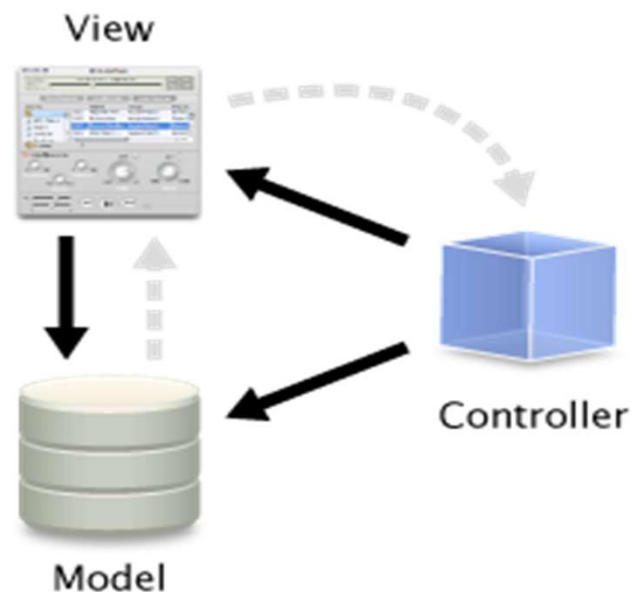


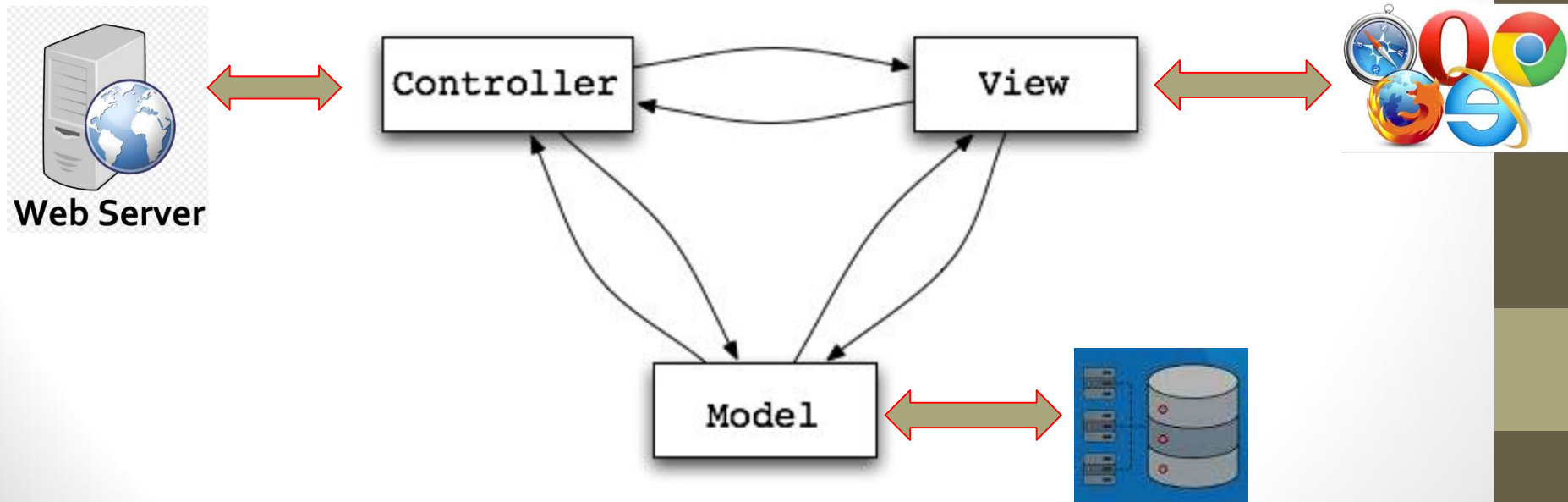
¿MVC? (Modelo-Vista-Controlador)

- MVC es un patrón arquitectural que divide una aplicación software en 3 grandes partes que trabajan juntas para formar la app.



Model View Controller

- Patrón arquitectural desarrollado por Centro de Investigación Xerox (Palo Alto) – Proyecto que uso Smalltalk (1979).
- Más utilizado en Apps Web en la actualidad.
- Desacopla Lógica de Negocios, Datos y Presentación.
- Facilita el desarrollo



El Modelo

- El modelo es la porción que implementa la “Lógica del Negocio”.
- Se le suele llamar el modelo porque representa objetos y sus interacciones del mundo real.
- Son representaciones activas de las tablas de la BD.
- Los modelos son los intermediarios que permiten conectar a la BD, consultarla y actualizarla si es necesario.

