

Front-end

1. Estructura de directorios.

La estructura del proyecto frontend de FacturaApp se ha organizado en directorios lógicos que separan claramente los recursos estáticos, las páginas HTML y los módulos funcionales. Esta organización facilita la mantenibilidad y escalabilidad del proyecto a medida que crecen sus funcionalidades.

```
FacturaApp/
├── assets/
│   ├── css/
│   │   ├── sb-admin-2.css
│   │   └── style.css
│   ├── img/
│   │   ├── esquema-base-datos.png
│   │   ├── favicon.png
│   │   ├── forgot.jpg
│   │   ├── login-img.png
│   │   ├── loginimage.jpg
│   │   └── logo.png
│   └── js/
│       ├── actions_reports.js
│       ├── article_actions.js
│       ├── company_actions.js
│       ├── customer_actions.js
│       ├── dashboard_vars.js
│       ├── factura_actions.js
│       ├── global.js
│       ├── menu_actions.js
│       ├── menu_functions.js
│       ├── utils.js
│       └── vista_facturas.js
├── index.html
├── forgot-password/
│   └── index.html
├── register/
│   └── index.html
├── pages/
│   └── report.html
├── favicon.ico
├── dashboard/
│   └── index.html
```

← funciones JS necesarias para la logica del negocio

← Página principal (inicio de sesión)

← Carpeta para recuperación de contraseña

← Página de con instrucciones para recuperar la cuenta

← Carpeta de registro de nuevos usuarios

← Página de registro nuevo usuario(empresa)

← Página de reportes solo para admin (para mi)

← Icono del navegador

← Módulo visual principal de usuario

← Página en la que esta toda la lógica del negocio (creación de artículos, clientes, facturas, envió emails, creación de PDF)

Justificación de la estructura

Esta organización cumple con buenas prácticas de desarrollo web, ya que:

- Separa claramente lógica (JS), estilo (CSS) y contenido (HTML).
- Agrupa funcionalidades por módulo, lo que permite escalabilidad.
- Facilita la navegación y mantenimiento del código por parte de nuevos desarrolladores

2. Planificar las tareas de programación front-end

Para el desarrollo del frontend de **FacturaApp**, he diseñado una planificación clara y estructurada que abarca tanto la construcción de las interfaces HTML/CSS como la programación funcional en JavaScript. El objetivo ha sido implementar una interfaz de usuario completa y funcional que permita crear, consultar y gestionar todos los datos empresariales clave antes de su envío al backend desarrollado en Django.

Fases de planificación de tareas

Diseño de la arquitectura de pantallas

Se decidió que la aplicación contaría con 4 pantallas principales, cada una con una finalidad específica:

Ruta	Función principal
/index.html	Pantalla de inicio de sesión del usuario.
/register/index.html	Formulario para el registro de nuevos usuarios.
/forgot-password/index.html	Página de ayuda para recuperar el acceso a la cuenta.
/dashboard/index.html	Página principal tras login, núcleo del área de gestión.

Maquetación HTML y estilo CSS

- Se crearon las páginas base con estructura semántica HTML5.
- Se definieron estilos consistentes desde `assets/css/style.css`.
- Se aplicó un diseño limpio, responsive y compatible con escritorio y dispositivos móviles.
- Se organizó todo dentro de carpetas funcionales: `assets/css`, `assets/img`, `assets/js`.

Implementación de lógica en JavaScript (JS)

En la vista **dashboard**, se concentró la lógica del negocio. Desde ahí, mediante JavaScript puro, se programaron funcionalidades para:

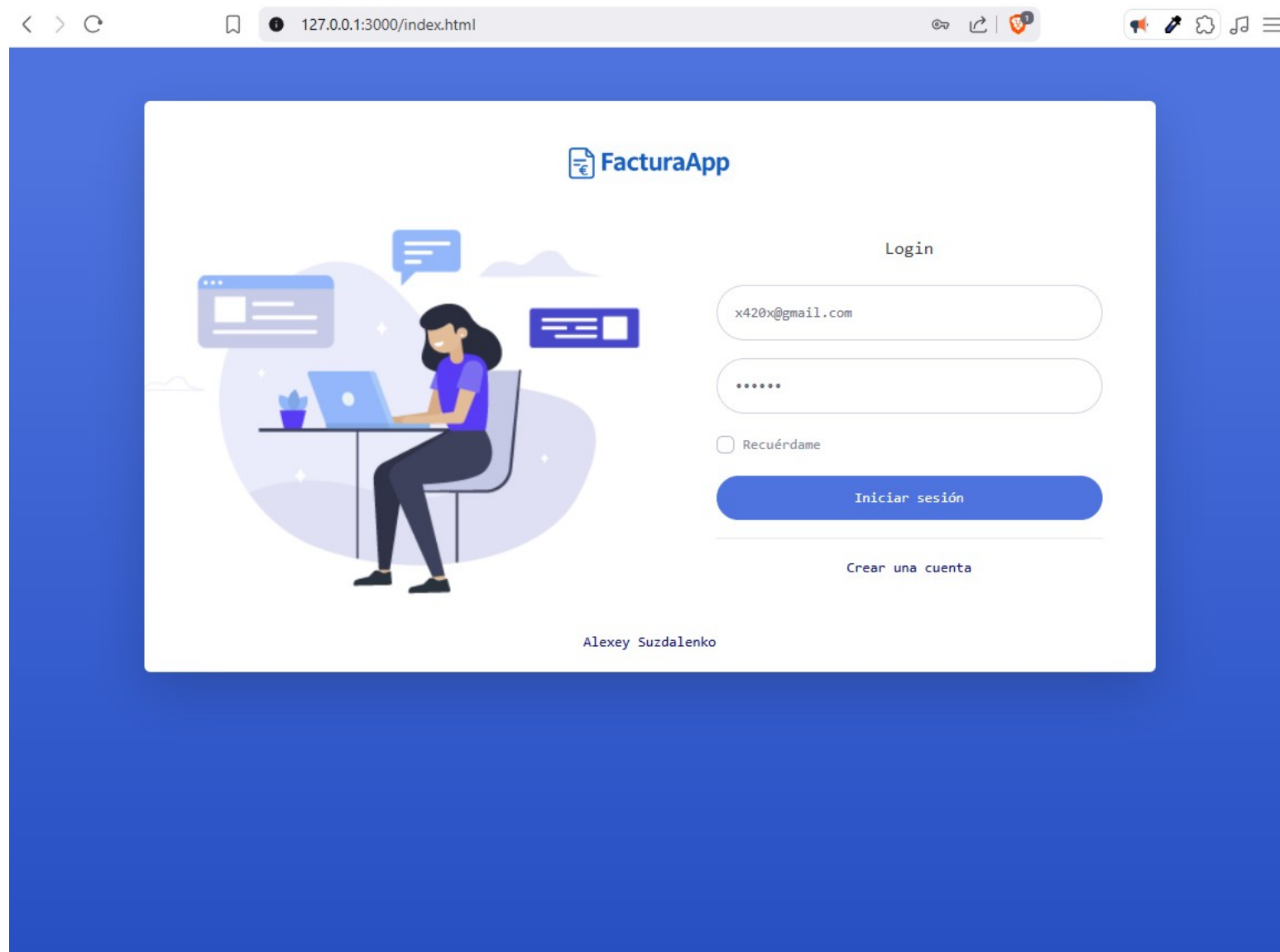
- Mostrar dinámicamente formularios según el módulo activo (empresa, artículos, clientes, facturas...).
- Capturar datos introducidos por el usuario en tiempo real.
- Calcular totales en líneas de factura (precio × cantidad, IVA, subtotal, etc.).
- Generar estructuras de datos listas para ser enviadas al backend Django (en JSON).
- Gestionar los formularios de creación, modificación y visualización de:
 - Datos de la empresa
 - Artículos (productos o servicios)
 - Clientes
 - Facturas y líneas de factura

Preparación para la comunicación con el backend

- Se estructuraron las llamadas a funciones `fetch` en JS para enviar/recibir datos con el backend (Django).
- Cada módulo JS maneja sus propias operaciones (`company_actions.js`, `factura_actions.js`, etc.), organizadas dentro de `assets/js/`.
- Se establecieron puntos clave para implementar los endpoints API en Django una vez esté listo el backend.

La planificación front-end ha seguido una lógica modular, separando visualización, comportamiento y lógica de negocio. Esto ha permitido construir una interfaz profesional y extensible, lista para conectarse a un backend en producción. Cada pantalla se ha implementado de forma progresiva, completando primero la estructura y luego la funcionalidad asociada en JavaScript.

3. Implementación y maquetación al menos de la página home completamente funcional.



La página de inicio de sesión (index.html) de FacturaApp ha sido diseñada y desarrollada con una estructura profesional, funcional y visualmente atractiva. Esta representa la puerta de entrada a la aplicación y su diseño responde tanto a criterios estéticos como funcionales.

Diseño y maquetación

- La interfaz se ha construido utilizando **HTML5 semántico**, asegurando accesibilidad y correcta estructura.
- El diseño visual está basado en **CSS3 personalizado** dentro del archivo style.css, respetando la paleta de colores corporativa de FacturaApp.
- Se ha dividido el layout en dos columnas principales:
 - **Columna izquierda:** ilustración representativa con estilo moderno, reforzando el contexto tecnológico y profesional del proyecto.
 - **Columna derecha:** formulario de login y elementos interactivos.

Elementos implementados

- **Logo corporativo** en la parte superior, con icono e identidad visual de la marca.
- **Formulario de inicio de sesión** con campos para:
 - Correo electrónico
 - Contraseña
 - Casilla "Recuérdame"

- **Botón de inicio de sesión** con estilo interactivo y accesible.
- Enlace de acceso directo a la creación de cuenta (register) para usuarios nuevos.
- Pie de página con firma de autor: “Alexey Suzdalenko”.

Funcionalidad

- El formulario está preparado para enviar datos al backend mediante POST para iniciar sesión.
- Se han aplicado **validaciones de entrada básicas con JavaScript** (campos obligatorios).
- El diseño es **responsive**, adaptándose correctamente a pantallas móviles y de escritorio.

El resultado es una **página de inicio funcional, intuitiva y coherente con la identidad de FacturaApp**, que cumple con su propósito principal: permitir el acceso seguro al área privada de la aplicación. La estructura está lista para conectarse a la lógica de autenticación en el backend Django.

