15ª Emisión

DIPLOMADO

Desarrollo de Sistemas con Tecnología Java

Módulo 5-6

Desarrollo de aplicaciones empresariales con Jakarta EE

Uriel Hernández

DGTIC

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
Dirección General de Cómputo y de Tecnologías de información y Comunicación
Dirección de Docencia en TIC

50 AÑOS
Educación Continua
1971 - 2021

Pixup Faces WebApp

webpage - Faces

webpage - Faces

PIXUP

Registro Usuario

Datos Personales

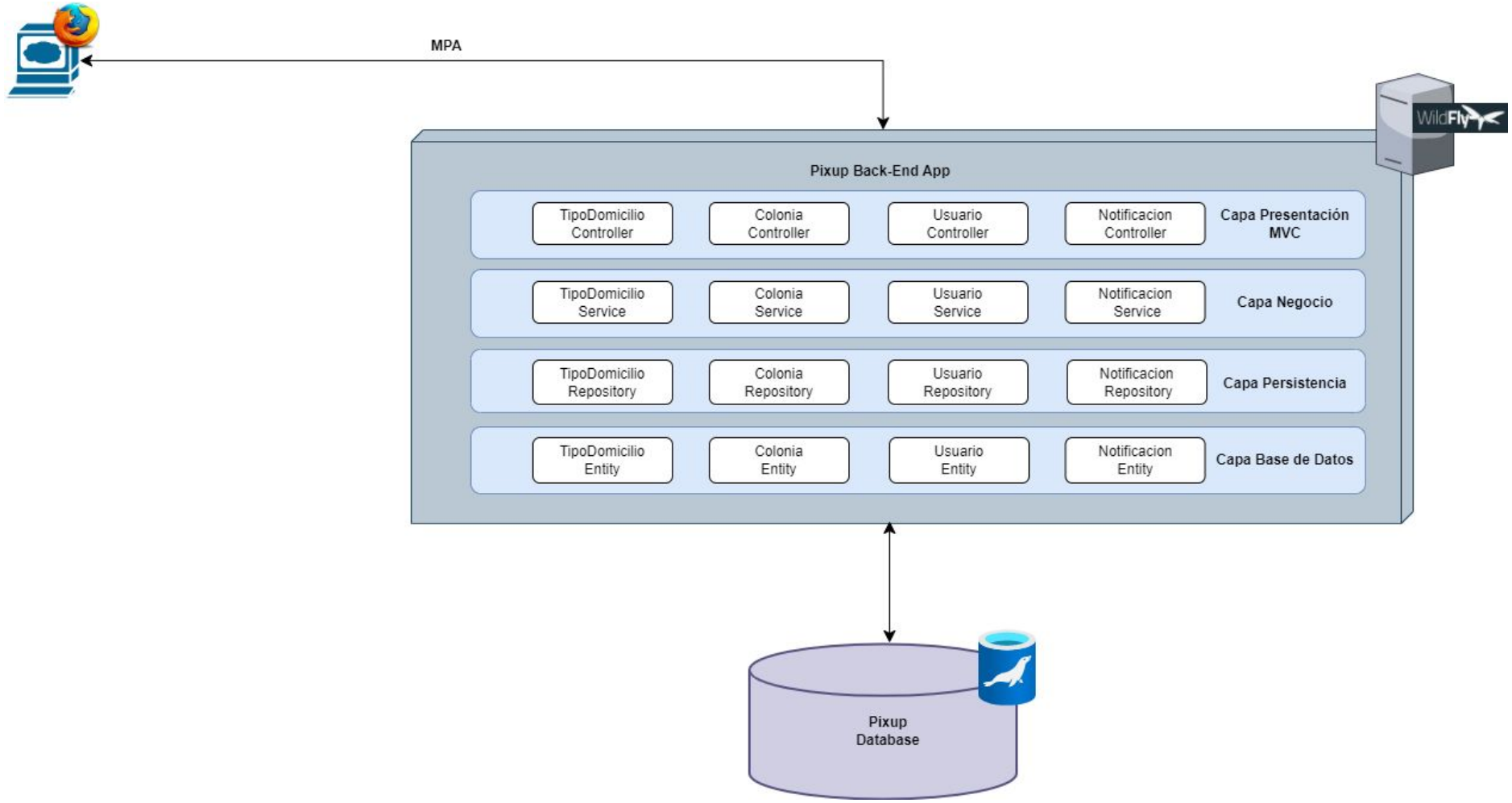
Nombre: Pedro
Apellido Paterno: Suárez
Apellido Materno: Orozco
Password: ••••••••
Email: pedro@gmail.com
RFC: TAER901010TR1

