JavaFX

Documentación Técnica Avanzada

Proyecto: ViMap

Versión 1.0 · 2025

Documento de uso interno. Cubre arquitectura MVC, componentes UI, layouts, CSS, eventos, FXML, animaciones y conexión con base de datos.

Índice de Contenidos

1. Introducción a JavaFX y Arquitectura MVC

JavaFX es el framework moderno de Java para construir interfaces gráficas de escritorio. Sustituyó a Swing como la opción recomendada por Oracle desde Java 8, y desde Java 11 se distribuye de manera independiente del JDK como proyecto OpenJFX. Su diseño se basa en un grafo de escena (Scene Graph) que organiza todos los nodos visuales en una jerarquía de árbol, facilitando la composición, transformación y animación de elementos.

✦ ¿Por qué JavaFX para ViMap?

JavaFX ofrece renderizado acelerado por GPU, soporte nativo para CSS, un sistema de propiedades y bindings reactivos, y una separación limpia de presentación (FXML) y lógica (Java/Kotlin). Ideal para aplicaciones de mapas y visualización.

1.1 Ciclo de vida de una aplicación

Toda aplicación JavaFX extiende la clase **Application** y su flujo de vida sigue tres métodos clave: **init()**, **start(Stage)** y **stop()**. El método start() es el único obligatorio y recibe el Stage primario.

```
package com.vimap;

import javafx.application.Application;
import javafx.stage.Stage;

public class ViMapApp extends Application {

@Override
public void init() throws Exception {
// Se ejecuta antes del hilo de UI.
// Ideal para inicializar servicios, cargar configs, conectar BD.
super.init();
}

@Override
```

```
public void start(Stage primaryStage) {

// Punto de entrada de la UI.

// Aquí se construye o carga la escena raíz.

var loader = new FXMLLoader(getClass().getResource("/fxml/main.fxml"));

Parent root = loader.load();

Scene scene = new Scene(root, 1280, 800);

scene.getStylesheets().add(getClass().getResource("/css/vimap.css").toExternalForm
());

primaryStage.setTitle("ViMap");

primaryStage.setScene(scene);

primaryStage.show();

}


@Override

public void stop() throws Exception {

// Limpieza: cerrar conexiones, guardar estado.

super.stop();

}


public static void main(String[] args) {

launch(args);

}

}
```

1.2 Arquitectura MVC en JavaFX

JavaFX promueve de forma natural el patrón MVC (Modelo-Vista-Controlador), donde el FXML actúa como la Vista, los POJOs/JavaBeans como el Modelo, y las clases Controller como el Controlador. Para proyectos de complejidad media-alta como ViMap se recomienda adoptar **MVP** (Model-View-Presenter) o **MVVM** usando las propiedades reactivas de JavaFX.

Capa	Responsabilidad	Tecnología

Modelo	Datos, reglas de negocio, acceso a BD	POJO + JavaBeans Properties
Vista	Presentación visual	FXML + CSS
Controlador	Lógica de UI, eventos, bindings	Java Controller class
Servicio	Lógica de negocio independiente de UI	Service classes + interfaces

1.3 Estructura de proyecto recomendada para ViMap

```
vimap/
■■■ src/main/
■ ■■■ java/com/vimap/
■ ■ ■■■ app/ # Application entry point
■ ■ ■■■ controller/ # FXML Controllers
■ ■ ■■■ model/ # Domain models
■ ■ ■■■ service/ # Business logic
■ ■ ■■■ repository/ # Data access (BD)
■ ■ ■■■ viewmodel/ # Propiedades reactivas (MVVM)
■ ■ ■■■ util/ # Helpers, conversores
■ ■■■ resources/
■ ■■■ fxml/ # Vistas FXML
■ ■■■ css/ # Hojas de estilo
■ ■■■ images/ # Iconos y recursos gráficos
■■■ src/test/ # Tests unitarios e integración
```

✦ Consejo de arquitectura

Separa estrictamente el acceso a datos en repositorios. Nunca accedas a la BD directamente desde un Controller. Usa un Service que llame al Repository. Esto facilita el testing y el mantenimiento a largo plazo.