4. Despliegue de la aplicación en un servidor web local.

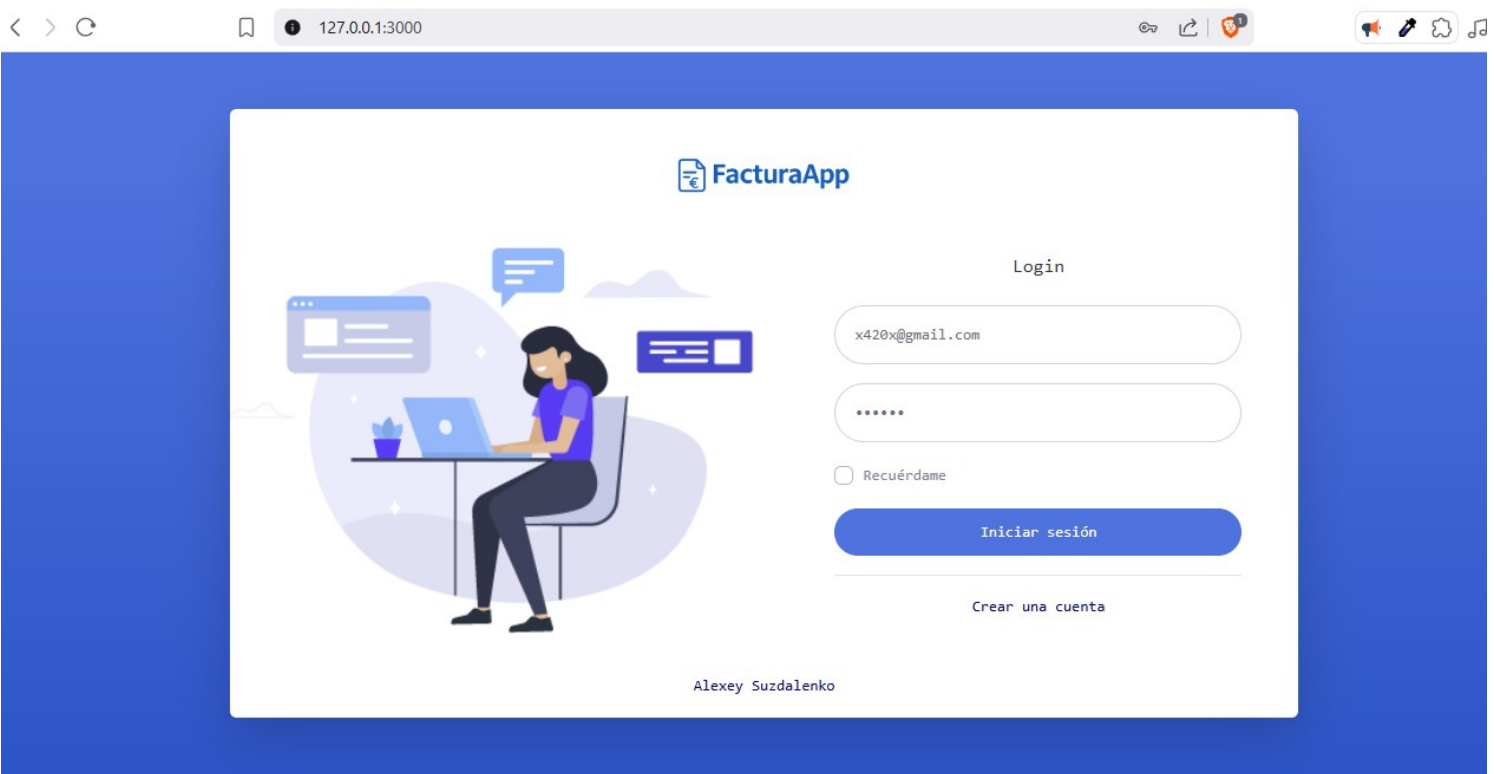
Para verificar y probar el correcto funcionamiento del frontend de **FacturaApp**, se realizó el despliegue local de la aplicación en un entorno de desarrollo personal. Este paso es fundamental para comprobar visualmente el resultado del código HTML, CSS y JavaScript, y garantizar su correcto comportamiento antes de subirlo a producción.

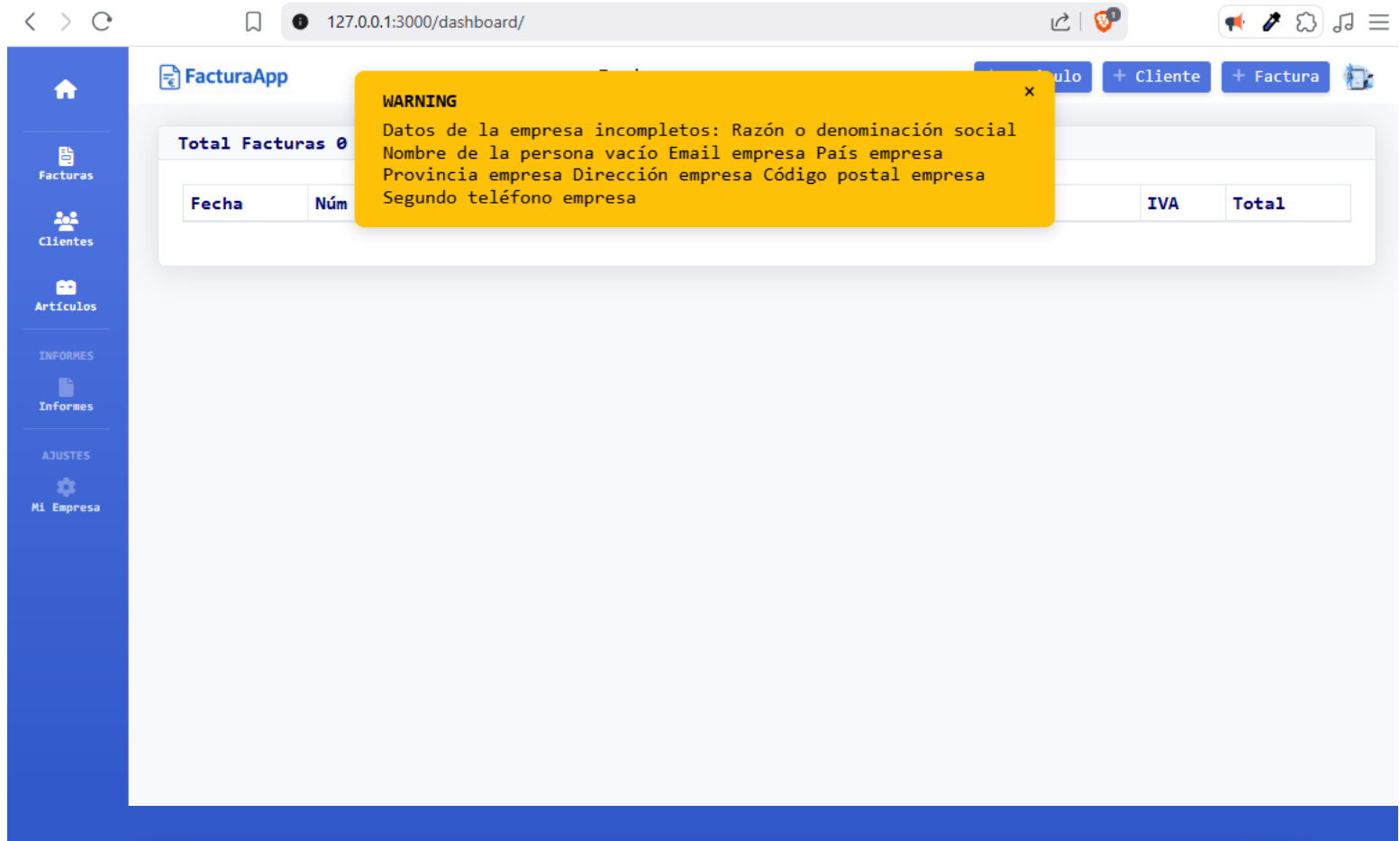
Método de despliegue utilizado

Para el despliegue local, utilicé el entorno de desarrollo **Visual Studio Code** junto con la extensión **Live Server**, que permite ejecutar un servidor local con recarga automática de páginas.

Pasos realizados:

1. Se abrió el proyecto en Visual Studio Code.
2. Se instaló la extensión “Live Server”.
3. Se hizo clic derecho sobre el archivo index.html y se seleccionó “Open with Live Server”.





4. El navegador abrió la dirección `http://127.0.0.1:3000/`, mostrando la página de inicio (`index.html`)

Verificaciones realizadas

- Visualización de todos los elementos gráficos, formularios y botones.
- Comprobación de la funcionalidad JavaScript (formularios, validaciones, interacción).
- Confirmación de la estructura responsive del diseño en distintas resoluciones.
- Pruebas de enlaces entre pantallas (navegación entre login, registro y recuperación de contraseña).

El despliegue local ha sido fundamental para desarrollar y depurar la interfaz de **FacturaApp** de forma efectiva. Gracias a esta prueba en entorno local, se garantizó la calidad visual y funcional de las páginas antes de subir la aplicación a un servidor remoto.