Domicilio

Calle: Lago Cuitzeo
No. Exterior: 65
No. Interior: A301
Código Postal: 11000
Buscar Colonia
Colonia: Lomas de Chapultepec II Sección-Miguel Hidalgo-CIUDAD DE MÉXICO
Tipo Domicilio: ○ Entrega ◉ Facturacion
Crear Cuenta

webpage - Faces

Pixup Capas – Faces



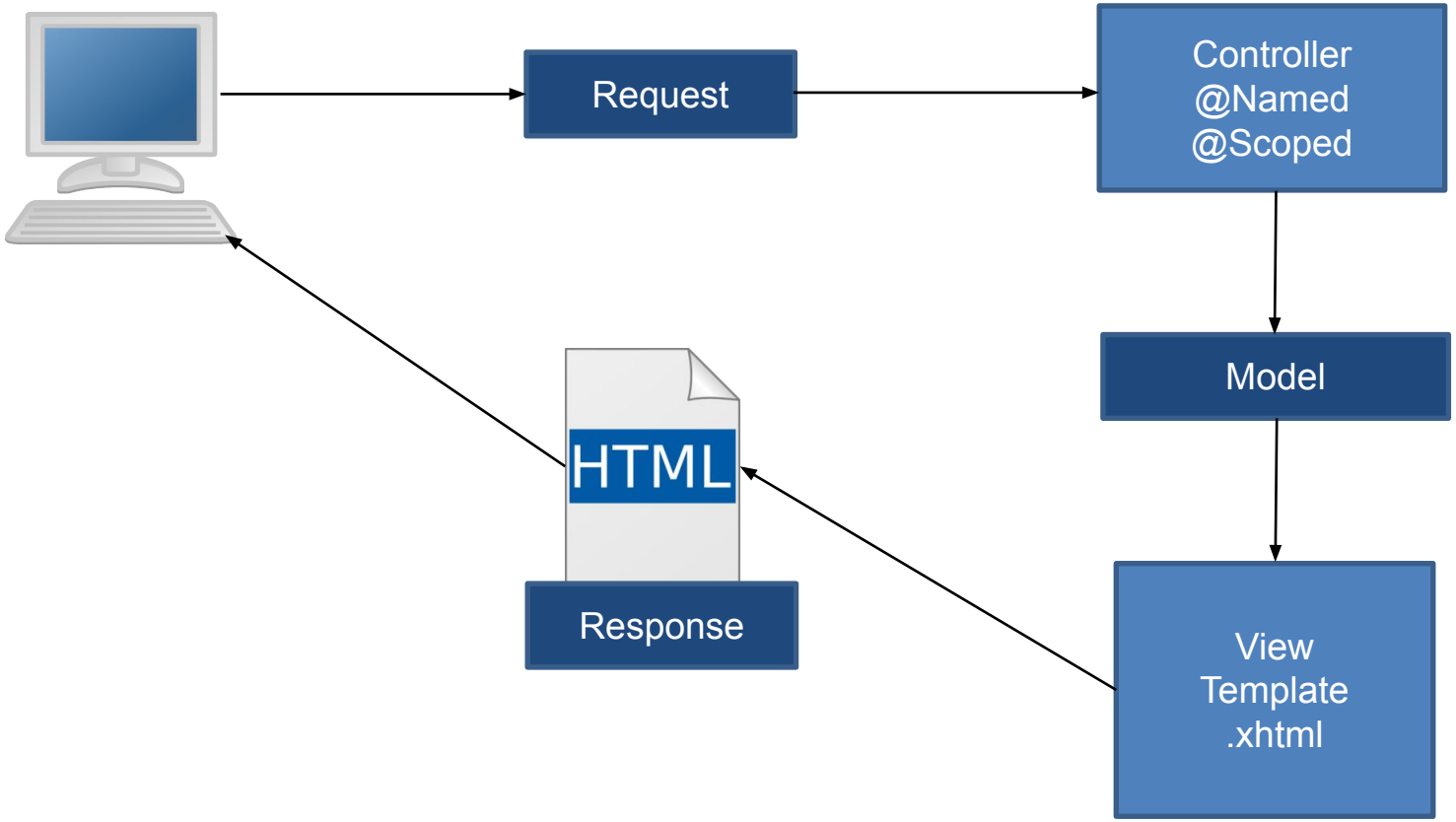
Jakarta Faces

- Fue creado para estandarizar el patrón de diseño MVC (Model-View -Controller) en la plataforma Jakarta EE.
- Originalmente se creó como una alternativa al framework Apache Struts que era muy popular para el desarrollo de Front Ends en java y el cual implementa el patrón MVC.
- Un par de semanas después de crearse la especificación Jakarta Faces 1.0 la versión Spring 1.0 fue liberada, Spring a la postre se consolidaría como uno de los frameworks web más importantes en java con Spring MVC.
- Existe una larga historia de implementaciones de esta especificación: MyFaces, RichFaces, PrimeFaces, Mojarra.

MVC (Model-View-Controller)

- Model: Representación del modelo de dominio. Se puede utilizar la capa del modelo de dominio persistente o alguna representación alternativa (para no acoplarse a esta definición), como por ejemplo con el uso de DTOs (Forms).
- View: Resultado que se renderea al usuario por medio del navegador en respuesta a una petición. Las vistas se definen por medio de templates .xhtml en las que se definen los elementos de la página por medio de librerías de tags (tag libraries). Se procesa el archivo .xhtml por medio del template engine y se devuelve el contenido en .html
- Controller: Se encarga de gestionar las peticiones web del usuario y responder con algún tipo de información. Para definir un controller en la capa MVC se utilizan las anotaciones @Named junto con alguna anotación de Scope (@RequestScoped, @ViewScoped, @SessionScoped).

MVC - Flujo Request-Response



MPA (Multi Page Application)

- Aplicaciones web (web apps) que tienen más de una página.
- Cargan la página cada vez que un usuario envía información o cuando selecciona un link.
- Requieren transferir datos entre el servidor y el browser por cada petición, el servidor genera al final una nueva página html que es enviada al cliente para continuar con la navegación en la web app.