2. Layouts y Escenas

El Scene Graph de JavaFX organiza todos los nodos en un árbol donde la raíz es siempre un contenedor (layout). Elegir el layout correcto es crucial para la adaptabilidad y el mantenimiento de la interfaz.

2.1 Scene Graph y Stage

La jerarquía básica es: **Stage** → **Scene** → **Parent (Root Node)** → **Nodos hijos**. El Stage representa la ventana del sistema operativo. La Scene gestiona el grafo de nodos y los estilos CSS. Puede haber múltiples Scenes, pero solo una activa por Stage.

```
// Crear Stage secundario (ventana modal)

Stage dialog = new Stage();

dialog.initModality(Modality.APPLICATION_MODAL);

dialog.initOwner(primaryStage);


FXMLLoader loader = new FXMLLoader(getClass().getResource("/fxml/dialog.fxml"));

Scene dialogScene = new Scene(loader.load(), 400, 300);

dialogScene.getStylesheets().add(getClass().getResource("/css/vimap.css").toExtern
alForm());


dialog.setScene(dialogScene);

dialog.setTitle("Detalles del Punto");

dialog.showAndWait(); // Bloquea hasta que se cierra
```

2.2 BorderPane — Layout principal

BorderPane divide el espacio en cinco zonas: **TOP, BOTTOM, LEFT, RIGHT y CENTER**. Es el layout más adecuado para la ventana principal de ViMap, con barra de menú arriba, panel de herramientas a la izquierda, mapa en el centro y detalles a la derecha.

```
// Estructura principal ViMap

BorderPane root = new BorderPane();
```

```
// Top: MenuBar + ToolBar
VBox topContainer = new VBox();
topContainer.getChildren().addAll(buildMenuBar(), buildToolBar());
root.setTop(topContainer);

// Center: Mapa principal
MapView mapView = new MapView();
root.setCenter(mapView);

// Left: Panel de capas
LayerPanel layerPanel = new LayerPanel();
root.setLeft(layerPanel);

// Right: Panel de propiedades (colapsable)
PropertiesPanel propsPanel = new PropertiesPanel();
root.setRight(propsPanel);

// Bottom: Barra de estado
StatusBar statusBar = new StatusBar();
root.setBottom(statusBar);
```

2.3 GridPane — Formularios y cuadrículas

GridPane organiza nodos en filas y columnas. Es ideal para formularios. Soporta colspan y rowspan, y permite definir restricciones de tamaño por columna y fila.

```
GridPane grid = new GridPane();
grid.setHgap(12);
grid.setVgap(10);
grid.setPadding(new Insets(20));

// Restricciones de columna
```

```
ColumnConstraints col1 = new ColumnConstraints(120); // etiquetas fijas
ColumnConstraints col2 = new ColumnConstraints();
col2.setHgrow(Priority.ALWAYS); // campos expandibles
grid.getColumnConstraints().addAll(col1, col2);

// Añadir filas
grid.add(new Label("Nombre:"), 0, 0);
grid.add(nameField, 1, 0);

grid.add(new Label("Coordenadas:"), 0, 1);
grid.add(coordField, 1, 1);

// Botón que ocupa ambas columnas
Button saveBtn = new Button("Guardar");
GridPane.setColumnSpan(saveBtn, 2);
GridPane.setHalignment(saveBtn, HPos.RIGHT);
grid.add(saveBtn, 0, 2);
```

2.4 HBox y VBox — Layouts lineales

```
// VBox: apila verticalmente
VBox sidebar = new VBox(8); // spacing = 8px
sidebar.setPadding(new Insets(16));
sidebar.setPrefWidth(250);
VBox.setVgrow(mapList, Priority.ALWAYS); // expande el ListView
sidebar.getChildren().addAll(searchField, filterBox, mapList, addButton);

// HBox: distribuye horizontalmente
HBox toolbar = new HBox(6);
toolbar.setAlignment(Pos.CENTER_LEFT);
toolbar.setPadding(new Insets(6, 12, 6, 12));
```

```
Region spacer = new Region();
HBox.setHgrow(spacer, Priority.ALWAYS); // empuja elementos a la derecha
toolbar.getChildren().addAll(zoomInBtn, zoomOutBtn, spacer, settingsBtn);
```