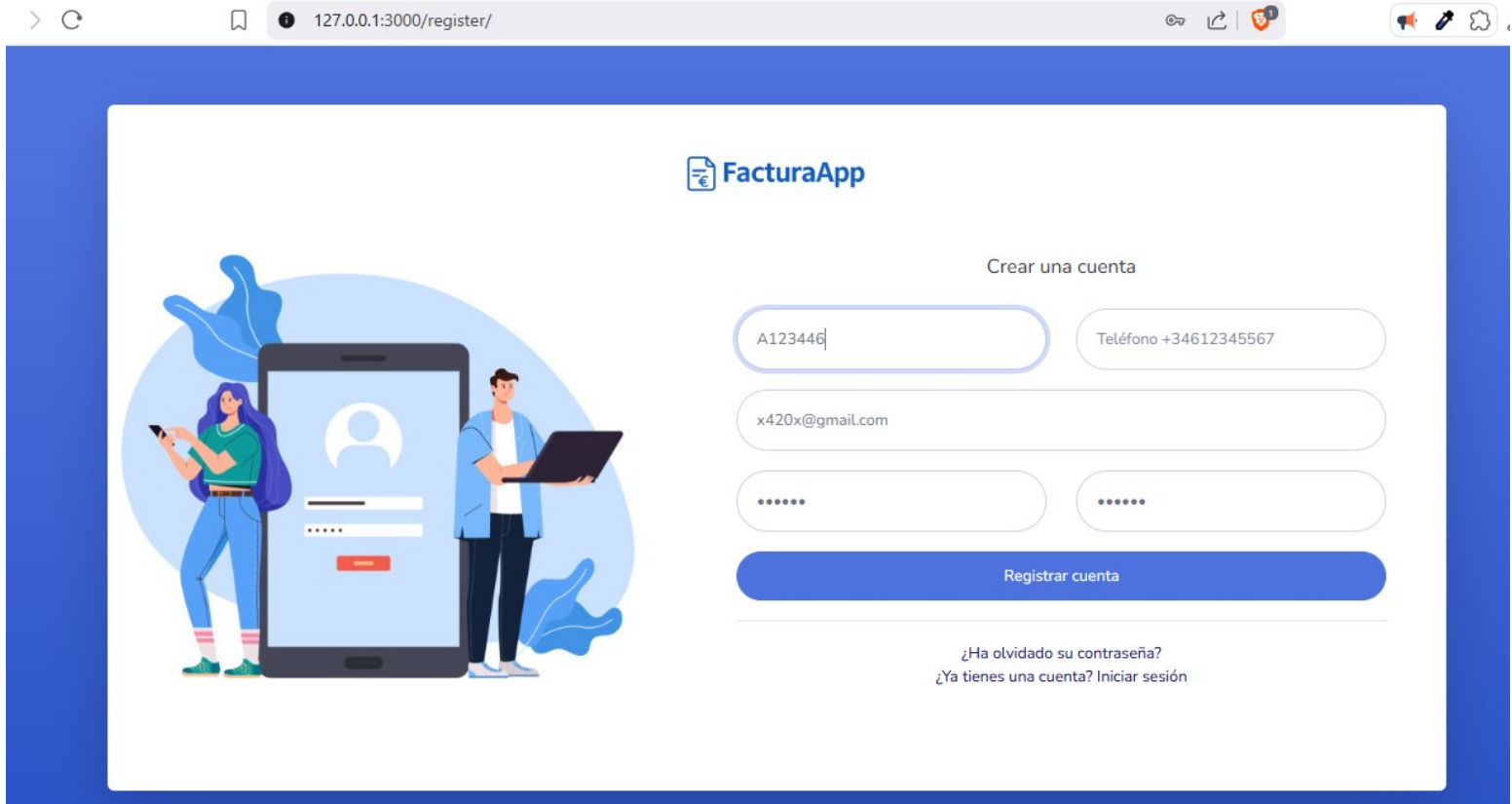
Front-end 2

1. Implementar el resto de las secciones front-end

Tras desarrollar la página principal de inicio de sesión (`index.html`), se implementaron el resto de las secciones necesarias para el correcto funcionamiento del frontend de FacturaApp. Estas secciones forman la estructura visual completa que permite al usuario interactuar con las distintas funcionalidades de la aplicación.

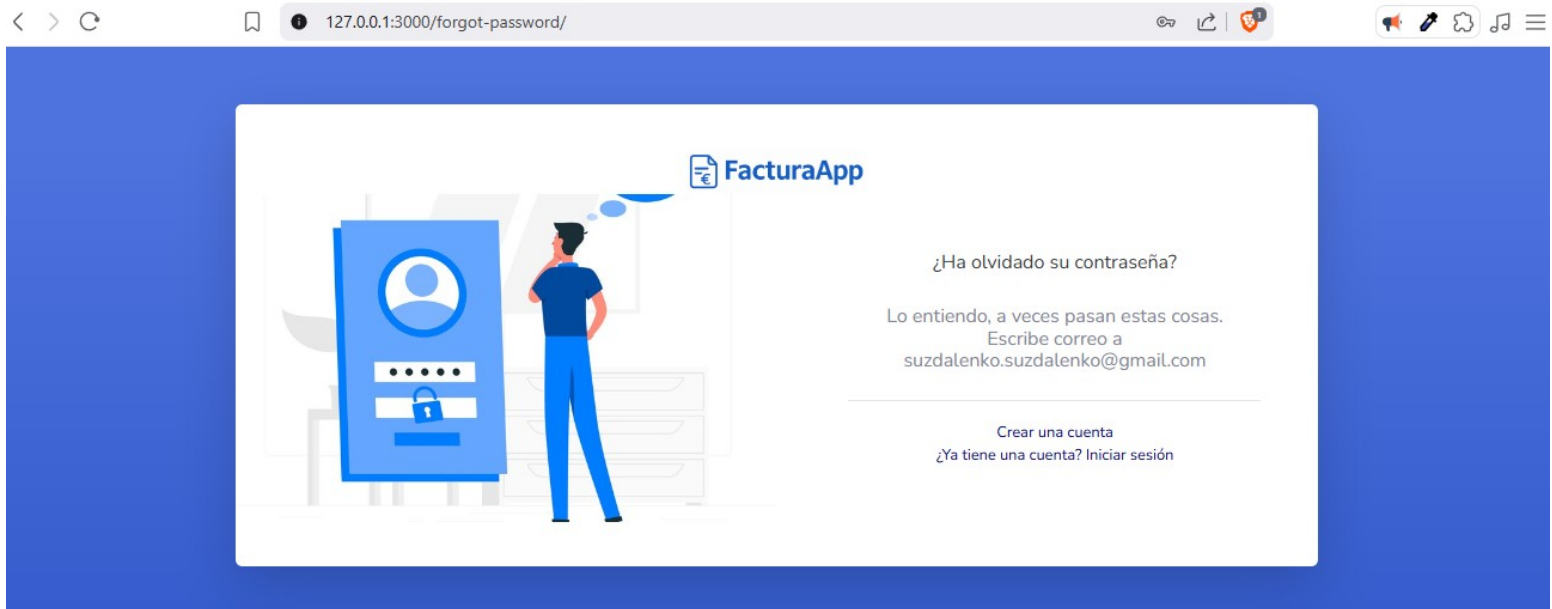
`/register/index.html`

- Página para el **registro de nuevos usuarios**.
- Contiene formulario con campos como nombre, email, contraseña, confirmación, etc.
- Estilizada con la misma identidad visual que el resto del sitio.
- Validación de datos con JavaScript antes de enviar al backend.



/forgot-password/index.html

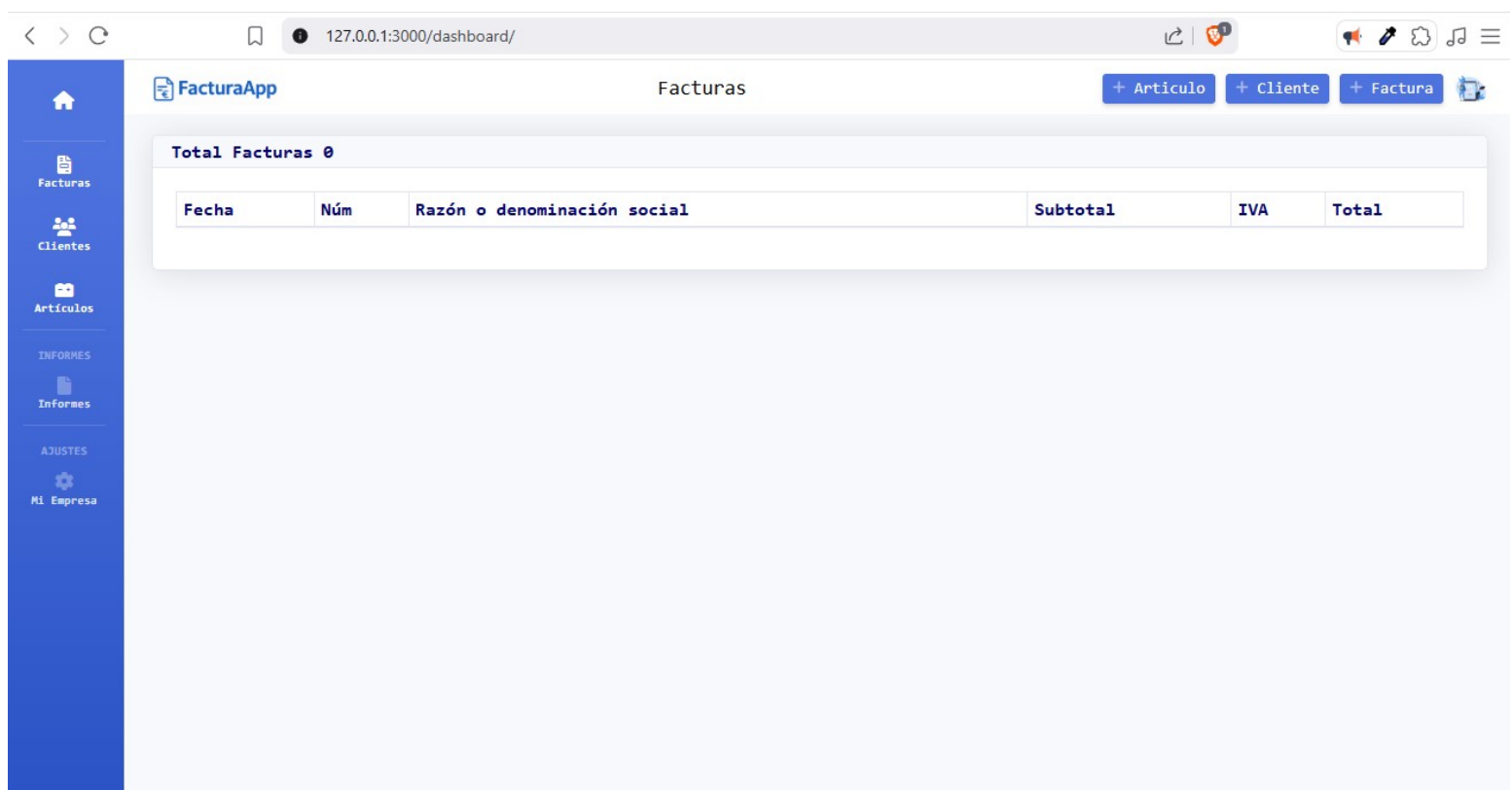
- Página para **recuperación de cuenta**.
- Muestra instrucciones al usuario para restablecer la contraseña.
- Diseño coherente con el resto de la aplicación.



/dashboard/index.html

- Página principal tras el inicio de sesión.
- Contiene el **núcleo del negocio**: gestión de clientes, artículos, facturas, y datos de empresa.
- Integra múltiples secciones mostradas dinámicamente mediante JavaScript:
 - Formulario de empresa
 - Módulo de clientes
 - Módulo de artículos

- Módulo de facturas
- Reportes y estadísticas
- Utiliza scripts modulares (assets/js/) para manejar cada bloque de funcionalidades.
- Estructura responsive adaptable a diferentes resoluciones.



