cambios en los datos o interacciones del usuario.

React usa el modelo de objeto documento (DOM, *Document Object Model* en inglés) para renderizar y actualizar los elementos de UI eficientemente. La estructura y el comportamiento de estos es descrita con JSX (*JavaScript XML*), un lenguaje específico al dominio que combina sintaxis de JavaScript con XML. La estructura elemental usada por React es el componente (*component* en inglés), que permite el desarrollo modular de unidades reusables que se adaptan mediante propiedades (*props*) ajustables a los diferentes usos del mismo componente (Ferreira et al., 2024).

Next.js es un popular marco de trabajo de React para aplicaciones web, que ofrece como ventajas una experiencia de usuario mejorada, rendimiento superior y eficiencia de optimización de motores de búsqueda (SEO, *Search Engine Optimization* en inglés), y simplificación del desa-

rollo. La evolución de Next.js ha sido estable, y su utilidad para la implementación de aplicaciones con React ha sido abundantemente demostrada por la experiencia de la comunidad de desarrolladores (Patel, 2023; Satter et al., 2023; Venkata Koteswara Rao Ballamudi et al., 2021). Por tanto, la aplicación web implementada usa React con Next.js para el lado del cliente.

1.4.4 Tailwind

Tailwind es un marco de trabajo de CSS usado en el desarrollo del lado del cliente para estilizar las interfaces gráficas de usuario. Se destaca por su popularidad, la claridad de su documentación y su eficiencia de almacenamiento en la compilación de archivos. Trabaja mediante la inserción de clases de utilidades (*utility classes* en inglés) predefinidas (y ampliables mediante personalización) directamente dentro de los elementos HTML, lo cual ofrece gran flexibilidad a los desarrolladores (Vargas Zermeño, 2024).

Debido a lo anterior, Tailwind fue seleccionado como herramienta de estilo para la aplicación web desarrollada.

1.5 Tecnologías de desarrollo del lado del servidor

1.5.1 Python

Python, desarrollado por Guido van Rossum, fue introducido en 1991. Es un lenguaje de programación de propósito general de alto nivel e interpretado. Su metodología y características están diseñadas para ayudar a los programadores a escribir código conciso y fácilmente legible para proyectos de cualquier envergadura. Su sintaxis es clara y simple y se apoya fuertemente en la indentación. Python usa recolección de basura para mejorar el uso de memoria y tiene tipado dinámico. Aunque concebido principalmente como un lenguaje orientado a objeto, es compatible también con los paradigmas funcional y estructurado (particularmente procedural). Viene con una biblioteca estándar notablemente amplia. Es uno de los lenguajes de programación más populares del mundo, y la extensísima comunidad de desarrolladores que lo usan lo ha enriquecido con diversas bibliotecas disponibles de forma gratuita (Kumar and Nandal, 2024).

Todas las ventajas anteriores fundamentan la elección de Python como lenguaje de programación adecuado para la implementación del lado del servidor de la aplicación web.

1.5.2 Django

Django es un marco de trabajo para el lado del servidor escrito en Python que goza de amplia aceptación en la comunidad de desarrolladores por su enfoque en la eficiencia y la simplicidad. Al seguir el patrón modelo-vista-plantilla (MTV, *Model-View-Template* en inglés), Django proporciona una estructura sólida para la implementación de aplicaciones web escalables y de alto rendimiento. Ofrece por defecto una gran variedad de herramientas listas para usar, y se destaca por su soporte para bases de datos, administración de usuarios, seguridad y gestión de formularios (Ramírez-Galvis, 2023).

Estas características justifican la adopción de Django como tecnología de base para el desarrollo del lado del servidor en la aplicación web propuesta.

1.5.3 Django REST Framework

Django REST Framework es un popular paquete de Django usado para construir APIs, usado en la implementación de numerosos servicios web actuales. Añade a Django útiles componentes preconstruidos, como las vistas REST basadas en clases, conjuntos de vistas (*viewsets*) y serializadores, de forma que la creación de las interfaces que el servidor ofrecerá a las aplicaciones del lado del cliente se vuelve más concisa, simple y mantenible para el desarrollador (Gagliardi, 2021).

Todo lo anterior fundamenta el uso de Django REST Framework en el desarrollo de la solución web propuesta.

1.6 Bases de datos

Las bases de datos son colecciones organizadas de información que se almacenan y gestionan en un sistema informático. Están diseñadas para gestionar eficazmente grandes volúmenes de datos estructurados o no estructurados, proporcionan una forma de almacenar y acceder a los datos para su procesamiento, y son por lo general el elemento central del lado del servidor de un sistema de software.

Una clase de gran importancia dentro de las bases de datos es las relacionales. Estas se basan en el álgebra relacional, una teoría que utiliza estructuras algebraicas con una semántica bien fundamentada para modelar datos y definir consultas sobre ellos. Los datos se almacenan como tablas bidimensionales con filas y columnas, y los valores de los datos deben pertenecer a uno

de los tipos definidos en el sistema. En la actualidad estas son las bases de datos más utilizadas (Fernández Iglesias, 2024).

Las bases de datos relacionales son propicias para el manejo de la información con la que ha de trabajar la aplicación web que dé solución al problema de investigación, debido a que se manejará instancias con las mismas estructuras de datos agrupables en relaciones o tablas.

1.6.1 Sistema gestor de bases de datos

Un sistema gestor de bases de datos es una aplicación que sirve para administrar bases de datos e interactuar con ellas de manera eficiente y segura. Proporciona una interfaz para crear, modificar, almacenar y recuperar datos de la base, y gestiona aspectos como la integridad de los datos, la seguridad, la concurrencia y la recuperación en caso de fallos (Fernández Iglesias, 2024).

1.6.2 PostgreSQL

PostgreSQL es un sistema gestor de bases de datos objeto-relacional y de código abierto, que usa y extiende el lenguaje estándar de consulta (SQL, *Standard Query Language* en inglés, muy usado e influyente en la historia de las bases de datos) combinado con diversas características que almacenan y escalan con seguridad el trabajo con grandes volúmenes de datos. PostgreSQL ha obtenido una sólida reputación por su arquitectura, confiabilidad, integridad de los datos, robustez de funcionalidad y extensibilidad, y la dedicación de la comunidad de código abierto que lo respalda. Funciona en todos los principales sistemas operativos (The PostgreSQL Global Development Group, 2024).

Por tales razones, PostgreSQL es una opción adecuada para suplir las necesidades de manejo de base de datos de la aplicación web propuesta.

1.7 Autenticación y autorización

La autenticación es el proceso de verificación de la identidad de un usuario mediante algún tipo de credenciales, como un par de nombre de usuario y contraseña. La autorización es la concesión o negación de acceso a recursos o acciones específicas dentro de un sistema.

Estos conceptos son capas esenciales de seguridad para los sistemas basados en Internet. A lo largo de los años, muchos esfuerzos han sido realizados en aras de propiciar entornos seguros

mediante el uso de mecanismos de autenticación, y se considera que en general las formas convencionales de verificación de identidad ofrecen un buen acercamiento y permiten a los usuarios acceder a aplicaciones web seguras (Olanrewaju et al., 2021).

Dos de las principales estrategias de autenticación en sistemas web son la basada en sesión y la basada en *tokens*. El primero es el más tradicional, y fue dominante en el desarrollo web de días pasados. Consiste en el seguimiento por parte del servidor de una sesión abierta por el cliente y finalmente cerrada al terminar su actividad. El segundo usa *tokens*, unidades de información dadas por el servidor al cliente que deben ser usadas en las solicitudes que este hace al servidor, de forma que el *token* sirve como clave secreta temporal (Balaj, 2017).

Como tipo de *token* para autenticación destaca *JSON Web Token* (JWT), que representa un medio ampliamente adoptado de intercambio seguro de información entre entidades como objetos JSON (véase el epígrafe 1.4.2) que facilitan la confidencialidad de alto nivel. Los JWTs están diseñados para permitir mecanismos de firmado, lo cual se puede lograr empleando criptografía de clave secreta o encriptación de par de llaves pública-privada, protegiendo los datos en la transmisión. Con tales protocolos independientes de almacenamiento de estados de sesión, JWT es adecuado para escenarios como APIs REST y constituye una opción aceptable para la funcionalidad de autenticación en el sistema requerido (Dimitrijevic et al., 2024).

1.7.1 NextAuth y Django REST Framework

NextAuth.js es una solución de autenticación de código abierto para aplicaciones hechas con el marco de trabajo Next.js, tratado en el epígrafe 1.4.3. Es un servicio flexible, fácil de usar, compatible con cualquier sistema de bases de datos, y prestigioso en la comunidad de desarrollo web con Next.js (NextAuth Documentation, 2024).

Django REST Framework, cubierto en el epígrafe 1.5.3, ofrece un rango de métodos de autenticación para salvaguardar las aplicaciones de accesos no autorizados. Entre las técnicas provistas destaca la autenticación basada en *tokens* como un acercamiento popular y robusto, amoldado a las necesidades de las aplicaciones web modernas, que fue abordado en el epígrafe 1.7. En Django REST Framework, a diferencia del *TokenAuthentication* preconstruido, la autenticación JWT no necesita usar una base de datos para validar un *token*. Un paquete significativo para JWT es `djangorestframework-simplejwt`. En cuanto al aspecto de la autorización, los

permisos en Django REST Framework siempre son definidos como una lista de clases de permisos. Antes de ejecutar el cuerpo principal de una *view* cada permiso en la lista es chequeado; y si algún permiso falla, una excepción es elevada, el cuerpo de la *view* no se correrá, y se retornará una respuesta 403 *Forbidden* o 401 *Unauthorized* (Django REST Framework Documentation, 2024).

La combinación de NextAuth.js y las funcionalidades basadas en JWT y clases de permisos de Django REST Framework ofrece un esquema de autenticación y autorización adecuado para el sistema requerido.