2.5 StackPane — Superposición de nodos

StackPane apila sus hijos uno encima del otro, alineados por defecto al centro. Perfecto para overlays: mostrar un spinner sobre el mapa, o un tooltip personalizado.

```
StackPane mapContainer = new StackPane();

MapView map = new MapView();
ProgressIndicator spinner = new ProgressIndicator();
spinner.setVisible(false);
spinner.setMaxSize(60, 60);

Label noDataLabel = new Label("Sin datos disponibles");
noDataLabel.setVisible(false);
StackPane.setAlignment(noDataLabel, Pos.CENTER);

mapContainer.getChildren().addAll(map, spinner, noDataLabel);

// Para mostrar el spinner:
// spinner.setVisible(true);
// spinner.toFront();
```

2.6 SplitPane y AnchorPane

```
// SplitPane: panel divisible con divisor arrastrable
SplitPane splitPane = new SplitPane();
splitPane.setOrientation(Orientation.HORIZONTAL);
splitPane.getItems().addAll(layerPanel, mapView, propertiesPanel);
splitPane.setDividerPositions(0.2, 0.8); // 20% | 60% | 20%
```

```
// AnchorPane: anclaje absoluto a los bordes

AnchorPane anchor = new AnchorPane();

Button fab = new Button("+");

fab.getStyleClass().add("fab");

AnchorPane.setBottomAnchor(fab, 24.0);

AnchorPane.setRightAnchor(fab, 24.0);

anchor.getChildren().add(fab);
```

✦ Regla de oro para layouts

Nunca uses coordenadas absolutas (x, y fijos) en producción. Siempre usa layouts responsivos que se adapten al tamaño de la ventana. Combina Priority.ALWAYS con setHgrow/setVgrow para que los elementos crezcan correctamente.

3. Componentes UI: Botones, Tablas y Formularios

3.1 Button y sus variantes

JavaFX ofrece varios tipos de botón. La diferencia clave entre Button, ToggleButton y RadioButton es el comportamiento de selección. MenuButton y SplitMenuButton son útiles para acciones con submenús.

```
// Button básico con icono SVG

Button addPointBtn = new Button("Añadir Punto");

addPointBtn.getStyleClass().addAll("btn-primary", "btn-icon");

addPointBtn.setGraphic(new ImageView(new Image("/icons/pin.png", 16, 16, true,
true)));

addPointBtn.setDefaultButton(true); // responde a Enter


// ToggleButton para modo de edición

ToggleButton editMode = new ToggleButton("Editar");

editMode.selectedProperty().addListener((obs, old, selected) -> {

mapView.setEditMode(selected);

editMode.setText(selected ? "Editando..." : "Editar");

});


// ToggleGroup para selección exclusiva

ToggleGroup toolGroup = new ToggleGroup();

selectTool.setToggleGroup(toolGroup);

drawTool.setToggleGroup(toolGroup);

measureTool.setToggleGroup(toolGroup);

selectTool.setSelected(true);


// SplitMenuButton: acción principal + opciones
```

```
SplitMenuButton exportBtn = new SplitMenuButton();
exportBtn.setText("Exportar KML");
exportBtn.setOnAction(e -> exportService.exportKml());
exportBtn.getItems().addAll(
new MenuItem("Exportar GeoJSON"),
new MenuItem("Exportar CSV"),
new MenuItem("Exportar Shapefile")
);
```

