



Juntos transformemos
Yucatán
GOBIERNO ESTATAL 2018 · 2024

SAF
SECRETARÍA DE
ADMINISTRACIÓN
Y FINANZAS



LINEAMIENTOS DE TRABAJO WEB



CONTENIDO

CONTENIDO	2
INTRODUCCIÓN	4
SEGURIDAD WEB	5
CODIFICACIÓN	6
CODIFICACIÓN FÍSICA DE LOS ARCHIVOS FUENTE	6
UTF-8 (SIN BOM) (Importante)	6
Fin de Línea (LF) (Deseable)	6
RECOMENDACIONES DE PROGRAMACIÓN	6
HTML y CSS	6
Nombres de archivos HTML	6
RESPONSIVIDAD	7
Librerías CSS para responsividad	7
RECOMENDACIONES GENERALES DE HTML5	8
RECOMENDACIONES GENERALES DE CSS	9
Nombres de archivos CSS	9
Codificación	9
Modos de uso CSS	12
LENGUAJES DE PROGRAMACIÓN WEB	12
Nombres de archivos de código	13
Codificación	13
Estructura de carpetas y códigos WEB	14
HERRAMIENTAS PARA MANEJO DE PAQUETES	16
COMPOSER PARA PHP	16
Cómo utilizarlo	16
CLI de composer	18
Instalación de paquetes	18
Desinstalación de paquetes	19
Instalación y actualización de paquetes incluidos en composer.json	20
BOWER PARA JAVASCRIPT ES5	20
Requerimientos previos	21
Instalación	21
Instalación de paquetes	21
Desinstalar paquetes	23
Actualizar paquetes	23

NPM PARA FRONTENDS ES6 O BACKEND NODEJS..... 23

Instalación	24
Instalación de paquetes	24
Desinstalar paquetes	26
Actualizar paquetes	26
Capacidades de ES6.....	26

REPOSITORIOS DE CÓDIGO28

GITLAB	28
Proyectos de GitLab.....	28
Proyectos de usuario o grupo.....	30
Opciones de creación de proyectos	30
Grupos	30
Usuarios	31
Roles de trabajo	31

INTRODUCCIÓN

Las nuevas implementaciones de trabajos web deben cumplir requerimientos que permitan mejorar la calidad de software. Esto también sienta las bases para actualizar componentes WEB antiguos a un nuevo modelo de trabajo, las consideraciones principales son:

- Seguridad WEB
- Codificación
- Guías de codificación
- Herramientas para manejo de paquetes
- Repositorios de código

Nota importante: Al momento de la publicación todos los recursos que se toman del repositorio de Gobierno requieren acceso explícito al repositorio; esta disposición está sujeta a cambiar previa revisión.



SEGURIDAD WEB

El servidor en el que se alojan los sitios web de las dependencias es compartido, y cualquier vulnerabilidad de seguridad en un sitio podría comprometer la seguridad en los demás sitios alojados en el mismo servidor, por lo que es necesario cumplir con las siguientes recomendaciones en cualquier sitio publicado bajo el dominio **yucatan.gob.mx**:

1. El código debe tener todas las validaciones necesarias para evitar la inyección SQL.
2. Limpiar todos los datos provenientes de formularios o de la barra de direcciones: eliminar palabras reservadas como `create, insert, update, delete, drop, select, union, alter, grant, truncate, between, or, jquery, javascript, script`; eliminar caracteres especiales como `< > % $ () " ' o` en su defecto, convertirlos a sus equivalentes códigos HTML.
3. Validar el código HTML ingresado por el usuario por medio de algún editor de texto enriquecido (TinyMCE, CKEditor, etc.), en especial los estilos en línea, por ejemplo, `<DIV STYLE="background-image: url(javascript:alert('XSS'))">`.
4. Validar que las variables acepten únicamente los tipos de datos necesarios (por ejemplo, una variable numérica no debe aceptar letras).
5. Si el sitio acepta archivos como entrada, limitar el tipo de archivos que se pueden enviar, y no aceptar archivos ejecutables o posiblemente maliciosos.
6. No mostrar mensajes de error que contengan información de la estructura de la base de datos, la instrucción SQL que generó el error, el tipo de servidor que se utiliza, etc.
7. Utilizar CAPTCHAS en todos los formularios.
8. En caso tener módulos que requieran autenticación, las contraseñas aceptadas deben ser robustas y libres de ataques por fuerza bruta:
 - o Deben almacenarse encriptadas en la base de datos (de preferencia, con el algoritmo SHA-1).
 - o Tener una longitud mayor a 8 caracteres.
 - o Utilizar una combinación de letras mayúsculas y minúsculas, números y caracteres especiales.

CODIFICACIÓN

CODIFICACIÓN FÍSICA DE LOS ARCHIVOS FUENTE

UTF-8 (SIN BOM) (IMPORTANTE)

Este formato de codificación de caracteres está catalogado como una regla y se espera que sea aplicado a todos los archivos de código fuente que conforman la página o aplicación, de igual forma se recomienda que en caso de utilizar bases de datos que permitan implementar codificación, se utilice UTF-8. Ejemplo: `mysql(utf8_default_collation)`.

Importancia de remover el BOM

Aunque hoy en día los principales IDEs para desarrollo web consideran su eliminación por defecto (VSCode, Meteor, Web/PhpStorm) existen algunos como el Notepad++ que dan la opción de incluir el BOM en el formato UTF-8, lo que presenta riesgo de inconsistencias y problemas de visualización en la salida de la información.