1.8 Conclusiones parciales del capítulo

En este capítulo se especificó los requisitos que ha de cumplir la aplicación concebida como respuesta al problema de investigación en el entorno de las facetas técnicas del mismo, y se describió las características más relevantes de las tecnologías de desarrollo web seleccionadas para ser usadas en la implementación del sistema; y así se argumentó que una aplicación web de tipo SPA con la típica arquitectura cliente-servidor, hecha con HTML, CSS y JavaScript mediante las plataformas React, Next.js y Tailwind en el lado del cliente, y Django y Django REST como marcos de trabajo basados en Python y una base de datos PostgreSQL en el lado del servidor, así como una combinación de NextAuth y las facilidades de Django REST para autenticación y autorización, constituye una solución viable y adecuada al problema planteado.

CAPÍTULO 2. DESCRIPCIÓN DE LA PROPUESTA TÉCNICA IMPLEMENTADA

Este capítulo aborda la modelación de la aplicación web propuesta como solución al problema de investigación, así como los principales aspectos estructurales y de implementación de la misma, tanto en el lado del cliente como del servidor. Se describe en términos generales la interfaz presentada al usuario final, y, con el grado de detalle apropiado en cada caso, los principales componentes de software que posibilitan el desempeño de las funcionalidades de la aplicación.

2.1 Modelación UML de aspectos generales de la aplicación web

El lenguaje unificado de modelación (UML, del inglés *Unified Modeling Language*) consiste de un conjunto integrado de diagramas, diseñado para ayudar a los desarrolladores de software a especificar, visualizar, construir y documentar los artefactos de los sistemas, así como modelar procesos de negocio; y, en particular, es parte importante del desarrollo de software orientado a objeto. UML usa notaciones gráficas para expresar el diseño de los proyectos, en diferentes tipos de diagramas que describen un sistema desde distintos puntos de vista. Los diagramas de estructura muestran la forma estática del sistema y sus partes en diferentes niveles de abstracción e implementación, y cómo estas se relacionan unas con otras. Los de comportamiento se enfocan en el desempeño dinámico de los objetos en un sistema, que puede ser descrito en términos de series de cambios en el sistema a través del tiempo cuando está en ejecución (DiagramasUML.com, 2024; Visual Paradigm, 2024).

Entre los diagramas de funcionamiento están los de casos de uso, que representan funcionalidades y captan requisitos de un sistema. Cada caso de uso provee uno o más escenarios que indican cómo se debe interactuar con los usuarios finales o con otros sistemas para lograr un objetivo específico del negocio. Al modelar los casos de uso generalmente se evita usar terminología técnica del desarrollo, prefiriéndose el lenguaje del usuario final o del dominio del experto (Alturas, 2023).

La aplicación web propuesta está concebida para un solo tipo de usuario, el experto en estudios de salud de las plantas que investiga la incidencia de *Phyllachora maydis*; y las formas en que interactuará con la aplicación son las lecturas, adiciones y eliminaciones que realizará sobre la

información relevante guardada, y, fundamentalmente, la consulta de los pronósticos automáticamente emitidos por el software.

En la figura 1 se muestra el diagrama de casos de uso de la aplicación web propuesta.

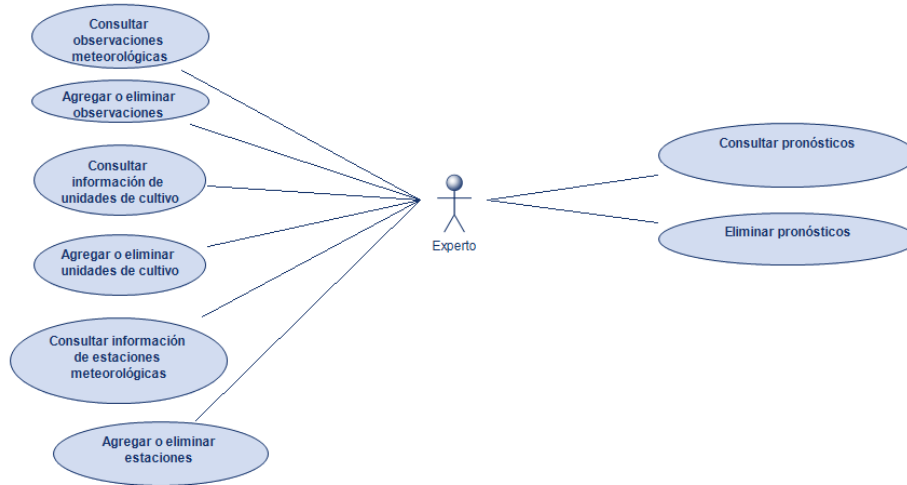


Figura 1: Diagrama de casos de uso.

Cada uno de los casos de uso anteriores es descriptible como una serie relativamente simple de pasos, representable mediante un diagrama de actividades, el cual modela el flujo de acciones realizadas en la interacción usuario-software (Kulkarni et al., 2021).

La figura 2 muestra el diagrama de actividades para la adición de una observación meteorológica diaria. El resto de los casos de uso se modela de forma similar.

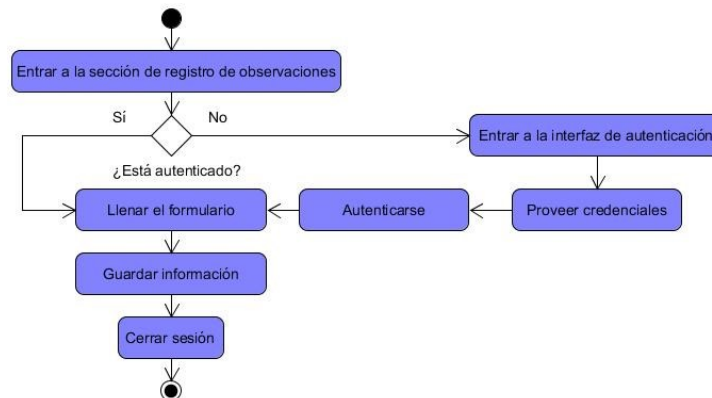


Figura 2: Diagrama de actividad del caso de uso "agregar observación meteorológica".

En cuanto a la descripción estructural de alto nivel de la aplicación web, los diagramas de despliegue muestran la configuración de elementos de procesamiento al momento de la ejecución, así como los componentes de software, procesos y objetos que corren en ellos. Es un grafo de nodos conectados por asociaciones de comunicación, y una de sus funciones es vincular la arquitectura de software con el hardware (Nicacio and Petrillo, 2020).

La figura 3 muestra el diagrama de despliegue de alto nivel de la aplicación web propuesta.

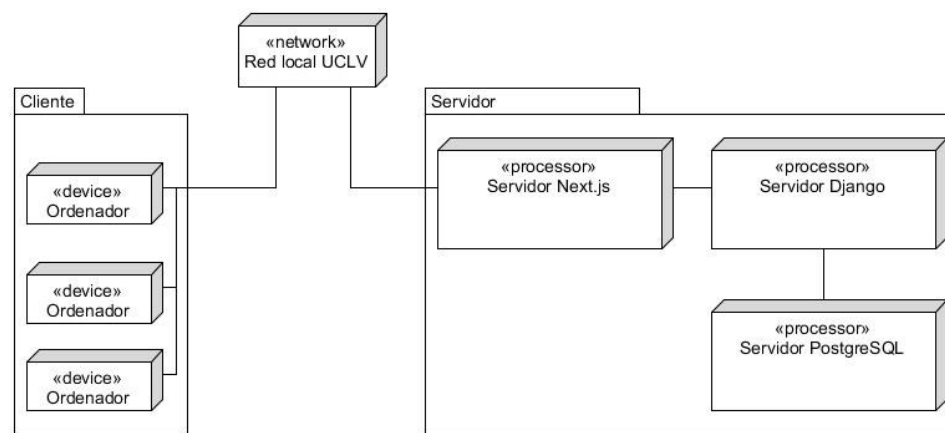


Figura 3: Diagrama de despliegue de alto nivel de la aplicación web.

La aplicación web está concebida para ser accesible desde cualquier dispositivo conectado a la red local en la cual sea desplegada (la de la UCLV, de acuerdo con su propósito), los cuales serán los clientes. El usuario, empleando un navegador en su dispositivo, accede a través de la red al servidor dedicado a atender directamente el lado del cliente, desde donde se carga el contenido implementado con Next.js en forma de un archivo HTML, de acuerdo con el funcionamiento de las SPA (véase epígrafe 1.2.1). A su vez, el servidor que provee el contenido del lado del cliente se conecta al servidor implementado con Django, que contiene la lógica del manejo de los datos, recibe las solicitudes de recuperación y modificación de los mismos, y a su vez depende de la base de datos implementada con PostgreSQL y desplegada en un servidor para el almacenamiento y gestión de los datos de la aplicación web.

Un diagrama de clases es una representación visual de la estructura estática de un sistema que visualiza sus elementos clave, como clases, atributos e interrelaciones (Alrawashdeh et al., 2024). Este tipo de diagrama se emplea en el epígrafe 2.2.2 en la descripción del servidor implementado con Django.

En los epígrafes 2.2 y 2.3 se aborda la modelación de aspectos específicos del lado del cliente o del servidor, así como particularidades de la estructura e implementación de los mismos.

2.2 Estructura e implementación del lado del servidor

El lado del servidor de la aplicación web desarrollada consiste esencialmente del servidor implementado con el marco de trabajo Django y una base de datos PostgreSQL para el almacenamiento, con la cual el anterior se conecta y a cuyas relaciones o tablas corresponden sus modelos.

2.2.1 Diseño de la base de datos, acoplamiento con Django e implementación en PostgreSQL

Los aspectos técnicos del negocio definidos en el epígrafe 1.1 demandan que la estructura de la base de datos de la aplicación web incluya cuatro entidades: una para guardar las observaciones meteorológicas diarias, dos para la información sobre las estaciones meteorológicas y unidades de cultivo, y otra para los pronósticos automáticos emitidos.

Dichas entidades fueron identificadas como *Registro*, *Estacion*, *Unidad* y *Pronostico*, respectivamente, según muestra el diagrama de la figura 4. Este tipo de diagrama se conoce como de entidad-interrelación, y es un soporte visual para el diseño de bases de datos desarrollado por Peter Chen (Uzun et al., 2018); y el mostrado corresponde al modelo conceptual, que es independiente de la implementación física subyacente (Fernández Iglesias, 2024).