Formulario de la propia empresa



Vista de creación de la factura:

127.0.0.1:3000/dashboard/#CreateInvoice

FacturaApp

Crear Factura

+ Artículo

+ Cliente

+ Factura

Tipo Factura

Tipo
Ordinaria/Rectificativa/Abono
Factura Ordinaria

Datos cliente y tipo de factura

Número cliente

NIF CIF cliente

Razón o denominación social

Nombre cliente

Datos vehículo

Matrícula

Marca/Modelo/Kilómetros

Líneas de factura

Código Art.	Descripción Art.	Cantidad	Precio	% Dto.	Importe	IVA
		1		0		21 %
+ Línea						
	Mano de obra		40,00			21 %
						Subtotal: 0
						IVA: 0
						Total: 0

Observaciones

Crear Factura

Organización y modularidad

- Todo el frontend se basa en HTML, CSS y JavaScript puros.
- Se usa una **estructura modular de scripts** en assets/js/, separando acciones por entidad (clientes, facturas, etc.).
- Se mantiene la coherencia visual con el uso de estilos comunes desde style.css.

Gracias a la implementación completa de todas las secciones del frontend, el usuario puede navegar por la aplicación, gestionar su empresa y sus documentos sin necesidad de recargar ni salir del entorno visual. Todo esto ha sido desarrollado en código limpio, funcional y fácilmente conectable con el backend de Django.

2. Desplegar la aplicación en un servidor web remoto, con acceso desde internet.

Una vez completado el desarrollo local del frontend de **FacturaApp**, se procedió al despliegue de la aplicación en un servidor remoto con acceso público a través de internet. Para ello se utilizó el servicio de alojamiento estático **Firebase Hosting** de Google.

Plataforma de despliegue: Firebase Hosting

Firebase fue la plataforma elegida por su facilidad de uso, gratuidad en proyectos pequeños y velocidad de implementación. Esta solución permite alojar proyectos web estáticos compuestos por HTML, CSS, JavaScript e imágenes.

Ventajas de Firebase Hosting:

- Acceso rápido y seguro mediante HTTPS.
- Fácil integración con herramientas de desarrollo.
- Configuración y despliegue en pocos minutos.
- URL personalizada sin coste adicional.

Pasos realizados para el despliegue

1. Se instaló la CLI de Firebase:

```
npm install -g firebase-tools
```

2. Se inició sesión con una cuenta de Google:

```
firebase login
```

3. Se inicializó el proyecto en la raíz del frontend:

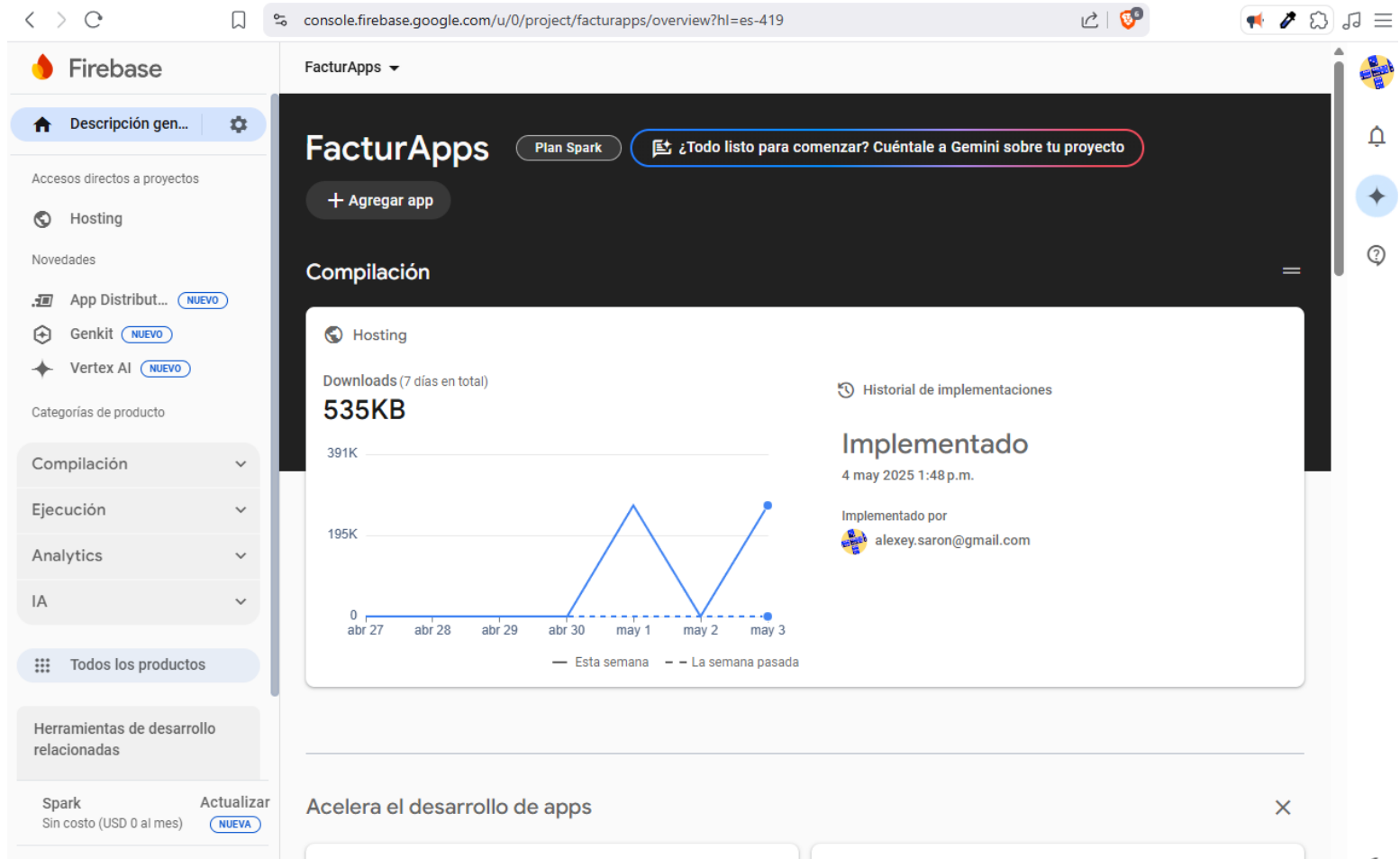
```
firebase init
```

- Se seleccionó "Hosting"
- Se configuró la carpeta pública como `.` o `front-html`

4. Se construyó el contenido y se desplegó:

```
firebase deploy
```

Proyecto FacturaApp en firebase:

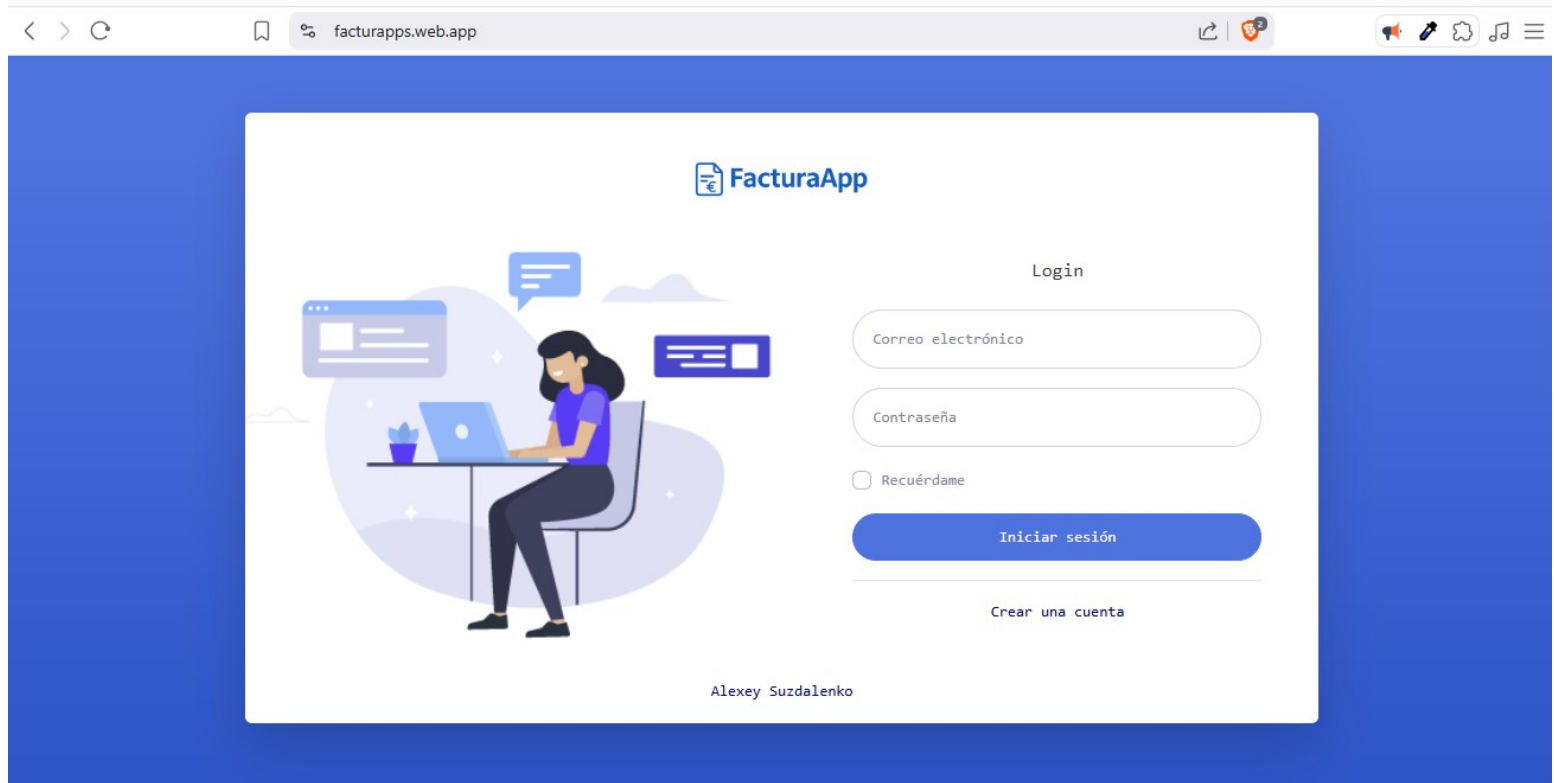


Resultado

El proyecto quedó publicado en la siguiente URL accesible desde cualquier navegador:

<https://facturapps.web.app/>

Esto permite probar la interfaz en un entorno real, mostrarla públicamente y facilitar el acceso a usuarios o clientes.