La vista

- Las vistas son las porciones de la aplicación MVC que presentan respuesta al usuario.
- La salida más común para aplicaciones web es el HTML. Podrían ser otras.
- Pueden ser descritas como archivos templates que presentan el contenido al usuario: variables, arrays y objetos son registrados por el controlador.
- Las vistas no deberían contener lógica del negocio compleja salvo estructuras de control para recorrer collecciones de datos (ej. Arrays) o validación de formularios.

El controlador

- El controlador es el corazón de la aplicación MVC. Este componente es el objeto que debería estar pendiente de las solicitudes HTTP hechas por el usuario.
- El controlador generalmente crea instancias de los modelos y utiliza métodos de esos modelos para conseguir los datos que se presentan a los usuarios, enviándolos a la vista correspondiente.
- Cada controlador puede ofrecer diferentes funcionalidades.

Flujo en MVC

- 1.El Usuario interactua con la vista (IU).
- 2.El Controlador maneja la entrada del Usuario (manejador o callback asociado a UI)
- 3.El Controlador actualiza el Modelo.
- 4.El Modelo interactua con la BD (de forma directa, con una capa de abstracción, Web service, etc)
- 5.El modelo retorna la información al Controlador.
- 6.El Controlador pasa la información a la vista
- 7.La Vista usa la información para generar una nueva UI.

¿Por qué MVC?

- Con MVC la aplicación se puede desarrollar rápidamente, de forma modular y mantenible.
- El diseño modular permite a los diseñadores y a los desarrolladores trabajar conjuntamente, así como realizar rápidamente el prototipado.
- Esta separación también permite hacer cambios en una parte de la aplicación sin que las demás se vean afectadas.

Definir un Modelo (User.js)

```
const { Sequelize, DataTypes, Model } = require('sequelize');  
const sequelize = new Sequelize(URIMysql);
```

```
class Usuario extends Model {
```

```
  Usuario.init({
```

```
    // Atributos del modelo son definidos aquí
```

```
    nombre: {
```

```
      type: DataTypes.STRING,
```

```
      allowNull: false // allowNull es true por defecto
```

```
    },
```

```
    apellido: DataTypes.STRING, //Solo si defino unicamente el tipo de datos
```

```
    fecha: { type: DataTypes.DATETIME, defaultValue: Sequelize.NOW }
```

```
  }, {
```


```
    // Otras opciones del modelo
```

```
    sequelize, // pasamos la instancia de la conexion
```

```
    modelName: 'Usuario' // El nombre del modelo
```

```
    //tableName: 'Usuarios'
```

```
  });
```



```
//si no queremos la pluralización  
{ define: { freezeTableName: true } }
```


Rutas con express

Express tiene un método Router() que crea un objeto router.

Este objeto puede verse como una mini-aplicación capaz de realizar funciones de middleware y rutas.

```
var express = require('express');  
var router = express.Router();  
// invocado por cualquier request pasado al router  
router.use(function (req, res, next) {  
  // alguna lógica..  
  next() })
```

Podemos usar un router para una vía de acceso particular y de esta forma separar las rutas en archivos o mini-aplicaciones.

```
// Solo request a /calendario serían enviados al "router"  
app.use('/calendar', router)
```

Rutas en archivos separados

```
//archivo routes/perro.js
var express = require('express');
var router = express.Router();
// middleware específico al router
router.use(function timeLog(req, res, next) {
  console.log('Time: ', Date.now());
  next(); });
// Define ruta al home de perro
router.get('/', function(req, res){
  res.send('Esta es mi home page de perro'); });
// define ruta al “acerca de”
router.get('/acercade', function(req, res) {
  res.send('Acerca de los perros'); });

module.exports = router;
```

```
//archivo app.js
var app = express();
var perrorouter = require('./routes/perro');
...
app.use('/perros', perrorouter);
```