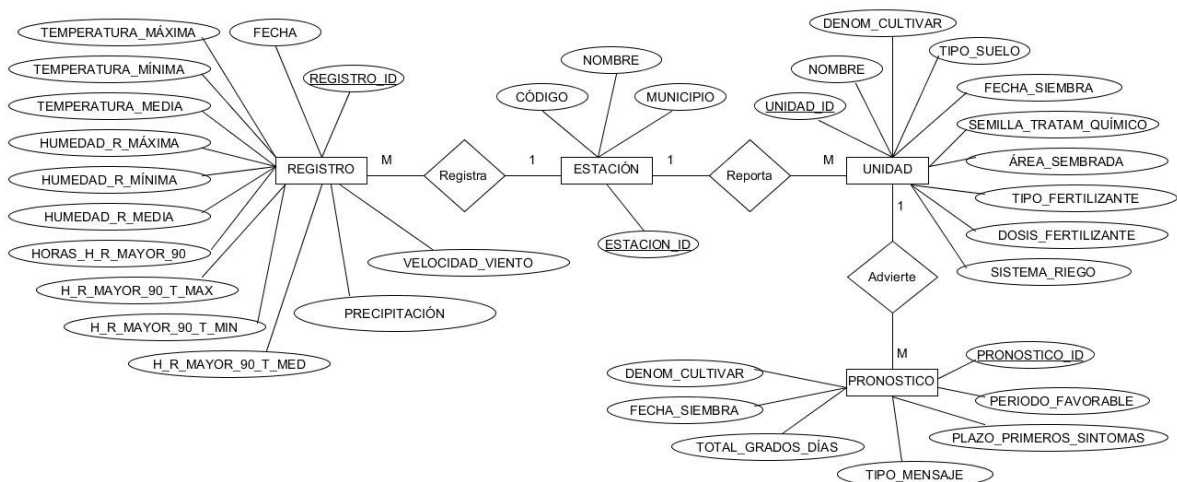


Figura 4: Esquema entidad-interrelación de la base de datos implicada por las reglas del negocio (modelo conceptual).

Cada instancia de `Registro` y de `Unidad` tiene asociada una y solo una instancia de `Estacion`, a la cual se vincula; pero cada instancia de `Estacion` puede corresponder a múltiples de `Registro` y `Unidad`. De forma análoga, cada pronóstico representado en `Pronostico` corresponderá solamente a una unidad de cultivo guardada en `Unidad`, la cual sin embargo podrá ser advertida por múltiples pronósticos.

Por facilidad de uso y en aras de una integración más natural con los modelos y serializadores de Django, se asume que la identificación de cada relación será un campo `ID` distinto de los atributos explícitamente requeridos por las reglas del negocio.

Los tipos de datos de los atributos varían entre cadenas de caracteres (`CODIGO`, `NOMBRE` y `MUNICIPIO` de `Estacion`; `NOMBRE`, `DENOM_CULTIVAR`, `TIPO_SUELO`, `TIPO_FERTILIZANTE` y `SISTEMA_RIEGO` de `Unidad`; y `PLAZO_FAVORABLE`, `PLAZO_PRIMEROS_SINTOMAS` y `TIPO_MENSAJE` de `Pronostico`), fechas (`FECHA` de `Registro` y `FECHA_SIEMBRA` de `Unidad`), un campo booleano (`SEMILLA_TRATAM_QUIMICO`), grandes enteros (los `IDS`) y números reales (el resto). `PLAZO_FAVORABLE` y `PLAZO_PRIMEROS_SINTOMAS` de `Pronostico`) fueron concebidos como cadenas de caracteres formadas por la unión de la representación textual de las fechas de inicio y fin de los períodos respectivos.

Las instancias de `Estacion` se ordenan por `CODIGO`, `MUNICIPIO` y `NOMBRE`, en ese orden de prioridad; las de `Registro`, por `FECHA` en orden descendente; las de `Unidad`, por `NOMBRE`; y las de `Pronostico`, por `FECHA_SIEMBRA`. Se impone las restricciones especiales de que las instancias de `Registro` tendrán que tener pares de `ESTACION` y `FECHA` únicos, y las de `Pronostico` tendrán que tener tuplas únicas de valores de `UNIDAD`, `FECHA_SIEMBRA`, `DENOM_CULTIVAR` y `TIPO_MENSAJE`.

El vínculo entre Django y la implementación PostgreSQL de la base de datos se define en el contenido del archivo `settings.py` con el fragmento de código mostrado en la figura 5.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'datamaiz',
        'USER': 'django',
        'PASSWORD': ' ',
        'HOST': 'localhost',
        'PORT': '5433',
    }
}
```

Figura 5: Configuración de conexión a la base de datos en Django.

Mediante las *migrations* (migraciones) las creaciones y modificaciones definidas en Django, así como la información guardada y actualizada mediante el mismo, se aplican a la base de datos PostgreSQL, los detalles de implementación se resuelven de forma automática, y de la misma manera se genera código SQL que especifica la estructura y el funcionamiento de la base de datos. La figura 6 muestra un diagrama de entidad-interrelación de las relaciones descritas, correspondiente al nivel lógico, más detallado que el conceptual.

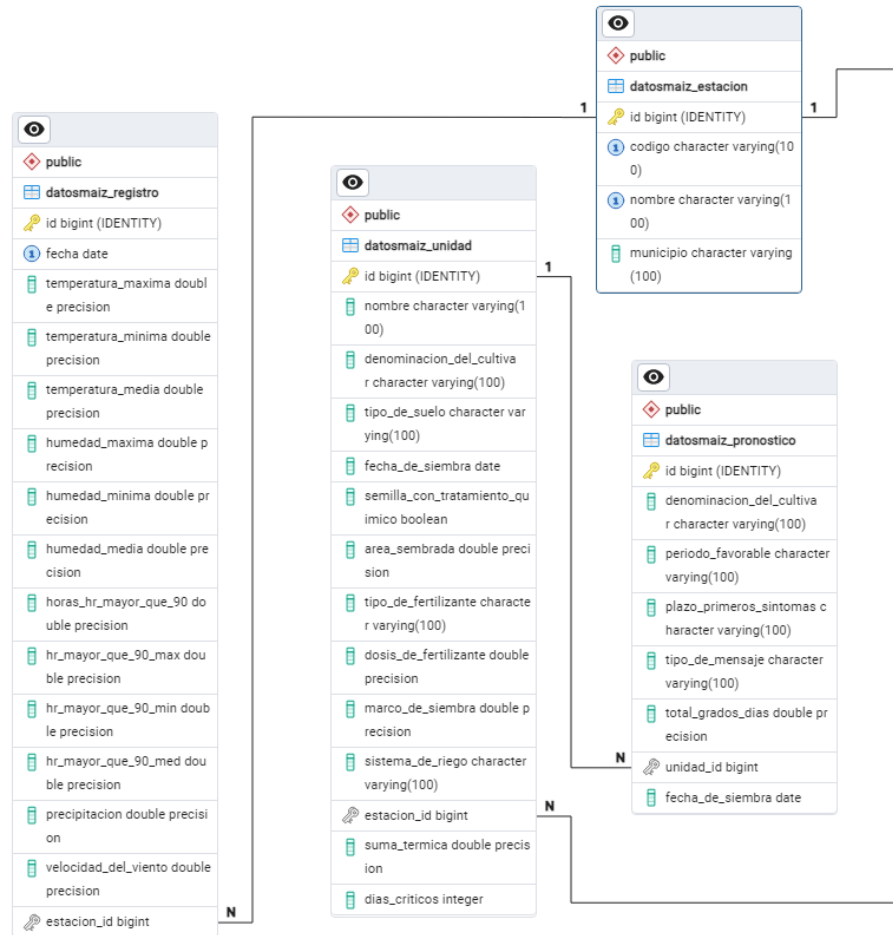


Figura 6: Diagrama de entidad-interrelación lógico de las relaciones creadas.

La base de datos recibe tablas adicionales relacionadas con el sistema de autenticación y usuarios, así como otros aspectos técnicos de Django. El sistema de tablas resultante, visto desde el explorador de la herramienta de trabajo con PostgreSQL de interfaz gráfica PgAdmin 4, aparece en la figura 7; y la figura 8 muestra lógicamente las relaciones que no están directamente determinadas por las reglas del negocio que motivan el desarrollo de la aplicación web, pero que

resultan necesarias para el funcionamiento de la autenticación, los permisos, migraciones y administración de Django.

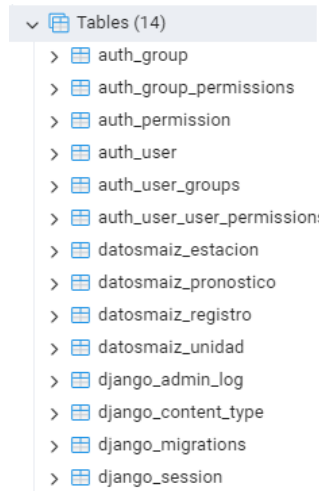


Figura 7: Tablas creadas por las migraciones de Django en la base de datos PostgreSQL.



Figura 8: Tablas adicionales requeridas para el funcionamiento de la autenticación, permisos, migraciones y administración.

2.2.2 Implementación del servidor Django

Desde el punto de vista del sistema de archivos, la implementación en Django del lado del servidor de la aplicación web contiene en su directorio raíz el fichero `manage.py`, mediante el cual se administra el proyecto con comandos de Python. Dos subcarpetas de este directorio agrupan el resto del contenido: `datamaizbackend/`, con los ficheros que controlan de forma global el servidor, y `datosmaiz/`, con la implementación funcional específica para el servidor, como una *app* de Django. Esta no debe ser confundida con el concepto de aplicación web, sino que consiste en un paquete de Python que provee un conjunto de facilidades y puede ser reusado en varios proyectos. Las *apps* incluyen combinaciones de modelos, vistas, plantillas, URLs, entre otros aspectos (Django Documentation, 2024). La figura 9 muestra el árbol de archivos.

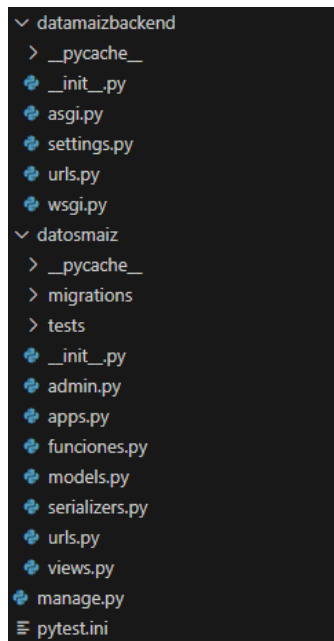


Figura 9: Árbol de archivos de la implementación del servidor Django.

Dentro de `datamaizbackend/`, los archivos más significativos son `settings.py` y `urls.py`. El primero contiene configuraciones que determinan el funcionamiento de todo el lado del servidor, como la base de datos a usar y sus ajustes (tal como se muestra en la figura 5), los *hosts* o sitios autorizados a acceder (es decir, la dirección web donde se despliegue el lado del cliente, que durante el desarrollo es `http://localhost:3000` o `http://127.0.0.1:3000`, según la configuración local típica), las *apps* instaladas (donde se registra `datosmaiz`, además de otras como

las incluidas por defecto en Django para el manejo de la autenticación y la administración, o las que permiten la integración de Django REST Framework), entre otras.

El segundo especifica las direcciones URL (véase el epígrafe 1.2.3) que se ofrecerán al cliente; sin embargo, este aspecto es asumido con mayor detalle por el archivo del mismo nombre ubicado en `datosmaiz/`, todas las rutas del cual son incluidas en el de `datamaizbackend/` mediante la función `include` importada del paquete `django.urls`. Estas rutas serán las vías mediante las cuales el lado del cliente accederá al servidor.

Dentro de `datosmaiz/`, el archivo `models.py` define los modelos (concepto análogo al de relación en bases de datos relacionales) como clases de Python. La figura 10 muestra el diagrama de clases de los modelos definidos, y la figura 11 ejemplifica la implementación de los mismos con el caso de `Pronostico`.

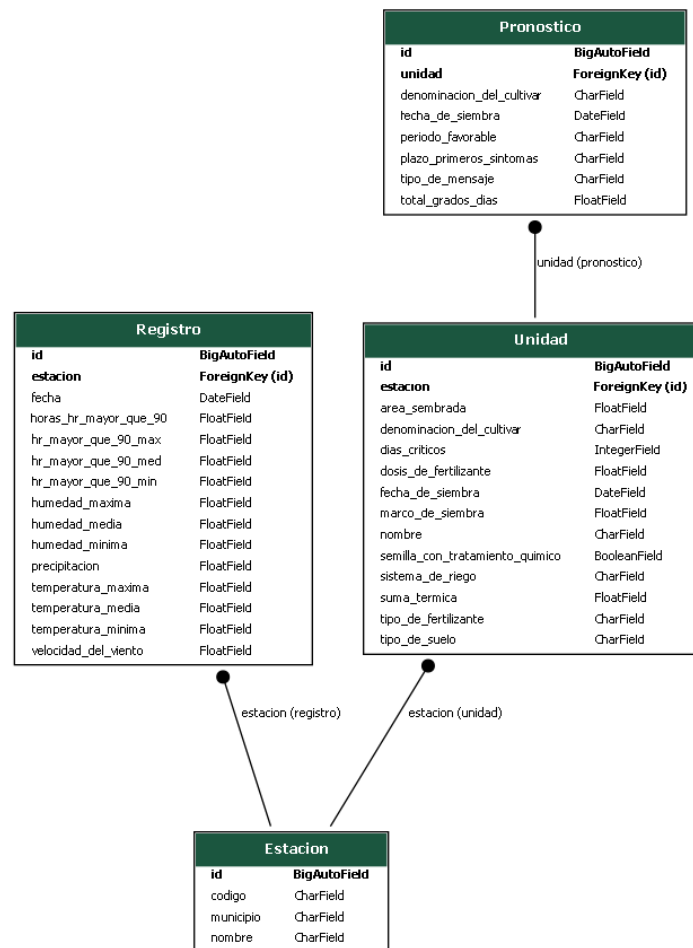


Figura 10: Diagrama de clases de los modelos de Django de `datosmaiz`.