1. Planificar las tareas de programación Back-end.

La programación del backend de **FacturaApp** se ha planteado utilizando el framework **Django**, aprovechando su potente sistema ORM y su estructura modular, lo que permite una arquitectura clara, reutilizable y fácilmente escalable. Para facilitar la portabilidad del proyecto, se ha optado por usar **SQLite** como base de datos por defecto.

Modelo de desarrollo backend

El desarrollo se organiza siguiendo el patrón **controlador personalizado**, donde cada grupo funcional de la aplicación está asociado a un archivo controlador dedicado en la carpeta `invoice/controllers`. Cada controlador contiene las funciones que manejan la lógica específica de cada recurso (empresa, facturas, artículos, usuarios, etc.).

Ejemplos:

- `login_controller.py` → inicio de sesión
- `invoice_controller.py` → operaciones de facturación
- `report_controller.py` → generación de informes
- `default_controller.py` → operaciones genéricas CRUD

Organización de las rutas

Las rutas se definen en el archivo `urls.py` utilizando `path()` y se estructuran para reflejar la acción y la entidad. Algunos ejemplos son:

```
path('register/', registrar_controller.try_register)
path('invoice/<str:action>/<int:id>', invoice_controller.invoice_actions)
path('default/<str:action>/<str:entity>/<int:id>', default_controller.default_actions)
```

Esto permite mantener una API limpia, orientada a entidades y acciones, como:

- `/invoice/create/0` → crear nueva factura
- `/default/delete/customer/5` → eliminar cliente con ID 5

Modelado de datos con Django ORM

Se han definido varias clases `models.Model` que representan las entidades del sistema. Estas se traducen automáticamente en tablas de la base de datos SQLite. Cada modelo contiene campos clave, restricciones (`unique`, `null`, `indexes`) y relaciones indirectas.

Modelos principales:

- **Company**: Datos de la empresa usuaria de FacturaApp
- **Customer**: Clientes de la empresa
- **Article**: Artículos o servicios facturables
- **Factura**: Cabecera de factura
- **Facturalineas**: Líneas de detalle asociadas a cada factura
- **Vehicledata**: Datos del vehículo (opcional, para facturas relacionadas)
- **Document**: Tipos y contadores de documentos
- **Country**: Catálogo de países para selector

Tareas de programación planificadas

1. **Definición de modelos ORM** y creación inicial de la base de datos.

2. Implementación de controladores para cada entidad funcional:

- Registro e inicio de sesión (try_register, try_login)
- CRUD de empresa, clientes, artículos y facturas
- Generación de PDFs
- Exportación de reportes

3. Configuración de rutas limpias que enlacen cada petición del frontend con su respectivo controlador.

4. Preparación de respuestas JSON para ser consumidas directamente por el frontend con fetch().

5. Validación de datos y seguridad básica, incluyendo almacenamiento de contraseñas con hash.

Conclusión

La planificación del backend de FacturaApp se basa en una arquitectura clara, separada por controladores y con rutas REST organizadas por entidad y acción. Gracias al uso del ORM de Django, se ha simplificado la gestión de datos, y con SQLite se garantiza que el proyecto pueda ejecutarse sin complicaciones en cualquier entorno de desarrollo.

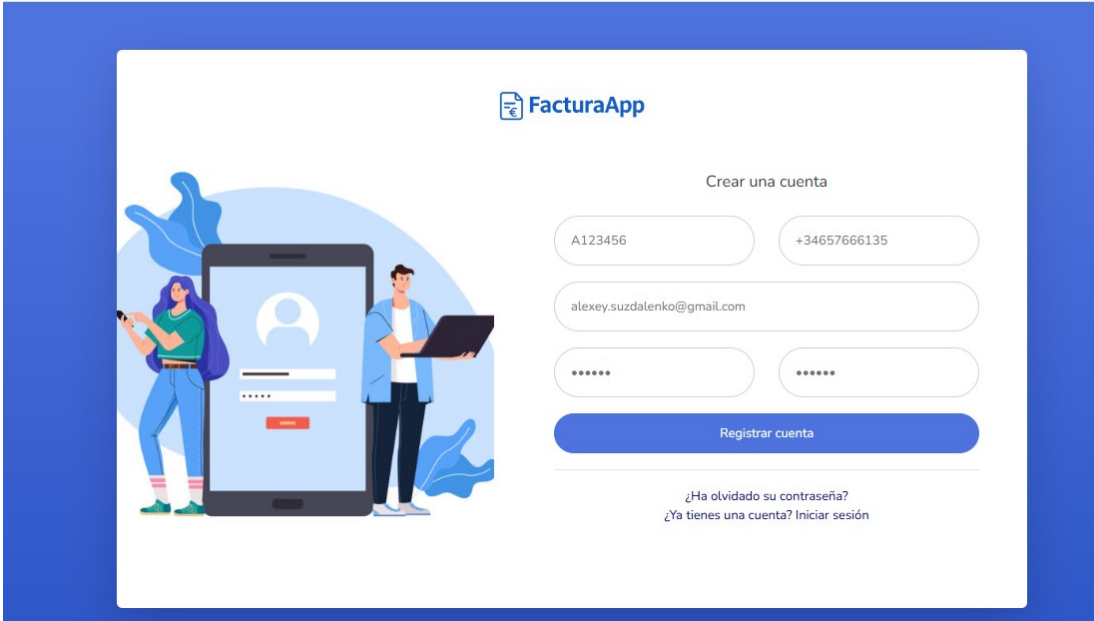
2. Autenticación.

El sistema de autenticación de **FacturaApp** se ha desarrollado utilizando una arquitectura clásica basada en formularios, con comunicación entre el frontend (HTML/JS) y el backend (Django) a través de peticiones POST. La autenticación está centrada en la entidad Company, que representa la empresa usuaria de la aplicación.

Registro de cuenta

La autenticación comienza con el proceso de **registro**, accesible desde la ruta:

/register/



Frontend:

- Formulario con campos: CIF, Teléfono, Email y Contraseña (doble entrada).
- Validación de campos en el navegador con JavaScript antes de enviar la solicitud.
- Uso de FormData para enviar los datos mediante fetch() a la API Django.

Backend:

- Ruta: path('register/', registrer_controller.try_register)

- El controlador try_register crea una nueva empresa (modelo Company).
- La contraseña se almacena encriptada con make_password() de Django.
- Se genera un uid único y se registra la fecha de alta.
- El resultado se devuelve como JSON con estado ok o error

Código python de registro de una cuenta nueva:

```

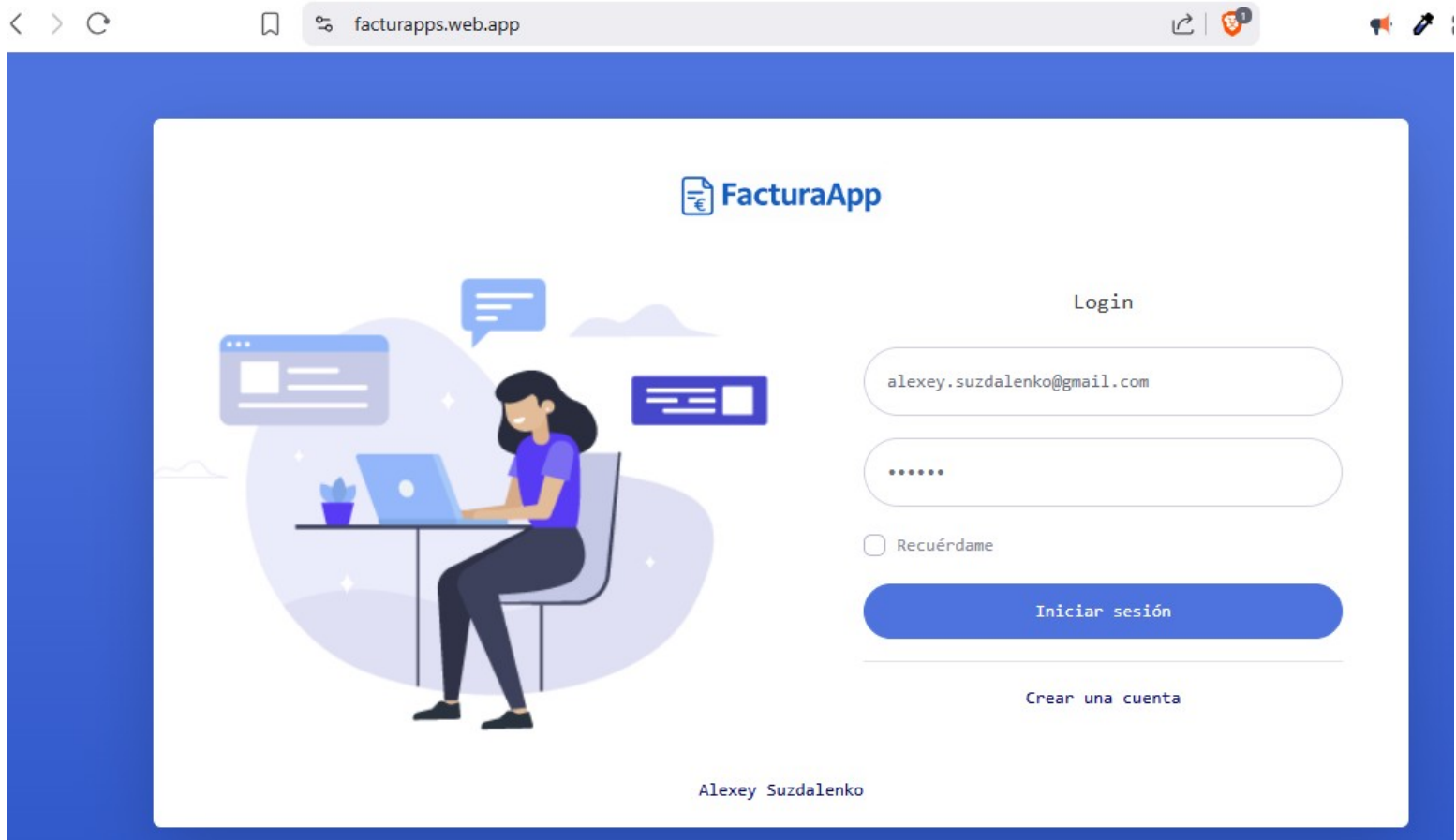
invoice > controllers > registrer_controller.py > try_register
1  from invoice.models.company import Company
2  from django.contrib.auth.hashers import make_password
3  from ..utils.util_suzdal import json_suzdal
4  from ..utils.time_suzdal import time_suzdal
5  from ..utils.time_suzdal import second_suzdal
6
7  def try_register(request):
8      try:
9
10         cif = request.POST.get('cif').strip()
11         email = request.POST.get('email').strip()
12         tlf = request.POST.get('tlf').strip()
13         password = request.POST.get('password').strip()
14
15         company, createdTrue = Company.objects.get_or_create(cif=cif, email=email,)
16         company.tlf = tlf
17         company.numvisit += 1
18
19         if createdTrue:
20             company.state = 'activo'
21             company.regtime = time_suzdal()
22             company.uid = second_suzdal()
23             company.password = make_password(password)
24
25         company.lastvisit = time_suzdal()
26         company.save()
27
28         rdata = {
29             'cif': company.cif,
30             'email': company.email,
31             'password': company.password,
32             'status': 'ok',
33         }
34
35         res = json_suzdal(rdata)
36         return res
37
38     except Exception as e:
39         return json_suzdal({'message': str(e), 'status': 'error'})
40
41