Controladores

```
//archivo controllers/UserController.js
//Trabaja con los modelos y vistas si es necesario
var User = require('../models/user')
var user = new User();
```

```
exports.list = function(req, res, next){
  res.render('list', { users: User.find()});
};
```

```
exports.edit = function(req, res, next){
  res.render('edit', { user: req.user });
};
```

```
exports.update = function(req, res, next){
  var body = req.body;
  req.user.name = body.user.name;
  res.message('Information updated!');
  res.redirect('/user/' + req.params.id);};
```

```
//archivo routes/user.js
```

```
var express = require('express');
var UserController =
  require('../controllers/micontroller');
```

```
var api = express.Router();
```

```
// GET listar usuarios
```

```
api.get('/user',UserController.list);
```

```
// PUT actualizar usuarios
```

```
api.get('/user/:id',UserController.update)
```

CRUD - READ

```
// archivo controllers/PersonaController.js
```

```
var Persona = require('../models/Persona');
```

```
// Lista de todas las personas.
```

```
exports.listarPersonas = function(req, res) {  
  Persona.findAll({}, function (err, personas) {  
    if (err) return handleError(err);  
    res.json(personas);  
  });  
}
```

```
// Detalle de una persona
```

```
exports.detallePersona = function(req, res) {  
  res.send('No implementado: Detalle de Persona: ' +  
    req.params.id);  
};
```

```
//archivo routes/persona.js
```

```
...
```

```
router.get('/', PersonaController.listarPersonas)  
router.get('/:id', PersonaController.detallePersona);
```

CRUD - CREATE

```
// archivo controllers/PersonaController.js
// Crear Persona POST
var Persona = require('../models/Persona');
exports.crearPersona = function(req, res) {

    var objetoPersona = {
        nombre: req.body.nombre,
        fechaNacimiento: req.body.fechaNacimiento,
        documento: req.body.documento,
        edad: req.body.edad
    }

    var persona = Persona.build(objetoPersona);
    persona.save(function (err, result) {
        if (err) return handleError(err);
        res.send(result);
    });

    //archivo routes/persona.js
    ...
    router.post('/create', personaController.crearPersona)
```

CRUD - UPDATE

```
// archivo controllers/PersonaController.js
```

```
// Actualizar Persona PUT
```

```
exports.actualizarPersona = async function(req, res) {  
  var idp = req.params.id;  
  var personaUpdate = req.body;  
  const perUpd= await Persona.update(personaUpdate, {where:{id:idp}})  
  if(perUpd){  
    return res.status(200).send({persona: personaUpdated});  
  }  
  else{  
    return res.status(404).send({message: 'No existe la Persona'});  
  }  
};
```

CRUD - DELETE

```
// archivo controllers/PersonaController.js
// Borrar Persona PUT
exports.borrarPersona = async function(req, res) {
  var id = req.params.id;
  //Buscamos por ID, eliminamos el objeto y devolvemos el objeto
  //borrado en un JSON

  Const persona = await Persona.findByPk(id);
  const perRemoved = await persona.destroy({returning: true,
  checkExistance: true});
  if(! perRemoved)
    return res.status(500).send({ message: 'Error en el servidor - no
se pudo borrar' });
  else{
    return return res.status(200).send({
    nota: personaRemoved});
  }
};
```

Manejador para Persona

```
// archivo routes/persona.js
var express = require('express');
var router = express.Router();
var personaController =

require('../controllers/personaController');
router.get('/', personaController.listarPersonas)
router.get('/:id', personaController.detallePersona);
router.post('/create', personaController.crearPersona)
router.delete('/delete/:id', personaController.borrarPersona);
router.put('/update/:id', personaController.actualizarPersona);

module.exports = router;
```

```
//archivo app.js
...
var persona = require('./routes/persona');
app.use('/persona', persona);
```


Ejercicio - Paso 1

- Crear los Modelos para Autor y Libro.
- De Autor interesa registrar nombres (obligatorio), apellidos (Obligatorio), documento (Obligatorio), fecha de nacimiento, mail (obligatorio),
- Para Libro interesa registrar el título, isbn (con formato correcto), género (enumeración), número de edición, y cantidad de páginas.
- Para cada modelo también es necesario registrar la fecha de creación y una fecha de borrado (para borrado lógico)