FIN DE LINEA (LF) (DESEABLE)

Se sugiere utilizar el fin de línea de UNIX en los códigos fuente, es decir, el uso del carácter LF a diferencia del CRLF, ya que no se requiere realmente el carácter adicional que incluye el fin de línea de estilo Windows CRLF

RECOMENDACIONES DE PROGRAMACIÓN

HTML Y CSS

En cuanto a HTML, la recomendación es dejar de utilizar estándares antiguos tales como HTML4 o XHTML, el estándar definido y recomendado es el HTML5.

NOMBRES DE ARCHIVOS HTML

La recomendación en cuanto a los estándares de nomenclatura para los nombres de archivos HTML, es el uso de snake case, es decir, separar cada una de las palabras por un guion bajo (_), ejemplos:

- **index.html**
- **formulario_contacto.html**
- **not_found.html**

Cómo indicar al navegador que utilizas HTML5 (DOCTYPE)

XHTML (Forma incorrecta de codificar)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

HTML5 (Forma correcta)

```
<!DOCTYPE html>
```

Parece algo sin importancia, pero con esta instrucción HTML se le indica al navegador qué características podría o no utilizar, hoy en día la mayoría de los navegadores son extremadamente sensibles a esto.

RESPONSIVIDAD

Actualmente, el 52% de la navegación web mundial se hace a través de dispositivos móviles, razón por la cual los diseños responsivos son una **regla** interna para los sitios y web apps.

Primeros pasos de un sitio responsivo

incluir el **meta tag** correspondiente para manejo de escalas de pantalla (**viewport**):

```
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-
scale=1">
```

Nota: Este ejemplo incluye además el meta para forzar al navegador a interpretar el contenido como UTF-8

LIBRERÍAS CSS PARA DISEÑO WEB RESPONSIVO

Actualmente existen diversas librerías CSS que permiten utilizar de manera sencilla el diseño web responsivo. Aquí algunas recomendaciones:

- [Bootstrap](#) - Recomendado
- [Bulma](#) - Recomendado
- [Foundation](#)
- [Skeleton](#)

Nota: En pro de la innovación y mejora dejamos elecciones abiertas para el manejo de hojas de estilos, sin embargo, es importante recalcar que se debe elegir un Framework CSS que permita cumplir con las exigencias de estilos visuales requeridas por la entidad a cargo, así que en su elección considere uno que pueda escalarse y adaptarse de manera correcta.



RECOMENDACIONES GENERALES DE HTML5

1. Ya no es necesario indicar el parámetro **type** en etiquetas **link** o **script**.

- Incorrecto

```
<link rel="stylesheet" type="text/css" href="style.css" />
<script type="text/javascript" src="script.js"></script>
```

- Correcto

```
<link rel="stylesheet" href="style.css">
<script src="script.js"></script>
```

2. No necesitas indicar la diagonal **'/'** para cierre de tags simples.

- Incorrecto

```
<br />
<input type="text" id="text" />
```

- Correcto

```
<br>
<input type="text" id="text">
```

3. Se recomienda aprovechar las nuevas funciones de input y evitar la carga de scripts para agregar funcionalidades que ahora vienen por default en los navegadores, como campos de número o email.

```
<input type="date" name="fecha" id="fecha">
<input type="time" name="hora" id="hora">
<input type="datetime-local" name="fecha-hora" id="fecha-hora">
<input type="number" name="numero" id="numero" min="1" max="10">
<input type="url" name="url" id="url">
<input type="range" name="rango" id="rango">
<input type="email" name="correo" id="correo">
```

4. Como regla tenemos el no utilizar estilos con tags depreciados que han sido suplantados por hojas de estilo css. Los más comunes que hemos encontrado son:

```
<bold>
<font>
<center>
```

5. No utilizar layouts basados en *frames* o *tables*; éstos están depreciados desde HTML4 y entran en conflicto directo con la regla de responsividad.

Esto no quiere decir que no podamos presentar información en formato de tablas o utilizar iframes para mostrar contenido externo, sin embargo, recomendamos hacer uso de ello en la menor medida posible.

Para una lista más completa y verificación de compatibilidad entre navegadores, sugerimos usar el recurso online: [Can I Use](#)

RECOMENDACIONES GENERALES DE HOJAS DE ESTILO CSS

NOMBRES DE ARCHIVOS CSS

La recomendación de estándar para la nomenclatura de los nombres de archivo CSS es el uso de kebab, es decir, separar cada una de las palabras por un guion medio (-), ejemplos:

- **style.css**
- **image-slider.css**
- **shadow-image-box.css**

CODIFICACIÓN

Para los nombres de las clases CSS la regla es utilizar la notación **BEM** (Block Element Modifier), ya que lo hace más entendible para su uso en la semántica html, ejemplo:

- Incorrecto

```
.galContainer{  
    padding: 12px;  
}
```

- Correcto

```
.gal-container{  
    padding: 12px;  
}
```

A manera de recomendación y sugerencia tenemos el adoptar la metodología BEM de forma integral en sus estilos css, utilizando clases y modificadores según la función que les corresponde y evitando anidamientos, ejemplo:

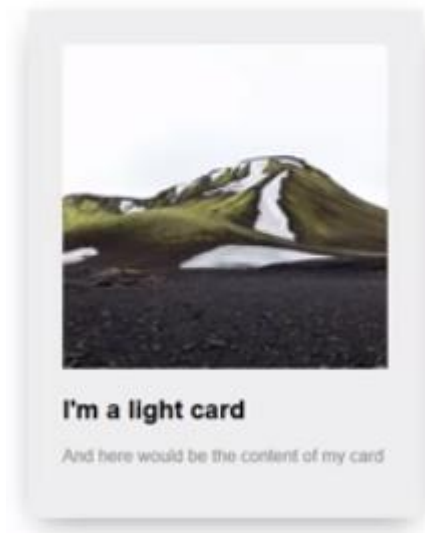
Una forma común (incorrecta en BEM) de realizar un estilo para un card sería este:

```
<!-- EL HTML -->  
<div clas="card">  
      
    <h2>I'm a light card</h2>  
    <p>Lorem ipsum</p>
```



```
</div>
/*EL CSS*/
.card {
  width: 300px;
  height: 400px;
  padding: .5em;
  border: 1px solid #000;
  margin: 0 1em;
  color: #0f0f0f;
}
.card p {
  font-weight: bold;
}
.card h2 {
  font-weight: bold;
  font-size: 1.5em;
}
.card img {
  margin: 1em;
  width: 100%;
  height: auto;
}
```

- Resultado





Si bien el resultado es el esperado, la forma tradicional nos está limitando la capacidad de hacer modificaciones al card principal. BEM nos sugiere el uso de modificadores y clases específicas, ejemplo:

```
<!--EL HTML-->
<div clas="card">
  
  <h2 class="card__title">Titulo</h2>
  <p class="card__text">Lorem ipsum</p>
</div>
/*EL CSS*/
.card {
  width: 300px;
  height: 400px;
  padding: .5em;
  border: 1px solid #000;
  margin: 0 1em;
  color: #0f0f0f;
}
.card__text {
  font-weight: bold;
}
.card__title {
  font-weight: bold;
  font-size: 1.5em;
}
.card__image {
  margin: 1em;
  width: 100%;
  height: auto;
}
```

El resultado es el mismo, pero nuestra capacidad de modificar se expande con menos código. Digamos que queremos cards de diversos colores:

```
<!-- Agregamos un modificador de color 'card--dark'-->
<div clas="card card--dark">
  
  ...
/*-Definimos el modificador en el codigo CSS-*/
.card--dark {
  background: #0f0f0f;
  color: #ddd;
}
```



```
}
```

Al final nuestros estilos serán más moldeables y entendibles ya que desde el HTML podremos observar las clases independientes de cada elemento y no tendremos que preocuparnos si en el CSS hay algún tipo de animamiento que no estemos contemplando.

- Resultado



MODOS DE USO CSS

- Hoja de estilos (**recomendado**) - Deben ir dentro del `<head>`

```
<link rel="stylesheet" href="ruta/style.css">
```
- Etiqueta Style - Deben ir preferentemente en el `<head>` y por debajo de las hojas de estilo, solo se recomienda en páginas independientes.

```
<style> .gal-container{ padding: 12px; } </style>
```
- CSS Inline - Solo en casos que requieran su uso explícito

```
<div style="width: 100%"></div>
```

LENGUAJES DE PROGRAMACION WEB

Las siguientes recomendaciones aplican para los lenguajes web comúnmente utilizados (Javascript, PHP, C# .Net, Ruby on Rails, etc.)

NOMBRES DE ARCHIVOS DE CÓDIGO

Se recomienda para los lenguajes de programación web el uso de Camel Case (la primera letra de cada una de las palabras es mayúscula, con excepción de la primera), permitiendo el uso de mayúscula inicial para Clases o Componentes, ejemplo:

- **main.js**
- **apiGobierno.js**
- **routes.php**
- **GobiernoController.php** (clase)
- **TopNavBar.js** (componente)

CODIFICACIÓN

Al igual que en los nombres de archivos, la codificación recomendada es mediante el uso de Camel Case, minúscula para funciones y variables, mayúscula para clases, exceptuando constantes que se recomiendan en snake case y mayúsculas:

```
/** variables y funciones */  
function fooBar() {  
    var logVar = 123;  
    console.log(logVar);  
}  
  
/** Clases */  
class GobiernoController() {  
    ...  
}  
  
/** constantes */  
  
const API_GOBIERNO_KEY = '2342i34oi2h219834n92n498213j4=='
```

Para mayor información sobre recomendaciones web puedes utilizar nuestro recurso de documentación más completo en el siguiente repositorio: [Documentación - Recomendaciones de programación Web](#)



ESTRUCTURA DE CARPETAS Y CÓDIGOS WEB

La manera de cómo se estructura y se maneja el código fuente de un proyecto es responsabilidad de cada desarrollador, sin embargo, aquí se provee una estructura sugerida, así como una template de proyecto base y documentación adicional requerida. Los ejemplos son considerados para código con un backend PHP y un frontend abierto, pero pueden adaptarse a otro tipo de lenguajes.

Estructura sugerida

- App - Carpeta para guardar y estructurar el código backend de nuestra aplicación, estructurando de manera correcta las clases y funciones de ayuda. Un ejemplo de estructura interna:
 - Clases
 - Templates
 - Catalogos
- Config - Carpeta para incluir todos los archivos de configuración requeridos (variables de entorno o configuraciones a BD).
- Files-Content - Carpeta para manejo de archivos estáticos, manejando una estructura interna que convenga para mantener el orden. Ejemplo de estructura interna:
 - Uploads
 - Cache
 - Documentos
- vendor - En caso específico de PHP se utiliza para almacenar librerías de terceros.
- Public o App - Carpeta con archivos de la parte pública (frontend) de la aplicación. La estructura recomendada para el frontend es la siguiente:
 - css - Carpeta para estilos
 - img - Imágenes estáticas del proyecto
 - js - Carpeta para scripts propios
 - fonts - Carpeta para fuentes utilizadas en frontend
 - favicon - puede ser un archivo **ico** o un folder con estructura de favicons para dispositivos diversos.