```

class Pronostico(models.Model):
    unidad = models.ForeignKey(Unidad, on_delete=models.CASCADE, to_field="id", default=1)
    fecha_de_siembra = models.DateField(null=True)
    denominacion_del_cultivar = models.CharField(max_length=100)
    periodo_favorable = models.CharField(max_length=100)
    plazo_primeros_sintomas = models.CharField(max_length=100)
    tipo_de_mensaje = models.CharField(max_length=100)
    total_grados_dias = models.FloatField()

    class Meta:
        ordering = ['-fecha_de_siembra']
        unique_together = ['unidad', 'fecha_de_siembra', 'denominacion_del_cultivar', 'tipo_de_mensaje']

```

Figura 11: Implementación del modelo Pronostico en models.py.

En Unidad se agregó los atributos `dias_criticos` y `suma_termica`, que aunque no están explícitamente requeridos en las reglas del negocio, son convenientes para la determinación de los pronósticos que deben ser emitidos.

En `serializers.py` se especifica la manera en que las instancias de los modelos serán representadas para su envío al cliente, o cómo los datos recibidos del cliente serán convertidos en nuevas instancias de un modelo del servidor.

El archivo `views.py` define las formas en las que el servidor se comunicará con el cliente, qué solicitudes aceptará, cómo las procesará, qué modelos consultará o modificará, y qué respuestas devolverá. La implementación realizada incluye las vistas `estaciones_list(request)`, `registros_list(request)`, `unidades_list(request)` y `pronosticos_list(request)` para recuperar y retornar al cliente las instancias de sus respectivos modelos, así como agregar nuevas (excepto en la última, ya que los pronósticos se emiten automáticamente); `estacion(request, pk)`, `registro(request, pk)`, `unidad(request, pk)` y `pronostico(request, pk)` para acceder a una instancia específica (identificada mediante su llave primaria) y eliminarla; y `registros_de_una_estacion(request, pk)` para recuperar y retornar las instancias de Registro que corresponden a una instancia particular de Estacion. Todas estas fueron implementadas como vistas basadas en funciones, según las características brindadas por Django REST Framework, el cual provee un conjunto de decoradores simples que envuelven las funciones para asegurarse de que reciben una instancia de Request (tipo de objeto de solicitud de Django REST) y permitir que retornen una respuesta Response (Django REST Framework Documentation, 2024). Sin embargo, la vista `RegisterView` se implementó como basada en clase, por comodidad, concisamente definida mediante las facilidades de Django REST, para participar en

la manipulación de la lógica de autenticación; y en `urls.py` se asocia a la ruta correspondiente como `RegisterView.as_view()`.

`funciones.py` contiene funciones con lógica necesaria para la emisión de pronósticos, separadas de `views.py` para conseguir una mejor organización, modularización y claridad en el código. De especial interés resulta la función `emitir_pronosticos()`, cuya implementación se muestra en la figura 12.