```

Ahora con la cuenta recién creada toca hacer el login

El acceso al sistema se realiza desde la pantalla:

/index.html



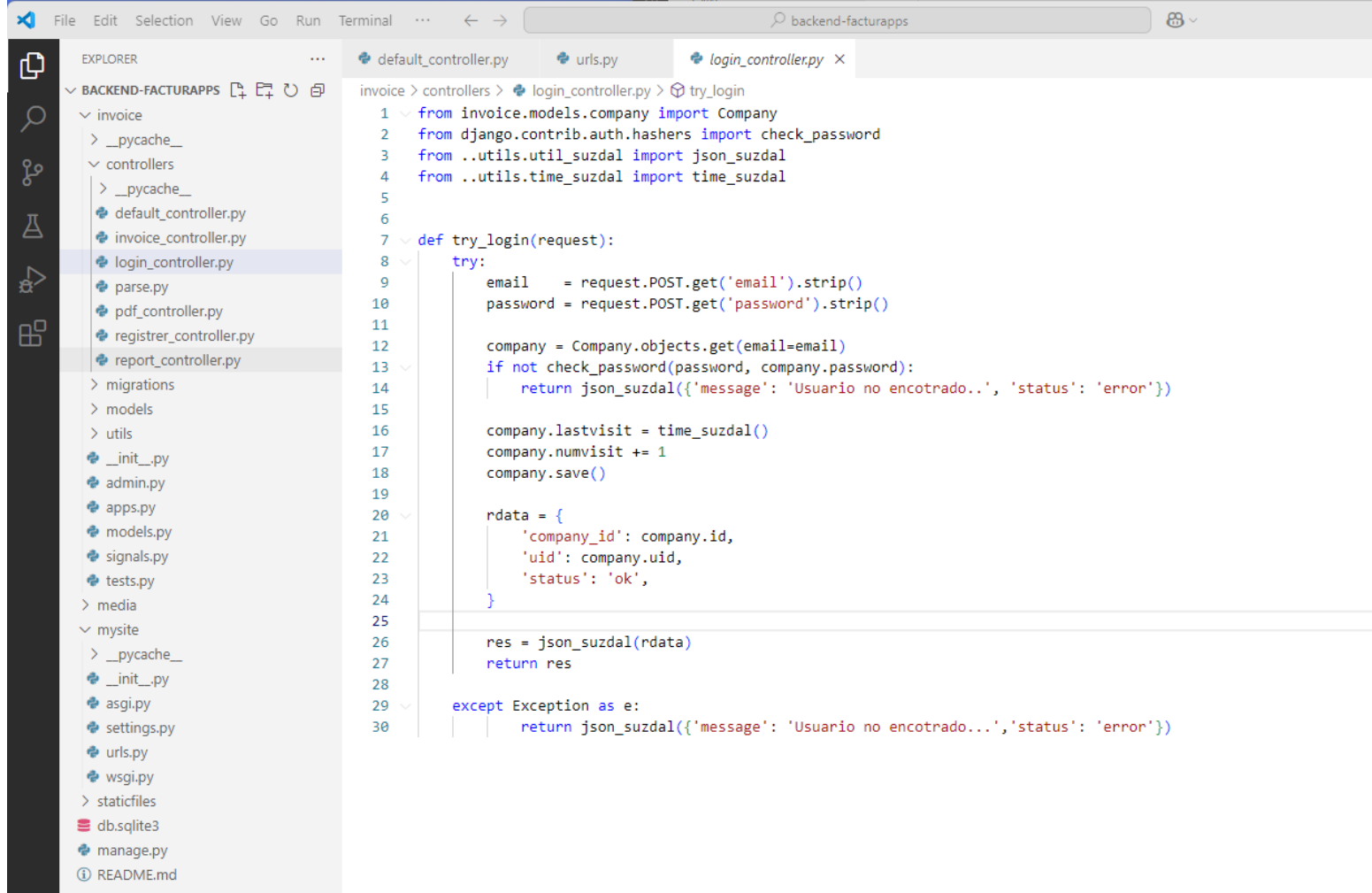
Frontend:

- Formulario de login con Email y Contraseña.
- Al enviar, se hace una llamada `fetch()` a la ruta `/login/`.
- Si la respuesta es válida, se almacenan datos en `localStorage` (como `company_id` y `uid`) y se redirige al dashboard.

Backend:

- Ruta: `path('login/', login_controller.try_login)`
- El controlador busca la empresa por email.
- Se valida la contraseña con `check_password()` contra el hash almacenado.
- Si los datos son correctos:
 - Se registra el último acceso (`lastvisit`) y se incrementa el contador de visitas.
 - Se devuelve un JSON con `company_id`, `uid` y `status: ok`

Código python login de usuario:



The screenshot shows a code editor with the following structure:

- EXPLORER:** A sidebar on the left showing the project structure. The 'controllers' directory is expanded, showing files like `__pycache__`, `default_controller.py`, `invoice_controller.py`, `login_controller.py` (selected), `parse.py`, `pdf_controller.py`, `registrer_controller.py`, and `report_controller.py`.
- EDITOR:** The main area displays the `login_controller.py` file. The code defines a `try_login` function that handles login requests. It imports `Company` from `invoice.models`, `check_password` from `django.contrib.auth.hashers`, and `json_suzdal` from `..utils.util_suzdal`. The function checks if a user exists and if the password is correct. If successful, it updates the user's last visit and returns a success response. If not, it returns an error response.

```
1 from invoice.models.company import Company
2 from django.contrib.auth.hashers import check_password
3 from ..utils.util_suzdal import json_suzdal
4 from ..utils.time_suzdal import time_suzdal
5
6
7 def try_login(request):
8     try:
9         email = request.POST.get('email').strip()
10        password = request.POST.get('password').strip()
11
12        company = Company.objects.get(email=email)
13        if not check_password(password, company.password):
14            return json_suzdal({'message': 'Usuario no encontrado..', 'status': 'error'})
15
16        company.lastvisit = time_suzdal()
17        company.numvisit += 1
18        company.save()
19
20        rdata = {
21            'company_id': company.id,
22            'uid': company.uid,
23            'status': 'ok',
24        }
25
26        res = json_suzdal(rdata)
27        return res
28
29 except Exception as e:
30     return json_suzdal({'message': 'Usuario no encontrado...', 'status': 'error'})
```

Seguridad implementada

- **Contraseñas cifradas** con el algoritmo por defecto de Django (PBKDF2 con sal).
- **Validación de email único** en el modelo.
- **Respuestas JSON controladas** y sin filtración de datos sensibles.
- **Frontend limpio de datos persistentes confidenciales** (aunque se guarda localmente para comodidad, no se muestra en pantalla).

La autenticación de usuarios en FacturaApp ha sido implementada de forma segura y eficaz. La arquitectura basada en Django permite separar claramente la lógica de registro e inicio de sesión. El sistema ya está preparado para escalar e incluir capas de seguridad adicionales como tokens, sesiones o autenticación con múltiples factores si fuera necesario en versiones futuras.

3. Desarrollar una de las tareas del Back-end completa.

La funcionalidad desarrollada completamente ha sido la **creación de una factura**, que representa el núcleo del funcionamiento de **FacturaApp**. Este proceso requiere interacciones entre múltiples entidades del sistema y lógica de negocio compleja, cubriendo desde la entrada de datos hasta el almacenamiento, validación, cálculo y generación de documentos.

Vista frontal desde donde se genera la factura

A través de la interfaz de usuario alojada en:

<https://facturapps.web.app/dashboard/#CreateInvoice>

.

FacturaApp Crear Factura

+ Artículo + Cliente + Factura

Tipo Factura

Tipo Ordinaria/Rectificativa/Abono
Factura Ordinaria

Datos cliente y tipo de factura

Número cliente 1 NIF CIF cliente IES Peñacastillo
Razón o denominación social IES Peñacastillo

Datos vehículo

Matrícula
Marca/Modelo/Kilómetros

Líneas de factura

Código Art.	Descripción Art.	Cantidad	Precio	% Dto.	Importe	IVA
1	Desarrollo App Personalizada	1	1000,00	0	1000,00	21 %
2	Informe Power BI	1	3000,00	0	3000,00	21 %
	P Informe Power BI Desarrollo App Personalizada	1		0	0,00	21 %
+ Línea						
	Mano de obra	10	22,00		220,00	21 %

Subtotal: 4220.00
IVA: 886.20
Total: 5106.20

Observaciones

Crear Factura

el usuario accede a un completo formulario con los siguientes bloques:

- Tipo de factura y ejercicio
- Datos del cliente
- Datos del vehículo (opcional)
- Líneas de factura (artículos, cantidades, precios, IVA, descuentos)
- Observaciones
- Cálculo en tiempo real del subtotal, IVA y total

Modelo de datos implicado

Para poder emitir una factura válida, deben existir:

1. Empresa registrada (Company)

facturapps.web.app/dashboard/#MyCompany

FacturaApp

Mi Empresa

+ Artículo+ Cliente+ Factura

Denominación

Razón o denominación social

Factura App SLNE

Nombre y apellidos persona

Suzdalenko Alexey

Email cara cliente

alexey.suzdalenko@gmail.com

CIF-NIF

A123456

Email login

alexey.suzdalenko@gmail.com

Dirección

País

España

Provincia

Cantabria

Código postal

39694

Cuidad

Santa Maria de Cayon

Dirección

El Traguezo 66

Teléfono principal

+34657666135

Teléfono

+34657666135

Precio

Precio €/hora mano de obra

22,00

Guardar

2. Cliente registrado (Customer)

facturapps.web.app/dashboard/#ShowClient-1

FacturaApp

Cliente IES Peñacastillo

+ Artículo+ Cliente+ Factura

Denominación

Código cliente

1

CIF NIF

IES Peñacastillo

Razón o denominación social

IES Peñacastillo

Nombre y apellidos persona

IES Peñacastillo

Email

IES@gmail.com

Teléfono

942 32 16 50

Dirección

País

España

Provincia

Cantabria

Código postal

39011

Cuidad

Peñacastillo

Dirección

C. Eduardo García

Guardar

3. Artículos disponibles (Article)

facturapps.web.app/dashboard/#Article

FacturaApp

Artículos

+ Artículo+ Cliente+ Factura

Total artículos 2

Número	Descripción	Precio unidad	IVA	Tipo
2	Informe Power BI	3000.00 €	21.00 %	norm
1	Desarrollo App Personalizada	1000.00 €	21.00 %	norm

4. Documento contador (Document) para numerar automáticamente las facturas

Además, la factura (Factura) se compone de varias líneas (Facturalineas), que recogen cada ítem facturado y sus valores.