Ejercicio - Paso 2

- Crear las **Rutas** y **Controladores** para Autor y Libro.
- Utilice `express.Router()` para gestionar las rutas.
- Interesa generar los CRUD para gestionar los Libros y Autores.
- Además interesa implementar un buscador de libros que pueda filtrar por:
 - Autor (retorna todos los libros de un autor)
 - Género (retorna todos los libros de un género)
 - Nombre (retorna todos los libros cuyo parámetro se encuentre en el nombre de los libros)
- Desarrolle las vistas necesarias para interactuar con las funcionalidades de la aplicación

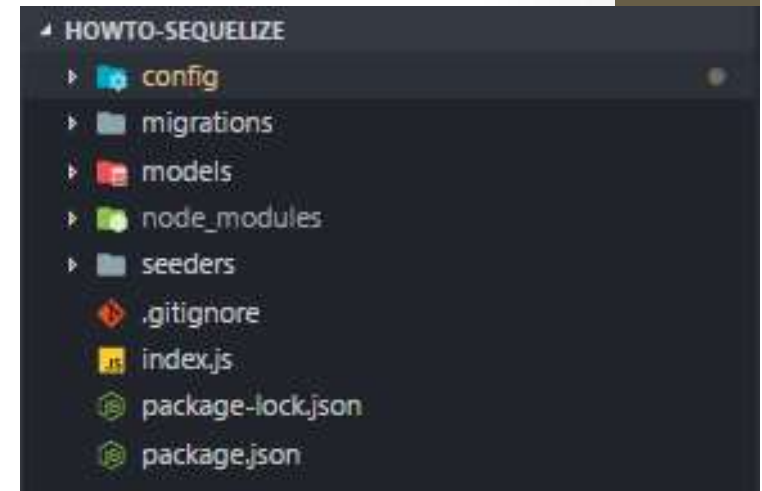
Resumen de Pasos para una appMVC Express-Sequelize-Mysql

- 1.- `npm install express-generator -g` (Si no lo tenemos)
- 2.- `express myapp -pug` (Crea la carpeta del proyecto y la estructura de una appweb express)
- 3.- `cd myapp & npm install` (Instala todas las librerías)
- 4.- `npm install --save-dev nodemon` (opcional)
- 5.- Actualizar package.json para hacer funcionar nodemon

```
"scripts": {  
  "start": "node ./bin/www",  
  "devstart": "nodemon ./bin/www"  
},
```

Resumen de Pasos para una appMVC Express-Sequelize-Mysql

- 6.- `npm install --save sequelize`
- 7.- `npm install --save mysql2`
- 8.- `npm run start` (levantar el servidor)
- 9.- probar <http://localhost:3000>



- 10.- `npm install -g sequelize-cli` (Setea herramienta de Sequelize)
(Solo si no la tenemos)
- 11.- `sequelize init` (Crea estructura de carpetas neces.)

Resumen de Pasos para una appMVC Express-Sequelize-Mysql

12.- Preparamos los paths de las carpetas sequelize

Creamos un archivo en la raíz **.sequelizerc** y agregamos:

```
const path = require('path');
module.exports = {
  "config": path.resolve('./config', 'config.json'),
  "models-path": path.resolve('./models'),
  "seeders-path": path.resolve('./seeders'),
  "migrations-path": path.resolve('./migrations')
};
```

Resumen de Pasos para una appMVC Express-Sequelize-Mysql

13.- Configuramos los datos de conexión para el archivo config.js
Que utilizara la herramienta sequelize-cli para correr migraciones.

```
"development": {  
  "username": "root",  
  "password": "",  
  "database": "database_development",  
  "host": "127.0.0.1",  
  "dialect": "mysql"  
},
```

14.- Crear la Base de Datos Vacía

Resumen de Pasos para una appMVC Express-Sequelize-Mysql

15.- Generar los Modelos (opcional, si usan migraciones)

```
sequelize model:create --name User --attributes username:string
```

El commando generará un archive `user.js` en la carpeta models y Generará un archive `xxxx--create-user.js` en migrations.

16.- Incluir en app.js las rutas necesarias (y crear controladores)

```
var usersRouter = require('./routes/users');  
app.use('/users', usersRouter);
```

17.- Ejecutar las migraciones para crear las tablas (opcional)

```
sequelize db:migrate
```

Resumen de Pasos para una appMVC Express-Sequelize-Mysql

18 – Crear Enrutamiento, controladores

19.- Crear nuevos modelos, vistas, controladores siempre teniendo en cuenta actualizar las migraciones para tenerlas sincronizadas con la Base de Datos.