3.2 TableView — Tabla avanzada

TableView es uno de los componentes más potentes de JavaFX. Soporta ordenación, filtrado, edición in-line y celdas personalizadas. Siempre debe enlazarse con un ObservableList para actualizaciones reactivas.

```
// Modelo
public class MapPoint {
private final StringProperty name;
private final DoubleProperty latitude;
private final DoubleProperty longitude;
private final ObjectProperty<LocalDate> createdAt;
// Constructor + getters/setters con propiedades JavaFX
}

// Tabla
TableView<MapPoint> table = new TableView<>();
table.setPlaceholder(new Label("No hay puntos registrados"));

// Columna de nombre con edición in-line
TableColumn<MapPoint, String> nameCol = new TableColumn<>("Nombre");
nameCol.setCellValueFactory(new PropertyValueFactory<>("name"));
nameCol.setCellFactory(TextFieldTableCell.forTableColumn());
```

```
nameCol.setOnEditCommit(e -> e.getRowValue().setName(e.getNewValue()));
nameCol.setPrefWidth(200);

// Columna de latitud con formato
TableColumn<MapPoint, Double> latCol = new TableColumn<>("Latitud");
latCol.setCellValueFactory(new PropertyValueFactory<>("latitude"));
latCol.setCellFactory(col -> new TableCell<>() {
@Override
protected void updateItem(Double value, boolean empty) {
super.updateItem(value, empty);
setText(empty || value == null ? null : String.format("%.6f", value));
}
});

// Columna de acciones (botones por fila)
TableColumn<MapPoint, Void> actionsCol = new TableColumn<>("Acciones");
actionsCol.setCellFactory(col -> new TableCell<>() {
private final Button deleteBtn = new Button("Eliminar");
{
deleteBtn.getStyleClass().add("btn-danger-sm");
deleteBtn.setOnAction(e -> {
MapPoint point = getTableView().getItems().get(getIndex());
pointService.delete(point);
});
}
@Override
protected void updateItem(Void v, boolean empty) {
super.updateItem(v, empty);
setGraphic(empty ? null : deleteBtn);
```

```
}
});

table.getColumns().addAll(nameCol, latCol, lonCol, actionsCol);
table.setEditable(true);

// Filtrado con FilteredList
ObservableList<MapPoint> masterList = pointService.getPoints();
FilteredList<MapPoint> filteredList = new FilteredList<>(masterList, p -> true);
searchField.textProperty().addListener((obs, old, text) -> {
filteredList.setPredicate(point ->
text == null || text.isEmpty() ||
point.getName().toLowerCase().contains(text.toLowerCase()));
});
SortedList<MapPoint> sortedList = new SortedList<>(filteredList);
sortedList.comparatorProperty().bind(table.comparatorProperty());
table.setItems(sortedList);
```

3.3 Formularios — TextField, ComboBox, DatePicker

```
// TextField con validación en tiempo real
TextField latField = new TextField();
latField.setPromptText("Ej: 40.416775");
latField.textProperty().addListener((obs, old, text) -> {
try {
double val = Double.parseDouble(text);
boolean valid = val >= -90 && val <= 90;
latField.pseudoClassStateChanged(
PseudoClass.getPseudoClass("error"), !valid);
latField.pseudoClassStateChanged(
```

```
PseudoClass.getPseudoClass("valid"), valid);
} catch (NumberFormatException e) {
latField.pseudoClassStateChanged(
PseudoClass.getPseudoClass("error"), true);
}
});

// ComboBox con renderizado personalizado
ComboBox<Category> categoryBox = new ComboBox<>();
categoryBox.setItems(FXCollections.observableArrayList(Category.values()));
categoryBox.setCellFactory(lv -> new ListCell<>() {
@Override
protected void updateItem(Category cat, boolean empty) {
super.updateItem(cat, empty);
if (empty || cat == null) { setGraphic(null); setText(null); }
else {
Circle dot = new Circle(6, cat.getColor());
setText(cat.getDisplayName());
setGraphic(dot);
}
}
});
categoryBox.setButtonCell(categoryBox.getCellFactory().call(null));

// DatePicker con formato personalizado
DatePicker datePicker = new DatePicker(LocalDate.now());
datePicker.setConverter(new LocalDateStringConverter(
DateTimeFormatter.ofPattern("dd/MM/yyyy"),
DateTimeFormatter.ofPattern("dd/MM/yyyy")));
```

```
datePicker.setDayCellFactory(picker -> new DateCell() {

@Override public void updateItem(LocalDate date, boolean empty) {

super.updateItem(date, empty);

setDisabled(date.isAfter(LocalDate.now()));

}

});
```

■ **Advertencia:** Nunca hagas validaciones de formulario solo en el botón de guardar. Valida en tiempo real con listeners en cada campo para mejorar la UX.