Lógica del backend: `invoice_controller.invoice_actions`

Cuando el usuario pulsa el botón **Crear Factura**, el frontend envía los datos en formato JSON al endpoint:

```
path('invoice/create/0', invoice_controller.invoice_actions)
```

La función correspondiente realiza los siguientes pasos:

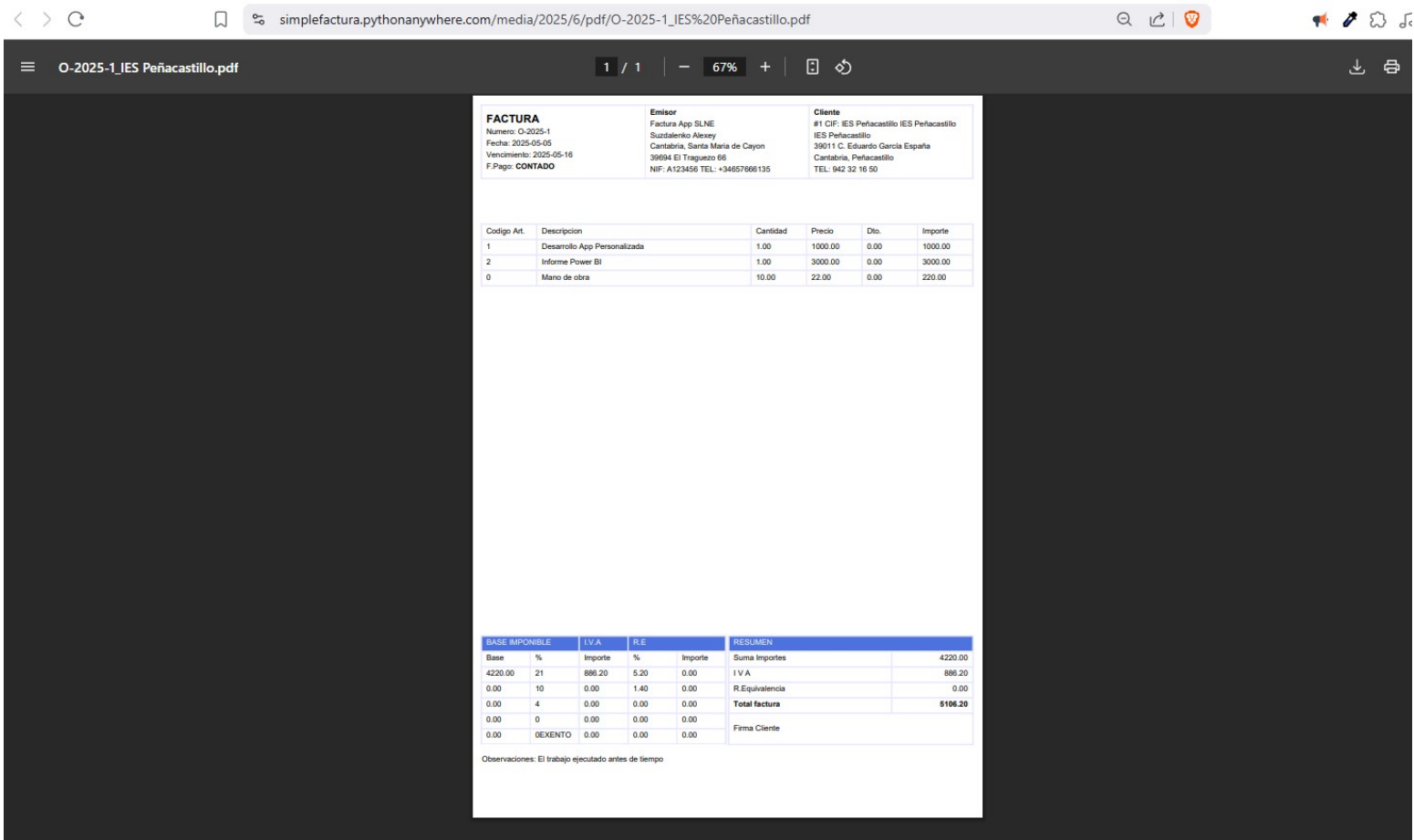
1. **Verifica si el usuario está autenticado** usando `user_auth(request, data)`
2. **Extrae datos** como tipo de factura, cliente, líneas, desglose de IVA, vehículo, observaciones, etc.
3. **Valida la existencia del cliente y artículos**
4. **Crea la factura (Factura)**, asignando un número único de serie a través del modelo `Document`
5. **Calcula automáticamente:**
 - Subtotal: base imponible por línea
 - Descuentos por línea
 - Valor de IVA por tipo
 - Total final con impuestos
6. **Guarda cada línea de la factura** (`Facturalineas`)
7. **Guarda los datos del vehículo**, si se indican
8. **Registra la factura en un hilo separado**, por si se desea generar un PDF u otro documento (función `wr_invoice_in_thread`)
9. Devuelve un JSON al frontend con `status: ok`, el ID de la factura y un mensaje

Podemos visualizar la factura creada:



.

y en PDF:



Código python de creación de la factura:

```
from invoice.models.article import Article

from invoice.models.company import Company
from invoice.models.customer import Customer
from invoice.models.document import Document
from invoice.models.factura import Factura
from invoice.utils.time_suzdal import current_date, fecha_expedicion, get_time_11days,
wr_invoice_in_thread, wr_invoice_to_file
from invoice.utils.vehicle_func import get_or_save_vehicle
from mysite import settings
from ..utils.util_suzdal import factura_new_article, factura_new_lines, json_suzdal, user_auth
import json, os
from datetime import datetime

def invoice_actions(request, action, id):
    if request.body:
        try:
            data = json.loads(request.body.decode('utf-8'))
        except json.JSONDecodeError:
            return json_suzdal({'status': 'error', 'message': 'Cuerpo de la solicitud no es JSON válido'})
        else:
            return json_suzdal({'status': 'error', 'message': 'Cuerpo de la solicitud vacío'})

    auth_status, company = user_auth(request, data)
    if auth_status is None or company is None:
        return json_suzdal({'login': False, 'status': 'error', 'message': 'Usuario no esta logeado'})
```

```

desglose      = data['desglose']
ejercicio     = str(datetime.now().strftime('%Y')).strip()
tipo_factura  = str(data['factura']['tipo_factura']).strip()
lineas        = data['lineas']
if len(lineas) == 0:
    return json_suzdal({'status':'error', 'message':'Factura sin lineas'})

try:
    customer = Customer.objects.get(id=data['cliente']['clientIdDeveloper'],
clientcode=data['cliente']['clientNumber'], company_id=company['id'])
    factura = Factura.objects.create(company_id=company['id'], tipo_factura=tipo_factura,
ejercicio=ejercicio)
    document = Document.objects.filter(company_id=company['id'], description=tipo_factura,
ejercicio=ejercicio).values('value').first()
    factura.name_factura      = str(data['factura']['name_factura']).strip()
    factura.apunta_factura    = str(data['factura']['apunta_factura']).strip()
    factura.numero            = document['value']
    factura.serie_fact        = f"{tipo_factura}-{ejercicio}-{factura.numero}"
    factura.serie_fact_unique = f"{tipo_factura}-{ejercicio}-{factura.numero}-
{company['id']}"
    factura.fecha_expedicion  = fecha_expedicion()
    factura.vencimiento       = get_time_11days()

    factura.customer_id       = customer.id
    factura.customer_num       = customer.clientcode
    factura.receptor_company_name = customer.razon

    SUBTOTAL_FACTURA = 0
    IMP_IVAS_FACTURA = 0
    TOTAL_FACTURA     = 0
    LINEAS_FACTURA    = []

    # Base Imponible = Precio del artículo × Cantidad de artículos
    for linea in lineas:
        description = str(linea.get('description', 'none')).strip()
        idArticle1  = str(linea.get('idArticle1', '')).strip()
        precio1     = float(linea.get('precio1', 0))
        cantidad1   = float(linea.get('cantidad1', 0))
        descPorc    = float(linea.get('descPorc', 0))
        ivaPorcent  = float(linea.get('ivaPorcent', 0))
        ivaTypeStr  = str(linea.get('ivaType', '0'))

        if idArticle1.isdigit(): # Comprobar si es un número válido
            articulo_current = Article.objects.filter(id=idArticle1,
company_id=company['id']).first()
        else:
            art_created, articulo_current = factura_new_article(description, company['id'],
precio1, ivaTypeStr, ivaPorcent)
            if art_created is None or articulo_current is None:
                return json_suzdal({'status':'error', 'message':'Error al crear artículo nuevo'})

    importe_inicio      = cantidad1 * precio1
    valor_descuento     = descPorc / 100 * importe_inicio
    importe_con_descuento = importe_inicio - valor_descuento
    SUBTOTAL_FACTURA   += importe_con_descuento
    valor_iva           = ivaPorcent / 100 * importe_con_descuento
    IMP_IVAS_FACTURA    += valor_iva