- Se pueden utilizar APIs javascript tales como XMLHttpRequest y Fetch (AJAX) cuando se requiere mostrar diferente contenido, logrando incrementar el performance y generar una experiencia más dinámica.

web.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="https://jakarta.ee/xml/ns/jakartaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
         https://jakarta.ee/xml/ns/jakartaee/web-app_6_0.xsd"
         version="6.0">

    <context-param>
        <param-name>jakarta.faces.INTERPRET_EMPTY_STRING_SUBMITTED_VALUES_AS_NULL</param-name>
        <param-value>true</param-value>
    </context-param>
    <context-param>
        <param-name>jakarta.faces.AUTOMATIC_EXTENSIONLESS_MAPPING</param-name>
        <param-value>true</param-value>
    </context-param>

    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.xhtml</url-pattern>
    </servlet-mapping>

</web-app>
```

FacesActivator

```java
package unam.diplomado;

import jakarta.enterprise.context.ApplicationScoped;
import jakarta.faces.annotation.FacesConfig;

@FacesConfig
@ApplicationScoped
public class FacesActivator {

}
```

RegistroUsuarioController

```java
@Named("registro")
@ViewScoped
@Log
public class RegistroUsuarioController implements Serializable {

    private Usuario usuario;
    private Domicilio domicilio;
    private String cp;
    private String coloniaId;
    private String tipoDomicilioId;
    private Collection<ColoniaDTO> colonias;
    private Collection<TipoDomicilio> tiposDomicilio;

    @Inject
    transient private ColoniaService coloniaService;
    @Inject
    transient private UsuarioService usuarioService;
    @Inject
    transient private ColoniaRepository coloniaRepository;
    @Inject
    transient private ColoniaMapper coloniaMapper;
    @Inject
    transient private TipoDomicilioRepository tipoDomicilioRepository;
```

registro.xhtml

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="jakarta.faces.core"
      xmlns:h="jakarta.faces.html">
<f:view>
    <h:head>
        <title>Pixup</title>
    </h:head>
    <h:body>
        <h:graphicImage value="resources/images/pixup_logo.png"/>
        <br/>
        <h:graphicImage value="resources/images/pixup_registro_usuario.png"/>
        <br/>
        <h:graphicImage value="resources/images/pixup_datos_personales.png"/>
```