- lib - Carpeta para librerías de terceros

Nota: El caso de nombrar la carpeta de Public como App o semejante es recomendado únicamente para aplicaciones compiladas, como por ejemplo Apps de Angular, Vue, React, etc., ya que es un mejor indicativo del contenido y de que la estructura interna del código puede variar de acuerdo al tipo de App.

Actualmente tenemos un Boilerplate PHP listo para uso y con métodos básicos de trabajo para desarrollo de aplicaciones WEB. Puedes usar las siguientes ligas de repositorio:

- [DAI PHP Backend - Proyecto de App base para desarrollo](#)
- [Presentación de slides del proyecto](#)
- [Documento de los slides del proyecto](#)

HERRAMIENTAS PARA MANEJO DE PAQUETES

Algunos lenguajes como Ruby on Rails cuentan con su propio manejador de paquetes (GEM). Para PHP o Javascript también existen y es una recomendación utilizarlos por las siguientes ventajas que ofrecen:

- Permiten controlar la versión de la librería que deseas incluir en tu proyecto
- Permiten actualizar las librerías del proyecto y realizar cambios de versión sin necesidad de cambiar código
- Permiten reducir el tamaño del código fuente ya que no es necesario guardar las librerías dentro del proyecto, sino que pueden instalarse cuando se requiera.
- Permiten trabajar en diversos ambientes de ejecución sin conflictos.

COMPOSER PARA PHP

Es un manejador de paquetes para PHP que proporciona un estándar para administrar, descargar e instalar dependencias y librerías. Prácticamente, cualquier librería de terceros libre que encontremos en páginas como GitHub, están disponibles en su lista de paquetes y pueden ser manejadas en composer; muchas incluso, solo recomiendan instalarse mediante este manejador y no intentarlo de manera manual.

- [Sitio principal y de descarga de composer](#)
- [Packagist - Lista general de repositorio de paquetes PHP](#)

CÓMO UTILIZARLO

Una vez instalado, podemos configurarlo en nuestro proyecto incluyendo un archivo composer.json en la raíz de nuestro proyecto php.

El Boilerplate sugerido en apartados anteriores maneja sus librerías por este método.

Este archivo contiene algo de información sobre nuestro proyecto, versión de PHP requerida, listado de paquetes necesarios, entre otros.

Si no se especifica la versión de PHP, ésta dependerá de nuestra versión de PHP de sistema, es decir, no puedes indicar que tu proyecto utiliza una versión PHP 5.6 si tu instalación es una versión menor, sin embargo, no habría problemas si tu versión fuese mayor.

La versión de PHP es importante en el sentido que composer instalará las versiones de librerías específicas que sean compatibles con la versión de PHP que indiquemos o esté instalada.

Ejemplo de composer.json:



```
{
  "name": "DAI Backend",
  "type": "Web Backend",
  "description": "Boilerplate para inclusión del backend DAI en
tus proyectos",
  "license": "MIT",
  "support": {
    "docs": "https://gitlab.yucatan.gob.mx/DAI/dai-
backend/master/readme.md",
    "issues": "https://gitlab.yucatan.gob.mx/DAI/dai-
backend/issues",
  }
  "config": {
    "platform": {
      "php": "5.6.39"
    }
  },
  "require": {
    "illuminate/database": "*",
    "slim/slim": "3.*",
    "phpmailer/phpmailer": "^5.2",
    "cboden/ratchet": "0.3.*",
    "doctrine/dbal": "*",
    "slim/twig-view": "^2.4"
  },
  "require-dev": {
  }
}
```

El apartado **require** es el más importante. Aquí podemos incluir la lista los paquetes que necesitamos instalar. Éstos también se insertan de manera automática en el listado si se instalan en el proyecto por CLI de composer.

El apartado **require-dev** es similar, pero solo sirve para librerías que únicamente utilizamos durante el desarrollo pero no son necesarias para el trabajo en producción.

CLI DE COMPOSER

La documentación completa del manejo del CLI puede encontrarse en los docs oficiales de composer; [Composer CLI DOCS](#), en este apartado únicamente sugerimos el uso básico que comprende:

- Instalación de paquetes
- Desinstalación de paquetes
- Instalación y actualización de paquetes listados en composer.json

Nota: Para Windows, asegúrate de haber hecho que la ruta a composer esté incluida en el PATH de sistema, de lo contrario no podrás ejecutar los comandos.