4. CSS y Estilos en JavaFX

JavaFX implementa un subconjunto de CSS 2 con extensiones propias. Las propiedades JavaFX se identifican con el prefijo **-fx-**. La hoja de estilo por defecto es **modena.css** (desde JavaFX 8). Puedes sobreescribirla parcial o totalmente con tus propios archivos CSS.

4.1 Estructura del archivo CSS para ViMap

```
/* vimap.css — Hoja de estilos principal */,


/* ■■ Variables globales (JavaFX 17+) ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
*/

.root {

-fx-font-family: "Segoe UI", "SF Pro Display", sans-serif;

-fx-font-size: 13px;


/* Paleta */,

vimap-primary: #1A1A2E;

vimap-accent: #E94560;

vimap-bg: #F8F9FA;

vimap-surface: #FFFFFF;

vimap-border: #DEE2E6;

vimap-text: #212529;

vimap-muted: #6C757D;

}


/* ■■ Botones
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ */

.btn-primary {

-fx-background-color: vimap-primary;

-fx-text-fill: white;
```



```
-fx-background-radius: 6px;

-fx-padding: 8px 18px;

-fx-cursor: hand;

-fx-font-weight: bold;

}

.btn-primary:hover {

-fx-background-color: derive(vimap-primary, 20%);

-fx-effect: dropshadow(gaussian, rgba(0,0,0,0.2), 8, 0, 0, 2);

}

.btn-primary:pressed {

-fx-background-color: derive(vimap-primary, -10%);

-fx-effect: none;

}


/* ■■ TextField con estados de validación ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
*/

.text-field {

-fx-background-color: vimap-surface;

-fx-border-color: vimap-border;

-fx-border-radius: 4px;

-fx-background-radius: 4px;

-fx-padding: 6px 10px;

}

.text-field:focused {

-fx-border-color: vimap-primary;

-fx-border-width: 1.5px;

-fx-effect: dropshadow(gaussian, rgba(26,26,46,0.15), 6, 0, 0, 0);

}

.text-field:error {
```

```
-fx-border-color: #DC3545;

}

.text-field:valid {

-fx-border-color: #28A745;

}


/* ■■ TableView
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ */

.table-view {

-fx-background-color: vimap-surface;

-fx-border-color: vimap-border;

-fx-border-radius: 6px;

}

.table-view .column-header {

-fx-background-color: vimap-bg;

-fx-font-weight: bold;

-fx-text-fill: vimap-text;

-fx-padding: 10px 12px;

}

.table-row-cell:selected {

-fx-background-color: derive(vimap-primary, 80%);

-fx-text-fill: vimap-text;

}
```

4.2 PseudoClasses — Estados dinámicos

Las PseudoClasses permiten cambiar el estilo CSS de un nodo desde Java, como si fueran pseudoclases CSS (:hover, :focused) pero completamente personalizadas.

```
// Definir la pseudoclase (una sola vez, preferiblemente como constante)

private static final PseudoClass ERROR_CLASS =
```



```
PseudoClass.getPseudoClass("error");

private static final PseudoClass LOADING_CLASS =

PseudoClass.getPseudoClass("loading");


// Activar/desactivar

node.pseudoClassStateChanged(ERROR_CLASS, true); // activa :error

node.pseudoClassStateChanged(ERROR_CLASS, false); // desactiva :error


// En CSS:

// .my-button:loading { -fx-opacity: 0.6; }

// .my-button:error { -fx-border-color: red; }
```

4.3 Lookup — Acceso a subnodos CSS

```
// Acceder a subnodos internos de controles compuestos

// (útil para personalizar componentes complejos como Spinner, DatePicker)

scene.getRoot().applyCss();

scene.getRoot().layout();


TextField editor = (TextField) spinner.lookup(".text-field");

editor.setAlignment(Pos.CENTER);


// ScrollBar interno de un ListView

Set<Node> scrollBars = listView.lookupAll(".scroll-bar");

scrollBars.forEach(sb -> sb.setStyle("-fx-opacity: 0;"));
```

✦ Consejo de rendimiento CSS

Evita el uso de setStyle() con strings inline en bucles o celdas de tabla, ya que JavaFX reparsea el CSS cada vez. Usa getStyleClass().add() con clases predefinidas en tu archivo .css. Es más rápido y más mantenible.