registro.xhtml

```
<h:form>
    <table>
        <tr>
            <td><h:outputLabel for="nombre">Nombre:</h:outputLabel></td>
            <td><h:inputText id="nombre" value="#{registro.usuario.nombre}"/></td>
        </tr>
        <tr>
            <td><h:outputLabel for="primerApellido">Apellido Paterno:</h:outputLabel></td>
            <td><h:inputText id="primerApellido" value="#{registro.usuario.primerApellido}"/></td>
        </tr>
        <tr>
            <td><h:outputLabel for="segundoApellido">Apellido Materno:</h:outputLabel></td>
            <td><h:inputText id="segundoApellido" value="#{registro.usuario.segundoApellido}"/></td>
        </tr>
        <tr>
            <td><h:outputLabel for="password">Password:</h:outputLabel></td>
            <td><h:inputSecret id="password" value="#{registro.usuario.password}"/></td>
        </tr>
        <tr>
            <td><h:outputLabel for="email">Email:</h:outputLabel></td>
            <td><h:inputText id="email" value="#{registro.usuario.email}"/></td>
        </tr>
```

registro.xhtml

```
<table>
    <tr>
        <td><h:outputLabel for="calle">Calle:</h:outputLabel></td>
        <td><h:inputText id="calle" value="#{registro.domicilio.calle}"/></td>
    </tr>
    <tr>
        <td><h:outputLabel for="numExterior">No. Exterior:</h:outputLabel></td>
        <td><h:inputText id="numExterior" value="#{registro.domicilio.numExterior}"/></td>
    </tr>
    <tr>
        <td><h:outputLabel for="numInterior">No. Interior:</h:outputLabel></td>
        <td><h:inputText id="numInterior" value="#{registro.domicilio.numInterior}"/></td>
    </tr>
    <tr>
        <td><h:outputLabel for="cp">Código Postal:</h:outputLabel></td>
        <td>
            <h:inputText id="cp" value="#{registro.cp}"/>
        </td>
    </tr>
```

registro.xhtml

```
<tr>
    <td colspan="2">
        <h:commandButton value="Buscar Colonia" action="#{registro.findColoniaByCp}">
            <f:ajax execute="cp" event="click" render="coloniaId"/>
        </h:commandButton>
    </td>
</tr>
<tr>
    <td><h:outputLabel for="coloniaId">Colonia:</h:outputLabel></td>
    <td>
        <h:selectOneMenu id="coloniaId" value="#{registro.coloniaId}">
            <f:selectItem itemValue="#{null}" itemLabel="-- Seleccionar Colonia --" />
            <f:selectItems value="#{registro.colonias}" var="colonia" itemValue="#{colonia.id}"
                           itemLabel="#{colonia.nombre}-#{colonia.municipio}-#{colonia.estado}"/>
        </h:selectOneMenu>
    </td>
</tr>
```

registro.xhtml

```
<tr>
    <td><h:outputLabel for="tipoDomicilioId">Tipo Domicilio:</h:outputLabel></td>
    <td>
        <h:selectOneRadio id="tipoDomicilioId" value="#{registro.tipoDomicilioId}">
            <f:selectItems value="#{registro.tiposDomicilio}" var="tipoDomicilio"
                           itemValue="#{tipoDomicilio.id}" itemLabel="#{tipoDomicilio.descripcion}"/>
        </h:selectOneRadio>
    </td>
</tr>
<tr>
    <td colspan="2">
        <h:commandButton value="Crear Cuenta" action="#{registro.altaUsuario}"/>
    </td>
</tr>
```