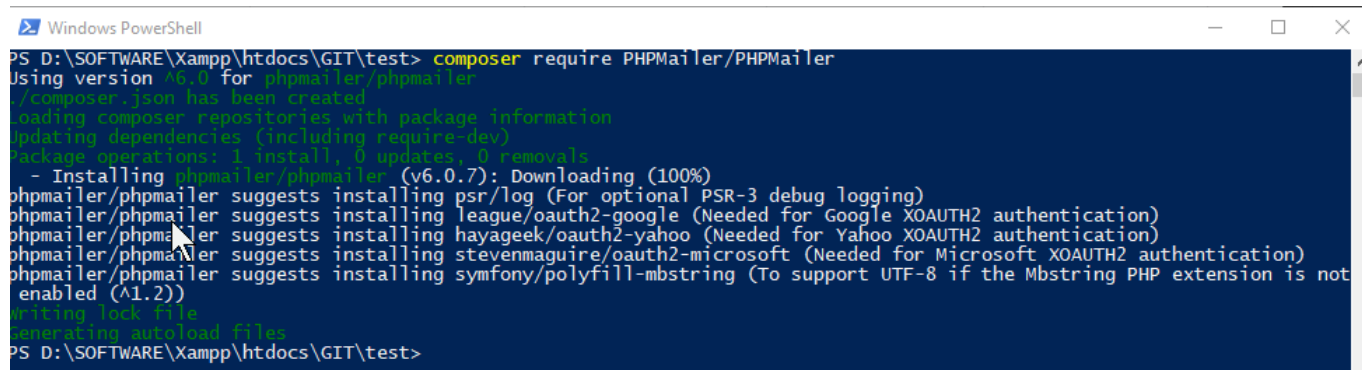
INSTALACIÓN DE PAQUETES

La instalación inicial de paquetes únicamente requiere que tengas el nombre del paquete listo para uso. A la fecha, la mayoría de las librerías de terceros incluyen el dato en su documentación inicial; de no hacerlo, la convención estándar es utilizar su nombre en Github. Por ejemplo, para PHP Mailer:

URL de GitHub `https://github.com/PHPMailer/PHPMailer`

Por lo tanto, el nombre del paquete es `PHPMailer/PHPMailer`

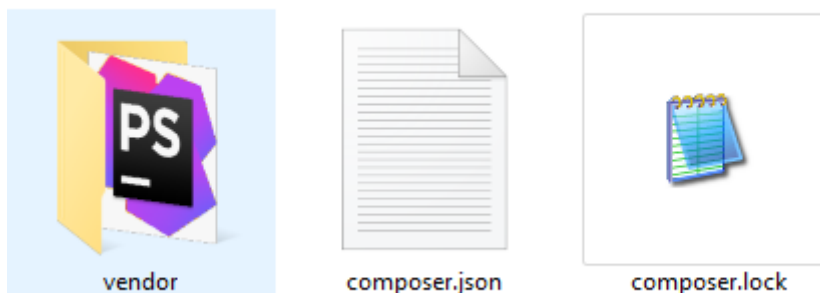
Comando de instalación composer `require PHPMailer/PHPMailer`, ejemplo:



```
PS D:\SOFTWARE\Xampp\htdocs\GIT\test> composer require PHPMailer/PHPMailer
Using version ^6.0 for phpmailer/phpmailer
./composer.json has been created
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
- Installing phpmailer/phpmailer (v6.0.7): Downloading (100%)
phpmailer/phpmailer suggests installing psr/log (For optional PSR-3 debug logging)
phpmailer/phpmailer suggests installing league/oauth2-google (Needed for Google XOAUTH2 authentication)
phpmailer/phpmailer suggests installing hayageek/oauth2-yahoo (Needed for Yahoo XOAUTH2 authentication)
phpmailer/phpmailer suggests installing stevenmaguire/oauth2-microsoft (Needed for Microsoft XOAUTH2 authentication)
phpmailer/phpmailer suggests installing symfony/polyfill-mbstring (To support UTF-8 if the Mbstring PHP extension is not enabled (^1.2))
Writing lock file
Generating autoload files
PS D:\SOFTWARE\Xampp\htdocs\GIT\test>
```

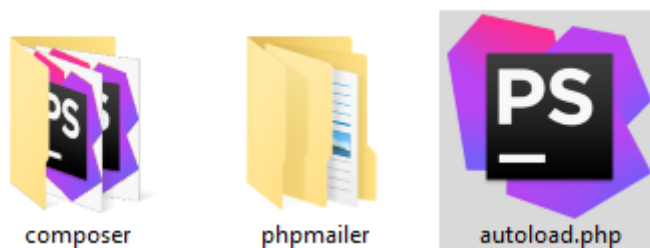
Una vez completa la instalación, se habrá generado automáticamente en la raíz una carpeta **vendor**, y en caso de no existir, los archivos **composer.json** y **composer.lock** con la librería requerida y sus dependencias.

SOFTWARE > Xampp > htdocs > GIT > test



Este proceso se puede repetir para todas las librerías de terceros que nuestro proyecto requiera, y al final otra ventaja de composer es que nos da un punto único de inclusión. El archivo **vendor/autoload.php** es el único archivo al que tenemos que hacer referencia en el código (include o require) y éste se encarga de cargar las librerías requeridas bajo demanda automática.

> SOFTWARE > Xampp > htdocs > GIT > test > vendor



DESINSTALACIÓN DE PAQUETES

Este proceso es requerido cuando deseamos remover una dependencia del proyecto. No podemos simplemente borrar la carpeta, ya que la referencia al paquete permanecería en **composer.json** y en **autoload.php**. El proceso es igual de sencillo:

Únicamente ejecutamos la instrucción `remove composer remove PHPMailer/PHPMailer`

```
Windows PowerShell
PS D:\SOFTWARE\Xampp\htdocs\GIT\test> composer remove PHPMailer/PHPMailer
phpmailer/phpmailer is not required in your composer.json and has not been removed
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 0 installs, 0 updates, 1 removal
- Removing phpmailer/phpmailer (v6.0.7)
Generating autoload files
PS D:\SOFTWARE\Xampp\htdocs\GIT\test>
```

INSTALACIÓN Y ACTUALIZACIÓN DE PAQUETES INCLUIDOS EN COMPOSER.JSON

El proceso de instalación se puede dar si instalamos paquetes que incluimos directamente en composer.json sin utilizar el CLI o en un nuevo equipo de desarrollo donde no tuvimos que copiar todo el contenido de **vendor**.