5. Eventos y Manejo de Interacciones

JavaFX usa un sistema de eventos basado en bubbling y capturing similar al DOM web. Cada nodo puede registrar Event Handlers (captura todas las fases) y Event Filters (solo fase de captura, antes de que el evento llegue al target).

5.1 Eventos de ratón

```
// Click simple vs doble click
node.setOnMouseClicked(event -> {
if (event.getButton() == MouseButton.PRIMARY) {
if (event.getClickCount() == 2) {
handleDoubleClick(event);
} else {
handleSingleClick(event);
}
} else if (event.getButton() == MouseButton.SECONDARY) {
showContextMenu(event.getScreenX(), event.getScreenY());
}
});

// Arrastrar un nodo (drag & drop libre)
final double[] dragDelta = new double[2];

node.setOnMousePressed(e -> {
dragDelta[0] = node.getLayoutX() - e.getSceneX();
dragDelta[1] = node.getLayoutY() - e.getSceneY();
node.setCursor(Cursor.MOVE);
});
```

```
node.setOnMouseDragged(e -> {
node.setLayoutX(e.getSceneX() + dragDelta[0]);
node.setLayoutY(e.getSceneY() + dragDelta[1]);
});

node.setOnMouseReleased(e -> node.setCursor(Cursor.DEFAULT));
```

5.2 Propiedades y Bindings — El corazón reactivo de JavaFX

El sistema de propiedades de JavaFX permite que los cambios en un valor se propaguen automáticamente a otros. Hay tres tipos principales: **bind()** (unidireccional), **bindBidirectional()** y listeners.

```
// Propiedad simple
IntegerProperty zoom = new SimpleIntegerProperty(10);

// Binding unidireccional: zoomLabel siempre refleja zoom
zoomLabel.textProperty().bind(
Bindings.createStringBinding(
() -> "Zoom: " + zoom.get() + "x",
zoom
)
);

// Binding bidireccional entre dos controles
slider.valueProperty().bindBidirectional(zoom);

// Expresiones complejas con Bindings API
BooleanBinding canSave = nameField.textProperty().isNotEmpty()
.and(latField.textProperty().isNotEmpty())
.and(lonField.textProperty().isNotEmpty());
saveButton.disableProperty().bind(canSave.not());
```

```
// Listener explícito (cuando necesitas lógica compleja)

zoom.addListener((ObservableValue<? extends Number> obs, Number old, Number now)
-> {

if (now.intValue() > 18) {

showDetailLayer();

} else {

hideDetailLayer();

}

});
```

5.3 Tareas en segundo plano — Task y Service

Toda operación lenta (BD, red, IO) DEBE ejecutarse fuera del hilo de UI (JavaFX Thread). JavaFX provee **Task** para operaciones únicas y **Service** para operaciones reutilizables. Ambas permiten reportar progreso y manejar errores de forma segura.

```
// Task: carga de datos desde BD

Task<List<MapPoint>> loadTask = new Task<>() {

@Override

protected List<MapPoint> call() throws Exception {

updateMessage("Conectando a base de datos...");

List<MapPoint> points = pointRepository.findAll();

int total = points.size();

for (int i = 0; i < total; i++) {

processPoint(points.get(i));

updateProgress(i + 1, total);

updateMessage("Cargando punto " + (i+1) + " de " + total);

}

return points;

}

};
```

```
// Binding de progreso a la UI (hilo seguro)
progressBar.progressProperty().bind(loadTask.progressProperty());
statusLabel.textProperty().bind(loadTask.messageProperty());

// Callbacks en el hilo de UI
loadTask.setOnSucceeded(e -> {
List<MapPoint> points = loadTask.getValue();
pointTable.setItems(FXCollections.observableArrayList(points));
progressBar.setVisible(false);
});

loadTask.setOnFailed(e -> {
Throwable error = loadTask.getException();
showErrorDialog("Error al cargar puntos", error.getMessage());
progressBar.setVisible(false);
});

// Ejecutar en hilo separado
Thread thread = new Thread(loadTask);
thread.setDaemon(true);
thread.start();

// Alternativa: usar ExecutorService
// executor.submit(loadTask);
```