registro_resultado.xhtml

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="jakarta.faces.core"
      xmlns:h="jakarta.faces.html">
<f:view>
    <h:head>
        <title>Pixup</title>
    </h:head>
    <h:body>
        <h:graphicImage value="resources/images/pixup_logo.png"/>
        <br/>
        <h1>Registro Exitoso</h1>
    </h:body>
</f:view>
</html>
```

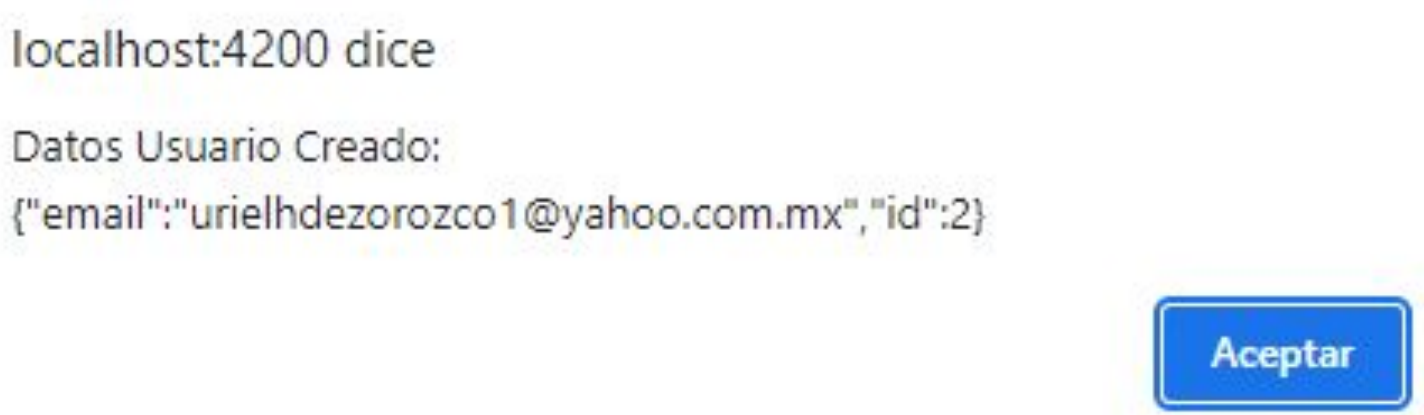
CharacterEncodingFilter

```java
package unam.diplomado;

import jakarta.servlet.*;
import jakarta.servlet.annotation.WebFilter;

import java.io.IOException;

@WebFilter("/*")
public class CharacterEncodingFilter implements Filter {

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
                         FilterChain chain) throws ServletException, IOException {
        request.setCharacterEncoding("UTF-8");
        chain.doFilter(request, response);
    }