Únicamente corremos la instrucción `composer install` desde la raíz del proyecto.

```
Windows PowerShell
PS D:\SOFTWARE\Xampp\htdocs\GIT\test> composer install
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
- Installing phpmailer/phpmailer (v6.0.7): Loading from cache
phpmailer/phpmailer suggests installing psr/log (For optional PSR-3 debug logging)
phpmailer/phpmailer suggests installing league/oauth2-google (Needed for Google XOAUTH2 authentication)
phpmailer/phpmailer suggests installing hayageek/oauth2-yahoo (Needed for Yahoo XOAUTH2 authentication)
phpmailer/phpmailer suggests installing stevenmaguire/oauth2-microsoft (Needed for Microsoft XOAUTH2 authentication)
phpmailer/phpmailer suggests installing symfony/polyfill-mbstring (To support UTF-8 if the Mbstring PHP extension is not enabled (^1.2))
Writing lock file
Generating autoload files
PS D:\SOFTWARE\Xampp\htdocs\GIT\test>
```

El proceso de actualización es similar y nos puede servir para actualizar un paquete en particular o todos los paquetes instalados por composer al proyecto.

Para un paquete, ejecutamos `composer update PHPMailer\PHPMailer`. Para actualizar el proyecto completo, `composer update`

```
PS D:\SOFTWARE\Xampp\htdocs\GIT\test> composer update
Loading composer repositories with package information
Updating dependencies (including require-dev)
Nothing to install or update
Generating autoload files
PS D:\SOFTWARE\Xampp\htdocs\GIT\test>
```

BOWER PARA JAVASCRIPT ES5

Este manejador está diseñado para librerías javascript que se deseen manejar en la modalidad antigua (ES5) de inclusión de scripts a sitio o app, es decir incluyendo los script tags dentro del código fuente:

```
<script src="lib/jquery/jquery.js"></script>
```

Nota: Aunque perfectamente funcional, veremos más adelante que la metodología de trabajo ES6 nos ofrece ciertas ventajas por sobre ES5.



REQUERIMIENTOS PREVIOS

- [Node & NPM](#)
- [GIT](#)

INSTALACIÓN

Una vez instalados los requerimientos **bower** se instala directamente sobre el CLI de NPM, con el comando:

```
npm install -g bower
```

Una vez finalizada la instalación podremos utilizar bower mediante su CLI; esto se realiza dentro de la carpeta Public o App del proyecto ya que en este caso las dependencias son instaladas para el frontend.

Nota: Por defecto bower instala las librerías en una carpeta **bower_components**. Para cambiar esto debemos crear previamente en la raíz de public un archivo nombrado **.bowerrc** con el nombre del directorio donde queremos almacenar nuestras dependencias; en nuestro caso utilizamos por estándar la carpeta **lib**.

```
{  
  "directory": "lib"  
}
```

INSTALACIÓN DE PAQUETES

Todo el proceso se realiza desde el cli de bower con 2 consideraciones:

- El paquete/librería se encuentra dado de alta en el listado de bower:

```
bower install jquery --save
```

Ya que jquery está listado en los paquetes de bower solo necesitamos su nombre; otros ejemplos serían: fontawesome, bootstrap, angular.

- El paquete/librería no se encuentra listado, pero hay un github o liga al script:

```
bower install git://github.com/twbs/bootstrap.git --save
```

o bien:

```
bower install http://example.com/script.js --save
```

Si bien bootstrap es soportado por nombre eso no impide instalarlo desde su liga de github como se ve en el ejemplo.



Si requerimos una versión específica podemos indicarla en el caso de un paquete Github o del listado de bower:

```
bower install jquery#1.2.2 --save
```

Esto instalaría jquery específicamente en la versión 1.2.2.

Nota: Podemos cambiar el sub parámetro `--save` por `save-dev` para dependencias que solo utilizaremos en desarrollo.

Las instalaciones generan o actualizan un archivo **bower.json** que se crea en la raíz de ejecución, cuya función es la misma que la de **composer.json**, dar seguimiento a los paquetes instalados y dar la facilidad de reinstalarlos o actualizarlos a versiones específicas. Ejemplo de bower.json:

```
{
  "name": "Frontend de prueba",
  "authors": [
    "Gobierno del Estado de Yucatán"
  ],
  "license": "MIT",
  "homepage": "www.yucatan.gob.mx",
  "private": true,
  "ignore": [
    "**/*.*",
    "node_modules",
    "bower_components",
    "test",
    "tests",
    "lib"
  ],
  "dependencies": {
    "jquery": "1.2.23",
    "bootstrap": "~3.2.0",
    "animate.css": "~3.2.0",
  }
}
```

Recuerda que a diferencia de composer los paquetes de bower no ofrecen un punto único de integración por ser utilizados bajo metodología ES5, por lo que debes ingresar los scripts correspondientes en los lugares necesarios, para jquery, por ejemplo:

```
<script src="lib/jquery/dist/jquery.min.js"></script>
```



DESINSTALAR PAQUETES

Las consideraciones son las mismas que en install, pero debemos utilizar el comando uninstall:

```
bower uninstall jquery
```

ACTUALIZAR PAQUETES

De la misma forma que composer funciona en modalidad general y específica:

```
bower update jquery --save  
bower update
```

Nota: El sub parámetro `--save` es necesario para actualizar el listado en bower.json; de no utilizarlo el paquete no quedará registrado como dependencia de la aplicación, o en su defecto la versión actualizada.