5.4 EventBus — Comunicación entre componentes desacoplados

```
// EventBus simple con propiedades
public class AppEvents {
private static final ObjectProperty<MapPoint> selectedPoint =
```

```
new SimpleObjectProperty<>();


public static ObjectProperty<MapPoint> selectedPointProperty() {

return selectedPoint;

}


public static void selectPoint(MapPoint point) {

selectedPoint.set(point);

}

}


// En MapController: publica el evento

mapView.setOnPointClicked(point -> AppEvents.selectPoint(point));


// En PropertiesController: se suscribe

AppEvents.selectedPointProperty().addListener((obs, old, point) -> {

if (point != null) populateForm(point);

});
```

■ **Advertencia:** Nunca actualices nodos de UI desde un hilo que no sea el JavaFX Application Thread. Usa Platform.runLater(() -> { /* actualizar UI */ }); si necesitas hacerlo desde otro hilo.

6. FXML y SceneBuilder

FXML es un formato XML que permite definir la estructura de la UI de forma declarativa, separando completamente la presentación de la lógica de negocio. SceneBuilder es el editor visual WYSIWYG para crear archivos FXML sin escribir XML manualmente.

6.1 Anatomía de un archivo FXML

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.layout.*?>
<?import javafx.scene.control.*?>
<?import javafx.geometry.Insets?>

<BorderPane xmlns="http://javafx.com/javafx"
xmlns:fx="http://javafx.com/fxml"
fx:controller="com.vimap.controller.MainController"
prefWidth="1280" prefHeight="800">

<top>
<VBox>
<MenuBar fx:id="menuBar"/>
<ToolBar fx:id="toolBar" styleClass="main-toolbar">
<Button fx:id="addPointBtn" text="+ Punto"
styleClass="btn-primary"
onAction="#handleAddPoint"/>
<Separator orientation="VERTICAL"/>
<TextField fx:id="searchField"
promptText="Buscar punto..."
prefWidth="250"/>
```

```
</ToolBar>

</VBox>

</top>


<center>

<StackPane fx:id="mapContainer">

<!-- Mapa se inyecta programáticamente -->

</StackPane>

</center>


<bottom>

<HBox styleClass="status-bar" alignment="CENTER_LEFT" spacing="12">

<padding><Insets top="4" bottom="4" left="12" right="12"/></padding>

<Label fx:id="coordsLabel" text="Lat: -- | Lon: --"/>

<Region HBox.hgrow="ALWAYS"/>

<Label fx:id="pointCountLabel" text="0 puntos"/>

</HBox>

</bottom>


</BorderPane>
```

6.2 Controller vinculado a FXML

```
public class MainController implements Initializable {


// Inyectados por FXMLLoader (deben coincidir con fx:id)

@FXML private StackPane mapContainer;

@FXML private Label coordsLabel;

@FXML private Label pointCountLabel;

@FXML private TextField searchField;
```

```
// Servicios — inyectados manualmente o con DI
private PointService pointService;
private MapViewModel viewModel;

@Override
public void initialize(URL location, ResourceBundle resources) {
// initialize() se llama DESPUÉS de que @FXML se inyectan.
// Aquí se hacen bindings, se cargan datos iniciales, etc.
pointService = ServiceLocator.get(PointService.class);
viewModel = new MapViewModel(pointService);

// Binding reactivo
pointCountLabel.textProperty().bind(
Bindings.createStringBinding(
() -> viewModel.getPoints().size() + " puntos",
viewModel.getPoints()
)
);

searchField.textProperty().bindBidirectional(viewModel.searchQueryProperty());
}

// Handler definido en FXML con onAction="#handleAddPoint"
@FXML
private void handleAddPoint(ActionEvent event) {
viewModel.startAddPointMode();
}
}
```

6.3 Pasar datos entre controladores