```
def emitir_pronosticos():
    unidades = Unidat.objects.all()
    for unidad in unidades:
        fecha_700 = calcular_suma_termica(unidad)
        determinar_dias_criticos(unidad, fecha_700)
        if unidad.dias_criticos >= 5:
            registros = Registro.objects.filter(estacion=unidad.estacion, fecha__gte=fecha_700).order_by('fecha')
            ultimo_dia_critico = registros[len(registros) - 1].fecha
            fecha_inicio_periodo_favorable = (ultimo_dia_critico - timedelta(days=10)).strftime('%d/%m/%Y')
            fecha_fin_periodo_favorable = ultimo_dia_critico.strftime('%d/%m/%Y')
            periodo_favorable = f"{fecha_inicio_periodo_favorable}-{fecha_fin_periodo_favorable}"
            fecha_inicio_plazo_sintomas = ultimo_dia_critico.strftime('%d/%m/%Y')
            fecha_fin_plazo_sintomas = (ultimo_dia_critico + timedelta(days=10)).strftime('%d/%m/%Y')
            plazo_primeros_sintomas = f"{fecha_inicio_plazo_sintomas}-{fecha_fin_plazo_sintomas}"
            tipo_de_mensaje = "Alerta" if unidad.dias_criticos == 5 else "Peligro máximo"
            pronostico = Pronostico(
                unidad=unidad,
                fecha_de_siembra=unidad.fecha_de_siembra,
                denominacion_del_cultivar=unidad.denominacion_del_cultivar,
                periodo_favorable=periodo_favorable,
                plazo_primeros_sintomas=plazo_primeros_sintomas,
                tipo_de_mensaje=tipo_de_mensaje,
                total_grados_dias=unidad.suma_termica
            )
            try:
                pronostico.save()
            except Exception as e:
                pass
```

Figura 12: Implementación de la función `emitir_pronosticos()`.

`urls.py` de `datosmaiz/` registra los puntos de entrada de la API del lado del servidor en forma de rutas URL, mediante las cuales el cliente accederá a las vistas. Se implementó rutas para la recuperación de listas completas de las instancias de cada uno de los cuatro modelos creados, así como para el acceso a instancias individuales identificadas por sus llaves primarias (captadas como parámetros enteros incluidos en las rutas), y una para la obtención de los registros de una estación específica, vinculada a la vista `registros_de_una_estacion(request, pk)`. Además, se incluyeron rutas vinculadas a las vistas `TokenObtainPairView.as_view()` y `TokenRefreshView.as_view()`, importadas del paquete `rest_framework_simplejwt.views`, para el manejo de *tokens*.

Para el almacenamiento de credenciales de usuario se emplea la clase `User`, importada del paquete `django.contrib.auth.models`. En `serializers.py` se define la estructura que, a vista del cliente, tendrán los datos del usuario (nombre y contraseña).

Para el manejo de tokens, en `settings.py` se incluyó `rest_framework_simplejwt` entre las *apps* instaladas, así como las configuraciones adecuadas de `REST_FRAMEWORK` y `SIMPLE_JWT` para el uso de JWT en Django (véase el epígrafe 1.7).

Para la determinación de permisos se atiende al método incluido en la solicitud del cliente (véase el epígrafe 1.2.3). En `views.py` las vistas tienen asociadas clases de permisos: `AllowAny` permite el uso de la vista sin necesidad de estar autenticado, mientras que `IsAuthenticatedOrReadOnly` permite las solicitudes `GET` a cualquiera, pero solo procede a aceptar solicitudes que modifican los datos (como `POST` o `DELETE`) a usuarios autenticados.

2.3 Estructura e implementación del lado del cliente

2.3.1 Rutas y páginas

La implementación del lado del cliente sigue la forma típica de una aplicación hecha con `Next.js` usando la técnica *AppRouter*, en la cual la interfaz gráfica se organiza en vistas conocidas como “páginas”. Estas son solo una abstracción de la distribución del contenido, y no deben ser confundidas con el concepto de *page* usado en el epígrafe 1.2.1, de acuerdo con el cual la aplicación propuesta se clasifica como *Single-Page*; pues en el contexto de la clasificación de las aplicaciones web, se entiende por *page* el documento en HTML generado por el código, cargado de una sola vez al inicio; mientras que en la presentación del lado del cliente, las páginas son divisiones virtuales que agrupan el contenido de dicho archivo de tal forma que la navegación sea más fácil e intuitiva. El usuario accede a las diferentes páginas usando enlaces a las rutas que las identifican, y que en el sistema de archivos de código fuente se implementan mediante subcarpetas del directorio `src/app/`. El contenido de cada una se define en un archivo llamado `page.js`, y se aplica una disposición uniforme compartida por todas las rutas subordinadas de una carpeta mediante un archivo `layout.js`. Los archivos ubicados en el directorio raíz `src/app/` definen la página inicial de la interfaz.

La interfaz de usuario implementada cuenta con cinco páginas: la inicial, con una presentación breve y básica del software y sus funcionalidades, así como el nombre de producción dado al software, “*Datamaíz*”; `app/rutas/registros/` para la consulta, adición y eliminación de observaciones meteorológicas, así como el acceso a información estadística adicional sobre las observaciones almacenadas; `app/rutas/estaciones/` para el manejo de datos sobre las estaciones meteorológicas; `app/rutas/unidades/` para las unidades de cultivo, y `app/rutas/pronosticos/` para la consulta y eliminación de pronósticos. La figura 13 muestra el árbol de rutas. Las figuras 14 y 15 contienen capturas de la página inicial y la de registros. Las restantes páginas lucen de forma similar.

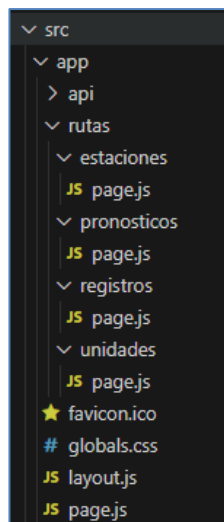


Figura 13: Árbol de rutas de páginas del lado del cliente.



Figura 14: Captura de la página inicial.

Estación	Fecha	Temperaturas			Humedad relativa			Período de HR ≥ 90%			Precipitación mm	Velocidad del viento (m/s)	
		Mínima °C	Media °C	Máxima °C	Mínima %	Media %	Máxima %	Horas ≥ 90%	Mínima °C	Media °C			Máxima °C
La Loma	2024-11-06	18.5	16.5	17.5	97	77	87	9.7	18.5	16.5	17.5	28	34

Figura 15: Captura de la página de registros.

2.3.2 Componentes

En términos de implementación, las estructuras básicas del lado del cliente de la aplicación propuesta son los componentes de React usados por el marco de trabajo Next.js. Estos fueron definidos en archivos separados, y frecuentemente se anidan de manera que un componente es importado y usado como parte de otro. Esta forma de programación modulariza el código y lo mantiene organizado, conciso y legible. Los componentes implementados fueron agrupados en el directorio `src/components/`, como se muestra en la figura 16. A continuación se particularizará en algunos de ellos.

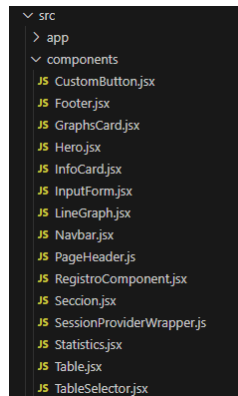


Figura 16: Árbol de archivos de la carpeta `components`.

`<Navbar/>` implementa la barra de navegación que aparece en la parte superior de la ventana de la aplicación web en todas las páginas de la misma. Contiene el ícono de la aplicación y una lista de botones con enlaces a las diferentes páginas. Está configurada para adaptarse a distintos tamaños de pantalla del dispositivo del cliente.

Uno de los componentes más significativos de la interfaz gráfica es `<Table/>`, que implementa una tabla para los datos que han de ser mostrados en las páginas de registro de observaciones meteorológicas, estaciones, unidades de cultivo y pronósticos. La cantidad de columnas, subcolumnas y sus nombres, así como otros aspectos específicos, le son pasados como *props* en el componente padre que engloba la instancia particular de `<Table/>`. Soporta los encabezados de columna de dos niveles, y se permite también la selección de filas individuales para su eliminación. El estilo usado (definido mediante clases de utilidad de Tailwind) es visualmente coherente con el diseño del resto de la interfaz gráfica. El núcleo del componente es una etiqueta HTML `<table>` adecuadamente modificada mediante las facilidades de JSX, como se muestra en la figura 17.

```


| {formattedColumns.map((column) => ( <th key={column.title} colspan={column.subColumns ? column.subColumns.length : 1} className="border-r-2 border-l-2 border-maiz-dark p-2"> {column.title} </th> ))}                                                                                                              |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| {compositeHeader && ( <tr> {formattedColumns.map((column) => ( column.subColumns ? column.subColumns.map((subColumn) => ( <th key={subColumn} className="border-r-2 border-l-2 border-maiz-dark p-2"> {subColumn} </th> )) : <th key={column.title} className="border-r-2 border-l-2 border-maiz-dark p-2"></th> )} |
| {data && ( <tbody> {data.map((row) => ( <tr> key={row.id} onClick={() => handleRowClick(row.id)}                                                                                                                                                                                                                    |


```

Figura 17: Fragmento de código del componente `<Table/>`.

`<TableSelector/>` permite la aplicación de filtros relativamente simples a las instancias de los modelos correspondientes en el lado del servidor que serán mostradas en la tabla. Fue diseñado específicamente para permitir la selección de la estación meteorológica y el año deseados para

```

<select>
  value={opcionSeleccionada}
  onChange={handleOptionChange}
  className="w-full p-2 border-2 border-maiz rounded-xl"
>
  <option value={0}>Todas</option>
  {Array.isArray(opcionesData) && opcionesData.map((opcion) => (
    <option key={opcion.id} value={opcion.id}>{opcion.nombre}</option>
  ))}
</select>
<select>
  value={yearSeleccionado}
  onChange={handleYearChange}
  className="w-full p-2 border-2 border-maiz rounded-xl"
>
  {Array.from({ length: 10 }, (_, i) => new Date().getFullYear() - i).map((year) => (
    <option key={year} value={year}>{year}</option>
  ))}
</select>

```

Figura 18: Fragmento de código del componente `<TableSelector/>`.

las observaciones meteorológicas consultadas. Se basa en el uso de las etiquetas HTML `<select>` y `<option>`, empleadas y ajustadas en el fragmento de código del componente que se muestra en la figura 18.

`<InputForm/>` implementa formularios para la introducción de información al sistema en forma de nuevas filas en las tablas, que representan instancias adicionales a los modelos del lado del servidor. Presenta al usuario los campos correspondientes (los cuales, junto con sus tipos de datos y otras particularidades, le son pasados como *props* en el componente padre que engloba la instancia particular de `<InputForm/>`). Se basa en el uso de la etiqueta HTML `<form>`, un fragmento de cuyo código aparece en la figura 19. La figura 20 ilustra el aspecto visual de un formulario generado por `<InputForm/>`, y la 21 muestra su uso en el código de la página de datos de las unidades de cultivo.

```
<form className="space-y-3 mb-4" onSubmit={handleSubmit}>
  {formFields.map((field) => (
    <div
      key={field.name}
      className="border border-gray-200 p-2 w-full rounded-xl text-gray-400"
    >
      {field.type === 'select' ? (
        <select
          name={field.name}
          value={formData[field.name]}
          onChange={handleChange}
          className="w-100%"
        >
          <option value="">{field.placeholder}</option>
          {options[field.name].map((opcion) => (
            <option
              key={opcion.id}
              value={opcion.id}
              className="w-100%"
            >
              {opcion.nombre}
            </option>
          ))}
        </select>
      ) : field.type === 'checkbox' ? (
        <div className="flex items-center">
```

Figura 19: Fragmento de código del componente `<InputForm/>` .

Figura 20: Vista del formulario generado por el componente `<InputForm/>` en la interfaz gráfica.

```

<InputForm
  formFields={
    [
      { name: 'nombre', type: 'text', placeholder: 'Nombre de la entidad' },
      { name: 'estacion_codigo', type: 'select', placeholder: 'Estación' },
      { name: 'denominacion_del_cultivar', type: 'text', placeholder: 'Denominación del cultivar' },
      { name: 'tipo_de_suelo', type: 'text', placeholder: 'Tipo de suelo' },
      { name: 'fecha_de_siembra', type: 'date', placeholder: 'Fecha de siembra' },
      { name: 'semilla_con_tratamiento_quimico', type: 'checkbox', placeholder: 'Semilla con tratamiento químico' },
      { name: 'area_semada', type: 'number', placeholder: 'Área sembrada (ha)' },
      { name: 'tipo_de_fertilizante', type: 'text', placeholder: 'Tipo de fertilizante' },
      { name: 'dosis_de_fertilizante', type: 'number', placeholder: 'Dosis de fertilizante' },
      { name: 'marco_de_siembra', type: 'number', placeholder: 'Marco de siembra' },
      { name: 'sistema_de_riego', type: 'text', placeholder: 'Sistema de riego' },
    ]
  }
  fetchUrls={
    [
      { name: 'estacion_codigo', url: 'estaciones/' },
    ]
  }
  postUrl='unidades/'
  buttonText='Registrar nueva unidad de cultivo'
  onFormSubmit={fetchDataAsync}
/>

```

Figura 21: Uso del componente `<InputForm/>` en la página de datos de unidades de cultivo .

`<Statistics/>` es un componente orientado a la función de calcular y mostrar estadísticas sobre las observaciones meteorológicas registradas en la última semana de una estación. Para ello se auxilia de la función `computeStatistics`, importada del archivo `src/services/statistics.js`, así como del componente `<Table/>`. La figura 22 muestra el código JSX producido por `<Statistics/>`, y la figura 23 es una captura de la sección de interfaz gráfica generada por el componente.