NPM PARA FRONTENDS ES6 O BACKEND NODEJS

No entraremos en detalle amplio de manejo de Apps en formato ES6 ya que es un tema extenso, sin embargo, su uso ofrece diversas ventajas sobre aplicaciones web tradicionales:

- Mejor manejo de paquetes
- Mejor manejabilidad del código
- Uso de librerías o código Javascript avanzado no disponible en navegadores web, pero compatible en compilación
- Uso de variantes de ECMAScript adicionales a javascript (TypeScript o CoffeeScript)
- Modificación y compilación de código automático en distribución
- Eliminación de elementos innecesarios en distribución (imágenes no utilizadas, READMEs, partes de librerías que no se utilizaron, etc.).
- Trabajar frontend y backend con un mismo lenguaje (NodeJS)

Si estás interesado en mejorar tus habilidades y programar frontend con ES6 aquí algunos recursos.

- [How to Build a WebPack App with vanilla Js](#) - Utilizando React
- [Laracast Vue Step by Step](#) - Utilizando framework vue
- [Angular Deployment](#) - Utilizando angular

Basado en estos recursos puedes construir cualquier app en ES6 y aprender de algunos frameworks disponibles.

INSTALACIÓN

Si trabajas un proyecto ES6 te darás cuenta que el manejador NPM trabaja directamente relacionado al desarrollo durante toda la vida del proyecto; únicamente necesitas descargar una distribución de NodeJs para comenzar:

- [NodeJs Sitio Oficial](#)

Durante el proceso, el node en Windows te solicitará permiso para instalar el npm e incluirlo en el PATH de sistema. Se deben aceptar estas opciones para poder continuar.

INSTALACIÓN DE PAQUETES

En una estructura de proyecto node todos los paquetes se almacenan en una carpeta **node_modules**; al igual que bower o composer, las dependencias se guardan en un archivo **package.json** y **package.lock** donde están divididas en:

- dependencies - Para dependencias del proyecto
- devDependencies - Para dependencias de desarrollo únicamente

Entre otros datos de información del proyecto, las instalaciones se realizan mediante el comando:

```
npm install --save jquery
```

ó

```
npm install --save-dev lodash
```

Las instalaciones a versiones específicas de paquetes, requieren la inclusión de una @ y el número de versión:

```
npm install --save lodash@4.17.4
```

Al igual que en bower el parámetro --save indica que el paquete será incluido en el listado de dependencias del **package.json**.

Por último, podemos ejecutar el comando de instalación sin un paquete específico; esto instalará todas las dependencias listadas en package.json en nuestro proyecto:

```
npm install
```

Ejemplo de package.json

```
{
  "name": "gob.mx.yucatan.vue.boilerplate",
  "description": "Boilerplate inicial para aplicaciones Vue",
  "version": "1.0.0",
  "author": "Gobierno del Estado de Yucatan",
  "license": "MIT",
  "private": true,
  "scripts": {
    "dev": "cross-env NODE_ENV=development webpack-dev-server --open --inline --hot",
    "webpack": "cross-env NODE_ENV=development webpack --watch",
    "build": "cross-env NODE_ENV=production webpack --progress --hide-modules"
  },
  "dependencies": {
    "axios": "^0.18.0",
    "popper.js": "^1.14.3",
    "vue": "^2.5.11",
  },
  "browserslist": [
    "> 1%",
    "last 2 versions",
    "not ie <= 8"
  ],
}
```



```
"devDependencies": {
  "babel-core": "^6.26.0",
  "babel-loader": "^7.1.2",
  "babel-preset-env": "^1.6.0",
  "babel-preset-stage-3": "^6.24.1",
  "webpack": "^3.6.0",
  "webpack-cli": "^2.0.14",
  "webpack-dev-server": "^2.9.1"
},
"main": "webpack.config.js",
"keywords": []
}
```

DESINSTALAR PAQUETES

La desinstalación es muy similar a bower; únicamente indicamos el paquete:

```
npm uninstall --save lodash
```

En este caso la instrucción save es requerida para indicar que el paquete también será removido del package.json.

ACTUALIZAR PAQUETES

La actualización de paquetes funciona de la misma forma; únicamente indicamos el paquete a actualizar o si no se indica, se realizará una actualización global de paquetes.

```
npm update--save-dev lodash
```

ó

```
npm update
```

CAPACIDADES DE ES6

Los paquetes instalados y trabajados en proyectos ES6 permiten importarse entre las diversas librerías o clases del proyecto de la siguiente forma:

```
import _ from 'lodash'
import $ from 'jquery'
function foo(num) {
  let result = _.sum(num + 5);
  $('#div').html(result);
}
```



No dependemos de que se importe un script tag en el html porque el proyecto base está corriendo sobre una versión avanzada de javascript como su motor principal, y que generalmente compilaremos en una distribución que un navegador pueda interpretar.

Una ventaja más de este modelo de trabajo es que podemos trabajar apps es6 que sean corridas directamente por un servidor NodeJs, lo que nos permitiría en un momento dado eliminar la dualidad de lenguajes y trabajar una aplicación o sitio web únicamente con Javascript ES6, tanto en el frontend, como en el backend (sin necesidad de php, rails, python, etc.).

Para más información sobre cómo publicar servicios basados en NodeJs puedes verificar los siguientes recursos:

- [Simple Web Server with NodeJs](#)
- [NodeJs Express App Tutorial](#)

Actualmente contamos en el repositorio con el Boilerplate para una app ES6 basada en Vue. Puedes verificar el recurso en la siguiente liga:

- [Vue Boilerplate](#)

REPOSITORIOS DE CÓDIGO

Un repositorio de código es un lugar donde el código de una aplicación o de un programa cualquiera está almacenado y desde donde se puede distribuir. Hoy en día debe ser una práctica mandatoria, dada la creciente necesidad de tener un control modular, recuperación, manejo de cambios y trabajo en equipo.