}
```

Pixup Angular WebApp

webpage - Angular

webpage - Angular



Angular

Angular es una plataforma de desarrollo construida en TypeScript. Incluye

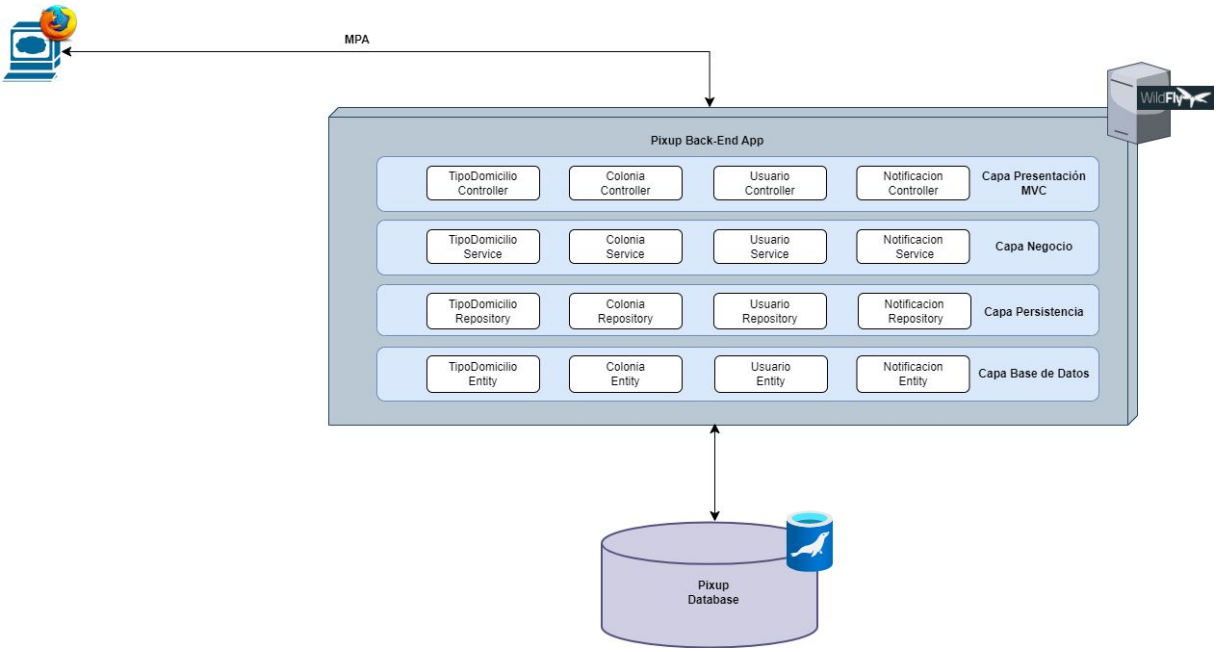
lo siguiente:

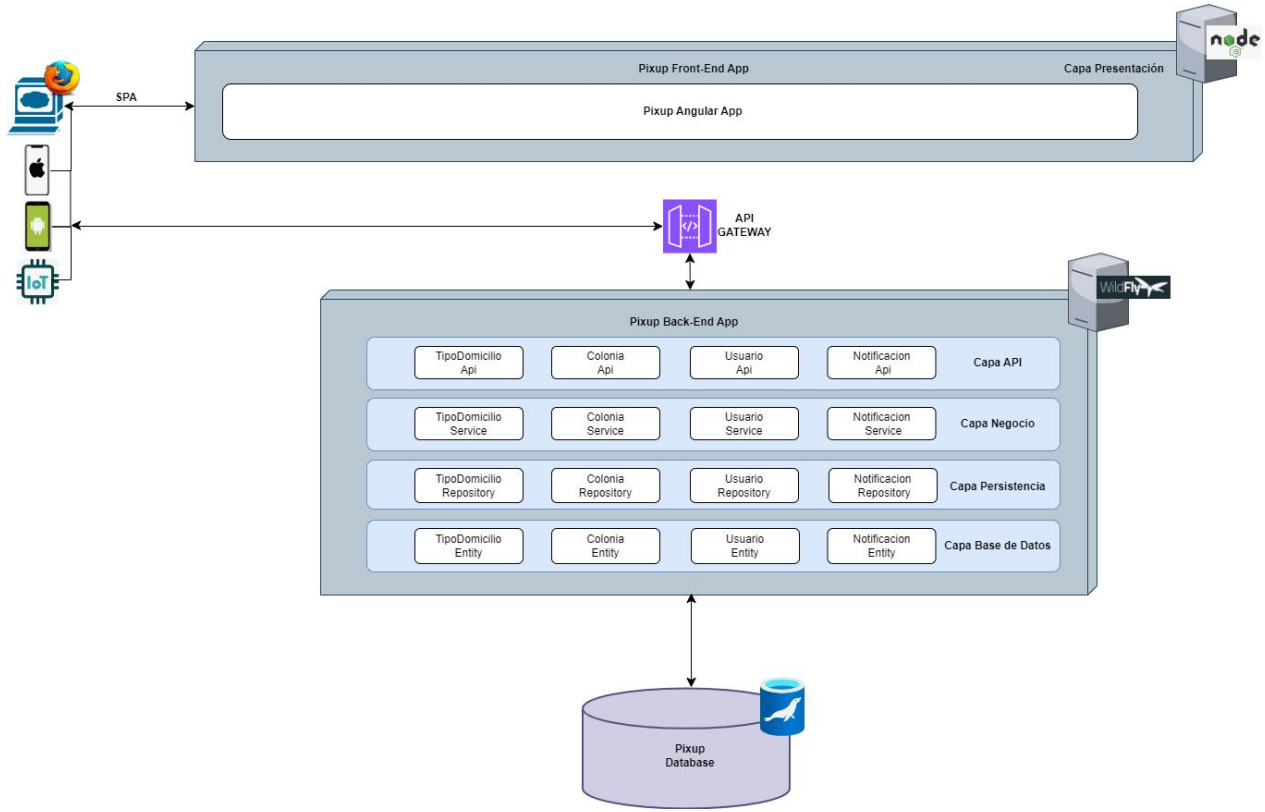
- Un framework basado en componentes para construir aplicaciones web escalables.
- Una colección de librerías bien integradas que cubren una variedad de funcionalidades, tales como: ruteo, administración de formas, comunicación cliente-servidor.
- Una suite de herramientas para el desarrollador, que le permiten desarrollar, construir, probar y actualizar el código.

SPA (Single Page Application)

- Una aplicación de una sola página (SPA) es una implementación de una web app que se carga en un único documento web, y luego actualiza el contenido del cuerpo (body) de ese único documento por medio de APIs javascript tales como XMLHttpRequest y Fetch cuando se requiere mostrar diferente contenido.
- Esto permite a los usuarios usar sitios web sin tener que cargar nuevas páginas enteras desde el servidor lo que puede resultar en un incremento de performance y en una experiencia más dinámica.

Comparación Faces vs Angular





app.component.ts