```

<div className="flex flex-col justify-center w-full">
  <Seccion
    title={"Resumen estadístico de los últimos 7 días de la estación seleccionada"}
    content={
      <div className="w-full overflow-x-auto">
        <Table
          columns={columns}
          formattedColumns={formattedColumns}
          data={statistics}
          compositeHeader={true}
        />
      </div>
    }
  />
</div>

```

Figura 22: Código JSX producido por el componente `<Statistics/>` .

Resumen estadístico de los últimos 7 días de la estación seleccionada

Medida estadística	Temperaturas			Humedad relativa				Periodo de HR ≥ 90%			Precipitación mm	Velocidad del viento (m/s)
	Mínima °C	Media °C	Máxima °C	Mínima %	Media %	Máxima %	Horas ≥ 90%	Mínima °C	Media °C	Máxima °C		
Promedio	19.79	17.89	18.84	94.62	73.78	84.20	8.74	19.79	17.89	18.84	27.88	31.69
Máximo	23.00	19.00	20.50	97.00	77.00	87.00	9.70	23.00	19.00	20.50	29.00	37.00
Mínimo	18.20	16.50	17.50	92.00	72.00	82.00	7.50	18.20	16.50	17.50	26.80	26.00

Figura 23: Sección de la interfaz gráfica producida por el componente `<Statistics/>` .

Los componentes `<Hero/>`, `<RegistroComponent/>` y `<Seccion/>` sirven para agrupar otros componentes, y su función es esencialmente organizativa. `<InfoCard/>` y `<PageHeader/>` ofrecen marcos de presentación visual para la información; el primero estiliza los títulos de las páginas, y el segundo agrupa en tarjetas virtuales las imágenes y textos de la interfaz gráfica. `<GraphsCard/>` agrupa los gráficos de línea implementados mediante `<LineGraph/>`, que permiten al usuario visualizar las tendencias de las variables meteorológicas mostradas en la página de observaciones.

2.3.3 Implementación de la autenticación

La configuración de autenticación mediante NextAuth se define en el archivo `src/app/api/auth/[...nextauth]/route.js`, que incluye especificaciones para la definición de credenciales y renovación de *tokens*, y en `src/middleware.js`, relacionado con la accesibilidad de las rutas para usuarios no autenticados. Las credenciales consisten de un par formado por un nombre de usuario y una contraseña.

En el archivo `layout.js` principal, que define arreglos uniformes a través de las diferentes páginas de la aplicación web, se engloba el contenido dentro del componente `<SessionProviderWrapper>`, que a su vez importa `<SessionProvider>` del paquete `next-auth/react`, para la provisión de sesiones de usuario.

En la barra de navegación, abordada en el epígrafe anterior y también incluida en `layout.js`, se incluye una pestaña como se muestra en la figura 24, para permitir al usuario autenticarse si no lo ha hecho, o cerrar su sesión si ya está autenticado. La lógica de autenticación se maneja en el fragmento de código de la barra de navegación que aparece en la figura 25.

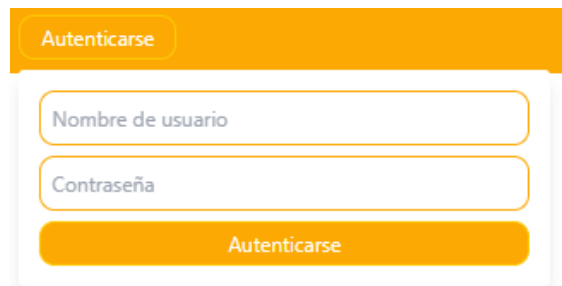


Figura 24: Pestaña de la barra de navegación para la autenticación .

```

const handleSignIn = async (e) => {
  e.preventDefault();
  const result = await signIn('credentials', {
    redirect: false,
    username: e.target.username.value,
    password: e.target.password.value,
  });
  if (!result.error) {
    setShowDropdown(false);
    window.alert("Usted ha iniciado su sesión exitosamente.")
    console.log(session)
  } else {
    console.error('Authentication error:', result.error);
    window.alert("No se pudo autenticar. Compruebe que las credenciales sean correctas.")
  }
};

```

Figura 25: Fragmento de código del componente <Navbar/> de lógica de autenticación .

2.4 Conclusiones parciales del capítulo

En el capítulo se describió la modelación y desarrollo de la aplicación web en forma global, así como en sus partes separadas del lado del cliente y del servidor, en aspectos funcionales y estructurales, y a diferentes niveles de abstracción, desde sus diagramas correspondientes en el diseño gráfico y la apariencia visual de la interfaz gráfica hasta los sistemas de archivos implementados y fragmentos relevantes del código; y se concluye que el diseño descrito por los esquemas usados, así como los archivos, clases, funciones, componentes y configuraciones que se codificó, resultan adecuados en aras de que el software desarrollado cumpla las funcionalidades definidas en los requisitos.

CAPÍTULO 3. PRUEBAS REALIZADAS AL SOFTWARE

En este capítulo se aborda el proceso de prueba de la aplicación web propuesta. Se clasifica y explica las pruebas realizables a un software, se describe las empleadas en el caso de la aplicación, y se expone las herramientas usadas para llevar a cabo las mismas.

3.1 Aspectos generales de las pruebas de software

El probado o testeado del software (*software testing* en inglés) es el proceso de evaluación de un programa que busca comprobar su corrección, confiabilidad y seguridad con el objetivo de asegurarse de que satisface el propósito para el cual fue desarrollado. Típicamente, se asume que en el software creado existen errores o defectos (lo cual en la práctica resulta inevitable, a menos que el programa sea exageradamente simple) y se trata de encontrarlos para depurarlos.

3.1.1 Propósito

Esta etapa del ciclo de vida del desarrollo de software es notablemente importante. En primer lugar, disminuye de forma drástica la probabilidad de que los usuarios finales del producto informático sufran los perjuicios que pueden resultar de la manifestación en uso real de errores no detectados por los desarrolladores. El daño resultante de tales faltas puede variar entre molestias que empobrecen la experiencia de usuario y accidentes de dimensiones catastróficas, en dependencia de la confianza puesta sobre el sistema creado y del área de actividad para el que se produjo. Además, la solución temprana de problemas en el software normalmente ahorra una cantidad significativa de recursos a los desarrolladores, pues es común que los errores acumulados sean más costosos mientras más se progresa en los proyectos; de forma que un proceso diligente de probado tiene el potencial de evitar dificultades mayores y compensa bien el esfuerzo dedicado. Este campo ha recibido considerable atención durante la historia del desarrollo de software, y la investigación se ha enfocado ampliamente en el diseño de técnicas de prueba y la validación de su eficacia en contextos reales de producción (Kassab et al., 2017; Umar, 2020).

3.1.2 Metodologías

Las pruebas realizadas a un software pueden ser categorizadas a grandes rasgos de la siguiente forma:

- **Estáticas:** El código es inspeccionado sin ejecución, mediante el análisis simbólico, chequeo de modelos y técnicas similares.
- **Dinámicas:** Implican la ejecución del código y trata con combinaciones de entradas, uso de procedimientos de prueba determinados estructuralmente, automatización de generación de ambientes de prueba, y métodos relacionados.

Las pruebas dinámicas se subdividen a su vez en dos categorías:

- **Pruebas de caja negra:** La estructura e implementación internas del software no son conocidas o tenidas en cuenta por el autor de la prueba. Los casos de prueba son contruidos sobre la base de la especificación de requisitos, y el énfasis recae en la evaluación de aspectos funcionales del programa mediante casos de uso y la conservación de la integridad de la información externa.
- **Pruebas de caja blanca:** La estructura e implementación interna del software es tenida en cuenta por el autor de la prueba. Se requiere el conocimiento del código fuente. La selección o creación de casos de prueba se basa en la implementación de las entidades lógicas que componen el programa.

La literatura reciente también teoriza sobre pruebas de “caja gris”, que constituyen un término medio entre los extremos anteriores y combinan aspectos de ambos (Anwar and Kar, 2019).

En cuanto a los niveles de alcance de las pruebas, se distingue los siguientes cuatro:

- **Unitarias:** Enfatizan unidades individuales o módulos vistos como elementos aislados. Tratan las mínimas porciones que se pueden probar del software para verificar su funcionalidad contra su especificación. Son realizables tan pronto como una nueva fracción del código se implementa.
- **De integración:** Involucran la prueba de dos o más unidades combinadas que deben funcionar en conjunto para asegurar un flujo de control y datos libre de errores. Son realizables una vez que se añade nuevas unidades vinculadas funcionalmente con las ya implementadas.
- **De sistema:** Prueban un software completo para chequear el cumplimiento de los requisitos especificados. Trata con la interacción general de las partes para asegurar la cooperación armoniosa de todos los módulos sin error. Son realizables una vez que todos los componentes del software han sido implementados.

- **De aceptación:** Comprueban de forma directa que el software satisface los requisitos de los clientes a través de la exposición del producto a combinaciones de desarrolladores y usuarios o a estos últimos solamente. Son realizables cuando el software está operativamente listo (Umar, 2020).

3.2 Pruebas realizadas a la aplicación web propuesta

A continuación se describe un conjunto de pruebas controladas y descritas realizadas al software en la etapa final de su desarrollo, sin perjuicio de los disímiles experimentos informales usados durante el proceso de implementación para comprobar el funcionamiento de las partes del código mientras iban siendo creadas y modificadas, y que permitió su perfeccionamiento al descubrir errores de código que fueron progresivamente solucionados durante la evolución del software.

3.2.1 Pruebas concernientes al lado del servidor

Para la realización automática de pruebas en la aplicación Django que constituye el núcleo del lado del servidor, se usó Pytest, un marco de pruebas que permite escribir *tests* concisos y legibles, y es escalable para el soporte de testeo funcional de aplicaciones y bibliotecas (Pytest Documentation, 2024).

Se creó el directorio `tests/` dentro de `datosmaiz/`, y dentro del mismo se implementó los archivos `test_models.py`, `test_serializers.py`, `test_urls.py` y `test_views.py` para agrupar las pruebas unitarias y de integración escritas en Python enfocadas particularmente en modelos, serializadores, URLs y vistas, respectivamente. La figura 26 muestra una prueba escrita para comprobar el funcionamiento del régimen de permisos en la vista `estaciones_list(request)`, y la figura 27 corresponde al resultado de la corrida de las pruebas automáticas con Pytest.