GITLAB

Actualmente utilizamos la herramienta GitLab ya que permite el uso de un repositorio con funciones de trabajo colaborativo y seguimiento de avance de proyectos (mejoras, desarrollos y bugs) y puede ser instalado y administrado de manera privada en nuestros servidores. La liga actual es:

[Servidor GitLAB de Gobierno del Estado](#)

De igual forma, tenemos un flujo completo de trabajo propuesto que se basa en el uso y explotación de la herramienta. La documentación completa la puedes encontrar en:

- [Presentación en slides](#)
- [Documento de la presentación](#)

En la documentación se explica a detalle tanto el flujo de trabajo propuesto como las ventajas y funcionalidades que se pueden aprovechar en las etapas de desarrollo con el GitLab. En este documento se tocarán únicamente los puntos básicos.

- Proyectos
- Grupos
- Usuarios
- Roles de trabajo

PROYECTOS DE GITLAB

Un proyecto es básicamente un espacio en el repositorio destinado a mantener y controlar el flujo de desarrollo de un sistema/página/plataforma o app en particular.

Un proyecto puede ser:

- Público
- Privado
- Interno



A su vez, un proyecto puede pertenecer a:

- Grupos
- Usuarios

Blank project	Create from template	Import project
<p>Project name</p> <input type="text" value="Nuevo proyecto"/>		
<p>Project URL</p> <input type="text" value="http://gitlab.yucatan.g"/>		
<p>Project slug</p> <input type="text" value="nuevo-proyecto"/>		
<p>Want to house several dependent projects under the same namespace? Create a group.</p>		
<p>Project description (optional)</p> <div>Description format</div>		
<p>Visibility Level ?</p> <p><input checked="" type="radio"/> Privado El acceso al proyecto debe concederse explícitamente a cada usuario.</p> <p><input type="radio"/> Interno El proyecto puede ser accedido por cualquier usuario conectado.</p> <p><input type="radio"/> Público El proyecto puede accederse sin ninguna autenticación.</p>		
<p><input type="checkbox"/> Initialize repository with a README Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.</p>		
<input type="button" value="Create project"/>		<input type="button" value="Cancel"/>



Diferencias

- Un proyecto **privado** requiere acceso explícito a cada usuario para que pueda utilizarlo; esta restricción no aplica si el proyecto pertenece a un **grupo** y ese grupo tiene otros **maintainers**.
- Un proyecto **interno** puede ser accedido por cualquier usuario conectado al Gitlab.
- Un proyecto **público** puede ser accedido por cualquier usuario que tenga acceso al portal de Gitlab, aun cuando no esté registrado; esto no quiere decir que pueda realizar cambios, este modo podría estar restringido si el proyecto se está creando dentro de un grupo privado.

PROYECTOS DE USUARIO O GRUPO

Estas dos vertientes tienen las siguientes diferencias:

- En un proyecto de usuario, el usuario individual tiene mayor control sobre el contenido; puede él mismo, dar accesos específicos a su proyecto tan público o privado como desee.
- Un proyecto de grupo está más restringido en el sentido de que no puede privatizarse para usuarios que pertenezcan al mismo grupo en cuanto a visibilidad.

OPCIONES DE CREACIÓN DE PROYECTOS

Un proyecto puede crearse en gitlab mediante 3 vertientes:

- **Proyecto en blanco.** Como su nombre lo indica, crea un repositorio vacío inicialmente.
- **Proyecto desde template.** Estos proyectos están basados en templates públicos de Gitlab o templates privados que la administración de Gitlab de Gobierno ponga a disposición como un template.
- **Importar proyecto.** Esta última opción nos permite importar un proyecto existente a nuestro repositorio privado; la importación puede ser desde otro Gitlab, github, bitbucket o en general cualquier otro repositorio git al que tengamos acceso.

GRUPOS

Los grupos son una forma de organizar a los proyectos y los desarrolladores. Puedes crear grupos en Gitlab para separar los desarrollos de las diversas áreas, y los desarrolladores pueden tener diferentes atribuciones en los proyectos de grupos diferentes.



USUARIOS

Los usuarios se dan de alta por la administración y pueden tener diversas atribuciones en los grupos y proyectos. Se manejan 2 roles principales:

- Developers
- Maintainers

La diferencia principal es que los **Maintainers** tienen algunas atribuciones especiales, pueden proteger ramas de proyectos (como **master**) y dar o negar permisos a los **Developers** en los proyectos que manejan.

ROLES DE TRABAJO

Mencionados anteriormente, tienen la funcionalidad de determinar el flujo del trabajo, Según el tamaño del proyecto, los roles de trabajo pueden tener un impacto diferente en los proyectos. Éstas son algunas de las facultades que se pueden manejar mediante los roles de trabajo:

- Maintainer puede dar acceso a proyectos a developers y programar **webhooks** para captura de eventos.
- Maintainer tiene acceso a **master** y **tags**.
- Developer requiere acceso explícito a proyectos no públicos.
- Developer puede crear ramas de trabajo basadas en **master** pero no modificarlo directamente.
- Developer puede hacer **Merge Request** para solicitar unir su rama a **master**.
- Maintainer puede aceptar, denegar o revertir estos merge request.



Juntos transformemos
Yucatán
GOBIERNO ESTATAL 2018 - 2024

SAF
SECRETARÍA DE
ADMINISTRACIÓN
Y FINANZAS



- Maintainer puede hacer liberaciones manuales o automatizadas de versiones de código.
- Ambos roles pueden asignar y cambiar las tareas levantadas en el sistema **issues**.