```
ngOnInit(): void {
  this.httpClient.get<TipoDomicilio[]>(
    'http://localhost:8080/pixup/api/tiposDomicilio')
    .subscribe(tiposDomicilio => this.tiposDomicilio = tiposDomicilio);
}

findColoniaByCp(event: Event): void {
  this.colonia.id = undefined;
  this.colonia.municipio = undefined;
  this.colonia.estado = undefined;
  this.httpClient.get<Colonia[]>(
    'http://localhost:8080/pixup/api/colonias?cp=' + this.colonia.cp)
    .subscribe(colonias => this.colonias = colonias);
}
```

app.component.ts

```
submitForm(form: NgForm): void {
  this.domicilio.colonia = this.colonia.id;
  this.domicilio.tipoDomicilio = this.tipoDomicilio.id;
  this.registroUsuarioRequest.domicilio = this.domicilio;
  this.registroUsuarioRequest.usuario = this.usuario;
  console.log('RegistroUsuarioRequest: ' +
    JSON.stringify(this.registroUsuarioRequest));

  this.httpClient.post<Usuario>(
    'http://localhost:8080/pixup/api/usuarios/registro',
    this.registroUsuarioRequest, {
      headers: new HttpHeaders()
        .set('Content-type', 'application/json')
        .set('Accept', 'application/json')})
    .subscribe(usuarioCreado =>
      alert('Datos Usuario Creado: ' + JSON.stringify(usuarioCreado)));

  form.reset();
}
```

app.component.html

```
<form novalidate #form="ngForm" (ngSubmit)="submitForm(form)">
  <table>
    <tr>
      <td><label>Nombre:</label></td>
      <td><input name="nombre" [(ngModel)]="usuario.nombre" /></td>
    </tr>
    <tr>
      <td><label>Apellido Paterno:</label></td>
      <td><input name="primerApellido" [(ngModel)]="usuario.primerApellido" /></td>
    </tr>
    <tr>
      <td><label>Apellido Materno:</label></td>
      <td><input name="segundoApellido" [(ngModel)]="usuario.segundoApellido" /></td>
    </tr>
    <tr>
      <td><label>Password:</label></td>
      <td><input type="password" name="password" [(ngModel)]="usuario.password" /></td>
    </tr>
    <tr>
      <td><label>Email:</label></td>
      <td><input name="email" [(ngModel)]="usuario.email" /></td>
    </tr>
    <tr>
      <td><label>RFC:</label></td>
      <td><input name="rfc" [(ngModel)]="usuario.rfc" /></td>
    </tr>
  </table>
```

CORS (Cross-Origin Resource Sharing)

- Es un mecanismo que permite acceder recursos restringidos entre dominios distintos.
- Por ejemplo: una página web que requiere consumir un servicio REST (resource) el cual se encuentra disponible en un dominio diferente en el que fue generada la página.
- Se considera el mismo origen (SOP) cuando ambas aplicaciones/recursos tienen idénticos valores para:
  - Protocolo
  - Puerto
  - Host

CORSFilter

```java
@Provider
public class CORSFilter implements ContainerResponseFilter {

    @Override
    public void filter(
            ContainerRequestContext containerRequestContext,
            ContainerResponseContext containerResponseContext)
            throws IOException {
        containerResponseContext.getHeaders().add(
            "Access-Control-Allow-Origin", "*");
        containerResponseContext.getHeaders().add(
            "Access-Control-Allow-Credentials", "true");
        containerResponseContext.getHeaders().add(
            "Access-Control-Allow-Headers",
            "origin, content-type, accept, authorization");
        containerResponseContext.getHeaders().add(
            "Access-Control-Allow-Methods",
            "GET, POST, PUT, DELETE, OPTIONS, HEAD");
    }

}
```

Contacto

Uriel Hernández

Solution Architect

urielhdezorozco@yahoo.com.mx

Redes sociales:

https://www.linkedin.com/in/juho-mex