```
@pytest.mark.django_db
def test_estaciones_unauthenticated():
    client = APIClient()
    url = reverse('estaciones')
    data = {'nombre': 'CSS Mala', 'codigo': '077', 'municipio': 'Corralillo'}
    response = client.post(url, data)
    assert response.status_code == status.HTTP_401_UNAUTHORIZED
```

Figura 26: Prueba escrita con Pytest para la vista `estaciones_list(request)`.

```
===== test session starts =====
platform win32 -- Python 3.12.4, pytest-8.3.3, pluggy-1.5.0
django: version: 5.0.6, settings: datamaizbackend.settings (from ini)
rootdir: C:\Alain Daniel\Universidad\Tesis\Websites\Django projects\maiz-backend\datamaizbackend
configfile: pytest.ini
plugins: django-4.9.0
collected 16 items

datosmaiz\tests\test_models.py ..
[ 12%]
datosmaiz\tests\test_serializers.py ....
[ 37%]

datosmaiz\tests\test_models.py ..
[ 12%]
datosmaiz\tests\test_serializers.py ....
[ 37%]
datosmaiz\tests\test_views.py .....
[ 75%]
datosmaiz\tests\test_urls.py ....
[100%]

===== 16 passed in 10.28s =====
PS C:\Alain Daniel\Universidad\Tesis\Websites\Django projects\maiz-backend\datamaizbackend> |
```

Figura 27: Resultado de la ejecución de las pruebas realizadas con Pytest.

También se realizó pruebas manuales mediante la interfaz navegable de la API provista por Django REST Framework. Se destaca la introducción de una secuencia de observaciones meteorológicas ficticias intencionalmente construidas de tal forma que cumplan las condiciones críticas y desencadenen la emisión de pronósticos de peligro para las unidades de cultivo asociadas a determinada estación, para probar la integración compleja entre varias partes del código (en particular, de la función `emitir_pronosticos()`). Las pruebas revelaron errores en la implementación de la función, así como de `calcular_suma_termica(unidad)` y de `determinar_dias_criticos(unidad)`, que fueron oportunamente corregidos; y luego los pronósticos se generaron correctamente.

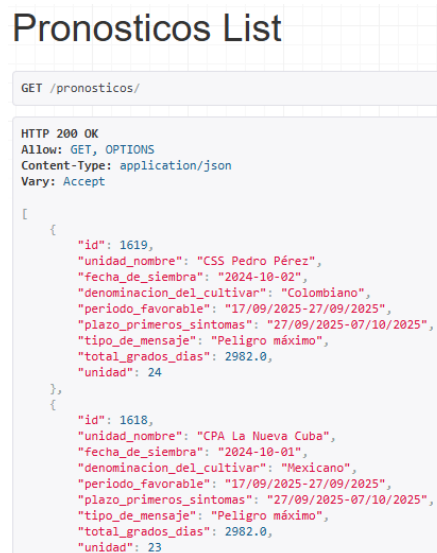


Figura 28: Captura de la API navegable de Django REST Framework que muestra pronósticos emitidos en las pruebas.

3.2.2 Pruebas concernientes al lado del cliente

Para probar la implementación del lado del cliente, se consideró la posibilidad de usar herramientas populares de software para la codificación de pruebas automáticas en este tipo de aplicaciones web, como Jest o React Testing Library. Sin embargo, se llegó a la opinión de que la realización manual de pruebas de caja negra desde el punto de vista simulado de un usuario de la interfaz gráfica es más adecuada en la práctica para la aplicación web propuesta, dados su propósito, contexto de uso, nivel de complejidad y estructura.

Las pruebas unitarias y de integración consistieron en la interacción con componentes aislados y en grupos para verificar que las partes de la interfaz gráfica generada por los mismos fueran correctas. En la etapa final del desarrollo del software, las pruebas indicaron al desarrollador la conveniencia de hacer ajustes menores a los espacios entre componentes de las páginas por motivos estéticos, corregir el formato del comodín mostrado al usuario al interactuar con los campos de fecha en el formulario de entrada de datos, y, más notablemente, revelaron la necesidad de modificar el código del componente `<RegistroComponent/>` para lograr que el resumen estadístico y los gráficos de línea se muestren con los datos correctos en la página de observaciones meteorológicas, solamente cuando una estación específica esté seleccionada, y no cuando se esté examinando los datos de todas las estaciones en conjunto.

También se percibió que la experiencia de usuario en el trabajo con los pronósticos emitidos sería mejor si estos estuvieran ordenados por fecha de siembra y no por unidad como estaba definido antes. Además, se notó que debía alertarse que un pronóstico no pudo ser eliminado por ser aún relevante cuando la unidad de cultivo asociada tenga una racha de días críticos y el usuario trate de eliminarlo, para informarlo sobre por qué no fue eliminado su pronóstico. Soluciones a estos problemas fueron implementadas, además de una funcionalidad para copiar al portapapeles un aviso redactado con los datos de un pronóstico seleccionado, lo cual durante la etapa de pruebas emergió como conveniente para comodidad del usuario experto.

3.2.3 Pruebas de la aplicación como un todo

La mayoría de las pruebas de sistema se basaron en recrear casos de uso tal como si un usuario real los ejecutara, empleando la interfaz gráfica para hacer solicitudes al servidor, de forma que se compruebe tanto el lado del cliente como el del servidor y el vínculo entre ambos para la

satisfacción de lo especificado para los casos de uso. Estas pruebas permitieron descubrir problemas en la implementación de las funcionalidades de los campos de selección, fecha y caja de marcado del formulario constituido por el componente `<InputForm/>`, que en ciertos casos de actividad del usuario provocaban malas solicitudes al servidor; así como un error que afectaba la notificación al usuario cuando este trataba de guardar datos incorrectos. Estos fueron oportunamente corregidos.

La aplicación web también fue sometida a una prueba de sistema al usarla en cuatro navegadores diferentes (Microsoft Edge, Google Chrome, Firefox y Opera) para verificar su compatibilidad. El comportamiento en todos ellos fue exitoso.

3.3 Conclusiones parciales del capítulo

El capítulo describió las pruebas realizadas a la aplicación web en la fase final de su desarrollo, en el contexto de la teoría general del probado de software. Se abordaron las pruebas unitarias y de integración efectuadas en el lado del cliente y en el del servidor, así como pruebas de sistema de toda la aplicación. La realización de pruebas permitió la detección oportuna de problemas en la implementación, y su subsecuente corrección, lo cual condujo a que la aplicación web propuesta en su forma terminada fuera más eficaz y robusta.

CONCLUSIONES

En esta investigación se determinó los requisitos que debía cumplir la aplicación web a desarrollar, en el contexto de los elementos técnicos del estudio de la incidencia de *Phyllachora maydis*.

Se identificó y precisó los conceptos relevantes para la elaboración de la aplicación, y el estudio del estado actual del desarrollo web permitió una selección justificada de tecnologías y herramientas informáticas adecuadas para su implementación.

Se modeló e implementó la aplicación web mediante el uso de las herramientas determinadas en la investigación, con una arquitectura conformada por un lado del servidor y un lado del cliente, cada uno con sus estructuras de alto nivel y sus aspectos específicos en forma de directorios, archivos y piezas de código concebidos para proveer de forma organizada y efectiva las funcionalidades requeridas.

Finalmente, se realizó numerosas pruebas unitarias y de integración a los lados de la aplicación web por separado, y de sistema a toda ella, lo cual permitió depurarla oportunamente, mejorar su desempeño y concluir que logra manejar los datos de estudio de la enfermedad fúngica mancha de asfalto y pronosticar automáticamente sobre su aparición en las unidades de cultivo bajo vigilancia.

RECOMENDACIONES

Se recomienda como trabajo futuro:

- Desplegar la aplicación web dentro de la red local universitaria y ponerla a disposición de los expertos de la Facultad de Ciencias Agropecuarias.
- Realizar pruebas de aceptación monitoreando la calidad de la experiencia de los usuarios expertos.
- Proveer mantenimiento futuro y continuar el desarrollo de la aplicación web sobre la base de los señalamientos y sugerencias de los usuarios expertos.

REFERENCIAS BIBLIOGRÁFICAS

- Abirami, N., Lavanya, S., Madhanghi, A., 2019. A Detailed Study of Client-Server and its Architecture (Short Paper). Sri Krishna Arts and Science College, Kuniyamuthur, Coimbatore.
- Ahyar Muawwal, 2024. The Implementation of PWA (Progressive Web App) Technology in Enhancing Website Performance & Mobile Accessibility: The Implementation of PWA (Progressive Web App) Technology in Enhancing Website Performance & Mobile Accessibility. *Bul. Pos Dan Telekomun.* 22. <https://doi.org/10.17933/bpostel.v22i1.395>
- Alrawashdeh, T.A., Hnaif, A.A., Alrifae, M., Kamel, M.S., 2024. An Intelligent Framework to Generate Use Case Diagrams and Class Diagrams from Requirements Documents. <https://doi.org/10.21203/rs.3.rs-4764870/v1>