```

```
importe_final      = importe_con_descuento + valor_iva
TOTAL_FACTURA      += importe_final
```

```
for d in desglose:
    if str(d['iva']) == ivaTypeStr:
        d['base_imponible'] += importe_con_descuento
        d['valor_iva'] += valor_iva
        d['total_con_iva'] += d['base_imponible'] + d['valor_iva']
```

```
linea_factura = {'invoice_id':0, 'serie': '', 'company_id':company['id'],
'article_id':articulo_current.id, 'article_num':articulo_current.artcode,
'article_name':articulo_current.description, 'cantidad':cantidad1, 'precio':precio1,
'descuento':descPorc, 'iva_porcent':ivaPorcent, 'iva_type':ivaTypeStr}
LINEAS_FACTURA.append(linea_factura)
```

ahora el calculo de mano de obra

```
canridadManoObra = float(data['manoObra']['canridadManoObra'])
precioManoObra   = float(data['manoObra']['precioManoObra'])
descManoObr      = float(data['manoObra']['descManoObr'])
ivaPorcentManoOb = float(data['manoObra']['ivaPorcentManoOb'])
tipoIvaManoObra  = str(data['manoObra']['tipoIvaManoObra'])
```

cuando existe mano de obra

```
if canridadManoObra > 0 and precioManoObra > 0:
    importe_inicio_mo      = canridadManoObra * precioManoObra
    valor_descuento_mo     = descManoObr / 100 * importe_inicio_mo
    importe_con_descuento_mo = importe_inicio_mo - valor_descuento_mo
    SUBTOTAL_FACTURA      += importe_con_descuento_mo
    valor_iva_mo           = ivaPorcentManoOb / 100 * importe_con_descuento_mo
    IMP_IVAS_FACTURA       += valor_iva_mo
    importe_final_mo       = importe_con_descuento_mo + valor_iva_mo
    TOTAL_FACTURA         += importe_final_mo
    linea_factura = {'invoice_id':0, 'serie': '', 'company_id':company['id'], 'article_id':0,
'article_num':0, 'article_name':'Mano de obra', 'cantidad':canridadManoObra, 'precio':precioManoObra,
'descuento':descManoObr, 'iva_porcent':ivaPorcentManoOb, 'iva_type':tipoIvaManoObra}
    LINEAS_FACTURA.append(linea_factura)
```

```
for d in desglose:
    if str(d['iva']) == tipoIvaManoObra:
        d['base_imponible'] += importe_con_descuento_mo
        d['valor_iva'] += valor_iva_mo
        d['total_con_iva'] += d['base_imponible'] + d['valor_iva']
```

```
factura.ivas_desglose = json.dumps(desglose)
factura.subtotal      = SUBTOTAL_FACTURA
factura.importe_ivas  = IMP_IVAS_FACTURA
factura.total         = TOTAL_FACTURA
factura.total2        = SUBTOTAL_FACTURA + IMP_IVAS_FACTURA
factura.observacion   = str(data['observaciones']['obstextareaid']).strip()[251]
factura.save()
```

pongo a las lineas de iva ID de la factura

```
for linea_fac in LINEAS_FACTURA:
    linea_fac['invoice_id'] = factura.id
    linea_fac['serie']      = factura.serie_fact_unique
```

```
factura_new_lines(LINEAS_FACTURA)
```

```

inputVehicleMatricula = str(data['vehicle']['inputVehicleMatricula']).strip()
inputVehicleMarca      = str(data['vehicle']['inputVehicleMarca']).strip()
if inputVehicleMatricula != '' and len(inputVehicleMatricula) > 3:
    get_or_save_vehicle(factura.id, company['id'], customer.id, inputVehicleMatricula,
inputVehicleMarca)

wr_invoice_in_thread(data, factura.serie_fact, customer.cif_nif)
if factura.id > 0:
    pass
else:
    return json_suzdal({'status':'error', 'message':'Fallo al crear factura'})
except Exception as e:
    wr_invoice_in_thread(data, factura.serie_fact, customer.cif_nif)
    return json_suzdal({'status':'error', 'message':str(e)})

rdata = {
    'factura_id': factura.id,
    'status': 'ok',
    'message': 'Factura creada '
}

return json_suzdal(rdata)

```

Gestión del número de factura automático

Cada vez que se crea una factura (o cliente o artículo), se actualiza automáticamente el contador correspondiente:

```

from django.db.models.signals import post_save
from django.dispatch import receiver
from .models import Article, Document, Customer, Factura

# añadido 1 al crear un articulo para una empresa determinada
@receiver(post_save, sender=Article)
def actualizar_document(sender, instance, created, **kwargs):
    if created:
        document, _ = Document.objects.get_or_create(description='articulo_numero',
company_id=instance.company_id)
        document.value += 1
        document.save()

# observer a la hora de crear un CLIENTE NUEVO actualizo el numero de clientes en la base datos
@receiver(post_save, sender=Customer)
def actualizar_document(sender, instance, created, **kwargs):
    if created:
        document, _ = Document.objects.get_or_create(description='cliente_numero',
company_id=instance.company_id)
        document.value += 1
        document.save()

```

```
# observer a la hora de crear una Factura nueva actualizo el numero Factura en la base datos
```

```
@receiver(post_save, sender=Factura)
```

```
def actualizar_document(sender, instance, created, **kwargs):
```

```
    if created:
```

```
        document, _ = Document.objects.get_or_create(description=instance.tipo_factura,
```

```
company_id=instance.company_id, ejercicio=instance.ejercicio)
```

```
        document.value += 1
```

```
        document.save()
```

La funcionalidad de creación de facturas ha sido desarrollada completamente, integrando modelos relacionados, lógica de negocio real, validaciones, persistencia y gestión automática de numeración. Este punto demuestra la capacidad del backend de **FacturaApp** para manejar procesos administrativos completos y coherentes desde una interfaz moderna y funcional.

Back-end-2

1. Desarrollar resto tareas del Back-end.

El desarrollo del backend de **FacturaApp** no se limita a una única operación. Para que el proceso de facturación sea funcional, han sido necesarias múltiples tareas complementarias, que conforman un sistema interconectado y completo. Estas tareas han sido desarrolladas en su totalidad, desde la definición de modelos hasta los controladores y endpoints funcionales.

Gestión de los datos de la empresa (Company)

Antes de emitir una factura, es imprescindible que el usuario haya registrado su empresa. Se ha desarrollado la lógica que permite:

- Crear o actualizar la información de la empresa.
- Asignarle un UID y estado activo.
- Registrar fecha de alta y visitas.
- Validar que el CIF y correo sean únicos.

Esta información es fundamental para completar los campos del emisor en la factura.

Registro y gestión de clientes (Customer)

También se ha implementado toda la lógica necesaria para:

- Crear un cliente con datos fiscales (NIF, razón social, dirección...).
- Asociar cada cliente a una empresa.
- Validar duplicados mediante código y NIF.
- Aumentar automáticamente el contador de clientes usando señales post_save.

Este módulo permite seleccionar un cliente concreto al generar una factura.

Creación y mantenimiento de artículos (Article)

Para facturar productos o servicios, el sistema permite:

- Registrar artículos con descripción, precio e IVA.
- Crear artículos automáticamente desde el formulario de factura.
- Asociarlos a una empresa específica.
- Controlar su numeración con el modelo Document.

Gracias a esto, las líneas de la factura pueden completarse con artículos ya guardados.

Generación del documento PDF de la factura

Una vez creada una factura, se ofrece al usuario la posibilidad de visualizarla o imprimirla como PDF. Para ello:

- Se ha desarrollado un módulo de generación PDF en Python (pdf_controller).
- Se obtiene la factura, sus líneas y los datos del cliente y emisor.
- Se renderiza una plantilla o se genera el documento desde cero.
- El archivo se guarda temporalmente o se devuelve al usuario para descarga.

Esta funcionalidad da al sistema un valor real para su uso comercial o profesional.

Coordinación de todas las tareas mediante controladores

Todas las funcionalidades del backend están organizadas mediante **controladores por entidad**, como:

- default_controller.py - operaciones CRUD genéricas
- invoice_controller.py - lógica de facturación
- pdf_controller.py - generación de PDF
- report_controller.py - informes y resúmenes

Esto garantiza que cada parte del sistema sea modular, escalable y fácilmente mantenible.

Desarrollar el backend completo de FacturaApp ha implicado mucho más que la creación de una factura. Se han implementado todas las funcionalidades necesarias para que el sistema sea útil en un entorno real, asegurando que los datos estén validados, organizados, enlazados y disponibles en formatos digitales como PDF. Todo esto convierte a FacturaApp en una solución robusta, funcional y lista para ser utilizada por autónomos y pequeñas empresas.

Pruebas

1. Realización de pruebas y ajustes finales.

Una vez finalizado el desarrollo de los distintos módulos del backend y frontend de **FacturaApp**, se ha llevado a cabo un proceso de pruebas funcionales completas para garantizar que la aplicación responde de forma correcta ante diferentes escenarios reales de uso. Estas pruebas se han centrado tanto en el flujo de trabajo como en la validación de datos, errores comunes y coherencia visual.

Pruebas realizadas

Registro y login

- Se verificó que un usuario puede crear una cuenta con datos válidos.
- Se comprobó que el sistema rechaza registros con datos incorrectos (email mal formado, contraseñas distintas...).
- Se probó el inicio de sesión con credenciales válidas e inválidas.

Gestión de datos

- Se realizaron pruebas de inserción y edición de:
 - Datos de empresa
 - Clientes
 - Artículos
- Se validó que cada entidad se asocia correctamente con el usuario autenticado.

Facturación

- Se probó la creación de facturas con:
 - Artículos existentes

- Artículos creados sobre la marcha
- Clientes válidos y con diferentes tipos de IVA
- Se revisó el cálculo automático de totales, IVA, descuentos y la lógica de desglose.

PDF

- Se probó la generación del PDF de factura tras su creación.
- Se verificó que los datos mostrados en el documento coinciden con los introducidos.

Flujo general

- Se simularon procesos completos desde el registro hasta la creación y exportación de una factura.
- Se validaron rutas protegidas, comportamiento sin conexión y mensajes de error.

Ajustes finales realizados

- Mejora de estilos CSS para coherencia visual entre secciones.
- Control de errores en frontend para mostrar mensajes claros al usuario.
- Limpieza del código JavaScript y mejora de funciones modulares.
- Validación extra en backend para campos críticos (precios, cantidades, IDs).
- Ajustes en el sistema de numeración automática de facturas.

Conclusión

El proceso de pruebas ha permitido asegurar que **FacturaApp funciona correctamente** desde el punto de vista técnico y de usuario. Todos los errores detectados han sido corregidos, lo que garantiza una experiencia fluida, estable y profesional para el usuario final. La aplicación ha sido desplegada y es totalmente operativa desde su versión web.

Web de la aplicación: <https://facturapps.web.app/>

Git de frontend: <https://github.com/suzdalenko-dev/FacturApp/tree/main/front-html/raw-html>

Git de backend: <https://github.com/suzdalenko-dev/backend-facturapps>

Hosting de frontend: <https://firebase.google.com/>

Hosting de backend: <https://www.pythonanywhere.com/>