- Alturas, B., 2023. Connection between UML use case diagrams and UML class diagrams: a matrix proposal. *Int. J. Comput. Appl. Technol.* 72, 161–168. <https://doi.org/10.1504/IJCAT.2023.133294>
- Anwar, N., Kar, S., 2019. Review Paper on Various Software Testing Techniques & Strategies. *Glob. J. Comput. Sci. Technol.* 43–49. <https://doi.org/10.34257/GJCSTCVOL19IS2PG43>
- Balaj, Y., 2017. Token-Based vs Session-Based Authentication: A Survey. University of Prishtina “Hasan Prishtina.”
- DiagramasUML.com, 2024. ▷ Todos los diagramas UML. Teoría y ejemplos [WWW Document]. DiagramasUML.com. URL <https://diagramasuml.com/> (accessed 9.24.24).
- Díaz-Morales, F., De León-García De Alba, C., Nava-Díaz, C., Mendoza-Castillo, M.D.C., 2018. Inducción de resistencia a *Puccinia sorghi* y complejo mancha de asfalto (*Phyllachora maydis* y otros) en maíz (*Zea mays*). *Rev. Mex. Fitopatol. Mex. J. Phytopathol.* 37. <https://doi.org/10.18781/R.MEX.FIT.1807-6>
- Dimitrijevic, N., Zdravkovic, N., Bogdanovic, M., Mesterovic, A., 2024. Advanced Security Mechanisms in the Spring Framework: JWT, OAuth, LDAP and Keycloak.
- Django Documentation, 2024. Applications | Django documentation [WWW Document]. Django Proj. URL <https://docs.djangoproject.com/en/5.1/ref/applications/> (accessed 10.12.24).
- Django REST Framework Documentation, 2024. Django REST framework [WWW Document]. URL <https://www.django-rest-framework.org/api-guide/authentication/>
- Ehsan, A., Abuhaliqa, M.A.M.E., Catal, C., Mishra, D., 2022. RESTful API Testing Methodologies: Rationale, Challenges, and Solution Directions. *Appl. Sci.* 12, 4369. <https://doi.org/10.3390/app12094369>
- European and Mediterranean Plant Protection Organization, 2024. *Phyllachora maydis* [WWW Document]. URL https://www.eppo.int/ACTIVITIES/plant_quarantine/alert_list_fungi/https%3A%2F%2Fwww.eppo.int%2FACTIVITIES%2Fplant_quarantine%2Falert_list_fungi%2Fphyllachora_maydis (accessed 8.19.24).
- Fernández Iglesias, M.J., 2024. Pequeña introducción a las bases de datos. Universidade de Vigo.

- Ferreira, F., Borges, H.S., Valente, M.T., 2024. Refactoring react-based Web apps. *J. Syst. Softw.* 215, 112105. <https://doi.org/10.1016/j.jss.2024.112105>
- Gagliardi, V., 2021. Decoupled Django: understand and build decoupled Django architectures for Javascript front-ends. Apress, Place of publication not identified.
- Góngora-Canul, C., Jiménez-Beitia, F., Puerto-Hernández, C., Carolina Avellaneda, M., Kleczewski, N., Telenko, D.E.P., Shim, S., Solórzano, J.E., Goodwin, S.B., Scofield, S.R., Cruz, C.D., 2022. Induction of *Phyllachora maydis* (Maubl.) signs on corn leaves. *Res. Sq.* 12.
- Hernández Cruz, L.M., González Novelo, M.J., Cab Chan, J.R., Mex Álvarez, D.C., 2023. Implementación de la aplicación web BITA en Google Compute Engine. *Multidiscip. Ing.* 7, 107–117. <https://doi.org/10.29105/mdi.v7i10.228>
- Kassab, M., DeFranco, J.F., Laplante, P.A., 2017. Software Testing: The State of the Practice. *IEEE Softw.* 34, 46–52. <https://doi.org/10.1109/MS.2017.3571582>
- Kim, M., Sinha, S., Orso, A., 2023. Adaptive REST API Testing with Reinforcement Learning.
- Kowalczyk, K., Szandala, T., 2024. Enhancing SEO in Single-Page Web Applications in Contrast With Multi-Page Applications. *IEEE Access* 12, 11597–11614. <https://doi.org/10.1109/ACCESS.2024.3355740>
- Kralina, H., Popova, A., 2024. NOWADAYS TRENDS IN WEB DEVELOPMENT, in: EDUCATION AND SCIENCE OF TODAY: INTERSECTORAL ISSUES AND DEVELOPMENT OF SCIENCES. Presented at the EDUCATION AND SCIENCE OF TODAY: INTERSECTORAL ISSUES AND DEVELOPMENT OF SCIENCES, European Scientific Platform. <https://doi.org/10.36074/logos-29.03.2024.068>
- Kulkarni, Dr.R.N., Srinivasa, C.K., Dept. of Computer Science & Engineering, BITM, VTU, Ballari, India., 2021. Novel approach to transform UML Sequence diagram to Activity diagram. *J. Univ. Shanghai Sci. Technol.* 23, 1247–1255. <https://doi.org/10.51201/JUSST/21/07300>
- Kumar, M., Nandal, D.R., 2024. Python's Role in Accelerating Web Application Development with Django. *Int. Res. J. Adv. Eng. Manag. IRJAEM* 2, 2092–2105. <https://doi.org/10.47392/IRJAEM.2024.0307>
- Liu, Yi, Li, Y., Deng, G., Liu, Yang, Wan, R., Wu, R., Ji, D., Xu, S., Bao, M., 2022. Morest: model-based RESTful API testing with execution feedback, in: Proceedings of the 44th International Conference on Software Engineering. Presented at the ICSE '22: 44th International Conference on Software Engineering, ACM, Pittsburgh Pennsylvania, pp. 1406–1417. <https://doi.org/10.1145/3510003.3510133>
- Llamuca-Quinaloa, J., Vera-Vincent, Y., Tapia-Cerda, V., 2021. Análisis comparativo para medir la eficiencia de desempeño entre una aplicación web tradicional y una aplicación web progresiva. *TecnoLógicas* 24, e1892. <https://doi.org/10.22430/22565337.1892>
- Mendez, M., 2014. The Missing Link - An Introduction to Web Development and Programming. Open SUNY Textbooks.
- Mwamba Nyabuto, G., Mony, V., Mbugua, S., 2024. Architectural Review of Client-Server Models. *Int. J. Sci. Res. Eng. Trends* 10, 139–143.
- NextAuth Documentation, 2024. Introduction | NextAuth.js [WWW Document]. URL <https://next-auth.js.org/getting-started/introduction> (accessed 9.6.24).

- Nguyen, T., 2023. What are APIs? A computer scientist explains the data sockets that make digital life possible [WWW Document]. The Conversation. URL <http://theconversation.com/what-are-apis-a-computer-scientist-explains-the-data-sockets-that-make-digital-life-possible-213042> (accessed 9.6.24).
- Nicacio, J., Petrillo, F., 2020. Applying system descriptors to address ambiguity on deployment diagrams. <https://doi.org/10.48550/ARXIV.2008.11060>
- Olanrewaju, R.F., Khan, B.U.I., Morshidi, M.A., Anwar, F., Kiah, M.L.B.M., 2021. A Frictionless and Secure User Authentication in Web-Based Premium Applications. *IEEE Access* 9, 129240–129255. <https://doi.org/10.1109/ACCESS.2021.3110310>
- Panwar, V., 2024. Web Evolution to Revolution: Navigating the Future of Web Application Development. *Int. J. Comput. Trends Technol.* 72, 34–40. <https://doi.org/10.14445/22312803/IJCTT-V72I2P107>
- Patel, V., 2023. Analyzing the Impact of Next.JS on Site Performance and SEO. *Int. J. Comput. Appl. Technol. Res.* <https://doi.org/10.7753/IJCATR1210.1004>
- Pytest Documentation, 2024. Pytest Documentation [WWW Document]. URL <https://docs.pytest.org/en/stable/> (accessed 10.21.24).
- Ramírez-Galvis, J.P., 2023. Generación de un servicio RSS con Django. <https://doi.org/10.13140/RG.2.2.15992.62723>
- Satter, A., Tabassum, A., Ishrat, J.E., 2023. Software Evolution of Next.js and Angular. *Int. J. Eng. Manuf.* 13, 20–33. <https://doi.org/10.5815/ijem.2023.04.03>
- Shukla, A., 2023. Modern JavaScript Frameworks and JavaScript's Future as a FullStack Programming Language. *J. Artif. Intell. Cloud Comput.* 1–5. [https://doi.org/10.47363/JAICC/2023\(2\)144](https://doi.org/10.47363/JAICC/2023(2)144)
- Stanek, J., Killough, D., 2023. Synthesizing JSON Schema Transformers. *Univ. Wis.* 1, 7.
- The PostgreSQL Global Development Group, 2024. PostgreSQL: About [WWW Document]. URL <https://www.postgresql.org/about/> (accessed 8.20.24).
- Umar, M.A., 2020. Comprehensive study of software testing: Categories, levels, techniques, and types. <https://doi.org/10.36227/techrxiv.12578714>
- Uzun, E., Yerlikaya, T., Kirat, O., 2018. Object-Based Entity Relationship Diagram Drawing Library: Entrel.js. *J. Tech. Univ. -Sofia Plovdiv Branch Bulg. Fundam. Sci. Appl.* 24.
- Valdez Ocampo, F.D., Huerta Maciel, Á.M., Mongelós Barrios, C.A., Sánchez Jara, R., Ruiz Díaz Lovera, E.M.D., Sanchez Gonzalez, M.A., 2024. Evaluación de cultivares de maíz (*Zea mays* L.) sembrados en diferentes arreglos espaciales. *Rev. Alfa* 8, 363–375. <https://doi.org/10.33996/revistaalfa.v8i23.269>
- Vargas Zermeño, E., 2024. Criterios de implementación de Tailwind CSS en desarrollos frontend. *Cuad. Téc. Univ. DGTIC* 2. <https://doi.org/10.22201/dgtic.ctud.2024.2.3.63>
- Venkata Koteswara Rao Ballamudi, Karu Lal, Harshith Desamsetti, Sreekanth Dekkati, 2021. Getting Started Modern Web Development with Next.js: An Indispensable React Framework. *Digit. Sustain. Rev.* 1.

- Vinces-Tachong, R.E., Vélez-Ruiz, M.C., Gaibor-Fernández, R.R., Herrera-Eguez, F.E., 2022. Implementación del procesamiento de imágenes para la evaluación de la mancha de asfalto (*Phyllachora maydis*) en maíz (*Zea mays*). *Rev. TERRA Latinoam.* 40. <https://doi.org/10.28940/terra.v40i0.1066>
- Visual Paradigm, 2024. What is Unified Modeling Language (UML)? [WWW Document]. URL <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/> (accessed 8.30.24).