```
// Técnica recomendada: inyectar datos antes de mostrar

private void openPointDetail(MapPoint point) throws IOException {

FXMLLoader loader = new FXMLLoader(

getClass().getResource("/fxml/point-detail.fxml")

);

Parent view = loader.load();


// Inyectar ANTES de mostrar la ventana

PointDetailController controller = loader.getController();

controller.setPoint(point);

controller.setOnSave(savedPoint -> {

viewModel.updatePoint(savedPoint);

});


Stage stage = new Stage();

stage.initModality(Modality.WINDOW_MODAL);

stage.initOwner(mapContainer.getScene().getWindow());

stage.setScene(new Scene(view));

stage.setTitle("Detalle: " + point.getName());

stage.showAndWait();

}
```

✦ SceneBuilder — Integración con IntelliJ IDEA

Instala el plugin "SceneBuilder" en IntelliJ. Haz clic derecho sobre cualquier .fxml → "Open in SceneBuilder". Los cambios se reflejan al guardar. Recuerda configurar la ruta de SceneBuilder en Preferences > Languages & Frameworks > JavaFX.

7. Animaciones y Efectos

JavaFX incluye un sistema de animación completo basado en Timelines y Transitions. Todas las animaciones se ejecutan en el hilo de UI y están sincronizadas con el ciclo de renderizado (60 fps por defecto).

7.1 Timeline — Animaciones de fotogramas clave

```
// Parpadeo de un marcador en el mapa
Timeline blink = new Timeline(
new KeyFrame(Duration.ZERO,
new KeyValue(marker.opacityProperty(), 1.0)),
new KeyFrame(Duration.millis(500),
new KeyValue(marker.opacityProperty(), 0.3, Interpolator.EASE_BOTH)),
new KeyFrame(Duration.millis(1000),
new KeyValue(marker.opacityProperty(), 1.0, Interpolator.EASE_BOTH))
);
blink.setCycleCount(Animation.INDEFINITE);
blink.setAutoReverse(false);
blink.play();

// Animar múltiples propiedades simultáneamente
Timeline expand = new Timeline(
new KeyFrame(Duration.millis(300),
new KeyValue(panel.prefWidthProperty(), 280, Interpolator.EASE_OUT),
new KeyValue(panel.opacityProperty(), 1.0, Interpolator.EASE_IN)
)
);
expand.play();
```

7.2 Transitions — Animaciones predefinidas

```
// FadeTransition — entrada suave de un panel
FadeTransition fadeIn = new FadeTransition(Duration.millis(250), node);
fadeIn.setFromValue(0.0);
fadeIn.setToValue(1.0);
fadeIn.play();

// ScaleTransition — efecto de "pop" en botón
ScaleTransition pop = new ScaleTransition(Duration.millis(120), button);
pop.setFromX(1.0); pop.setFromY(1.0);
pop.setToX(1.15); pop.setToY(1.15);
pop.setAutoReverse(true);
pop.setCycleCount(2);

// TranslateTransition — deslizar panel lateral
TranslateTransition slide = new TranslateTransition(Duration.millis(300),
sidePanel);
slide.setFromX(-260);
slide.setToX(0);
slide.setInterpolator(Interpolator.EASE_OUT);
slide.play();

// SequentialTransition — encadenar animaciones
SequentialTransition sequence = new SequentialTransition(
new FadeTransition(Duration.millis(200), overlay),
new ScaleTransition(Duration.millis(300), dialog),
new PauseTransition(Duration.millis(100))
);
sequence.setOnFinished(e -> dialog.requestFocus());
```

```
sequence.play();

// ParallelTransition — animaciones simultáneas
ParallelTransition parallel = new ParallelTransition(
fadeIn,
new TranslateTransition(Duration.millis(300), card)
);
parallel.play();
```

7.3 AnimationTimer — Animaciones de 60fps (render loop)

```
// Útil para animaciones frame-by-frame (mapas, visualizaciones)
AnimationTimer renderLoop = new AnimationTimer() {
private long lastUpdate = 0;

@Override
public void handle(long now) {
// now está en nanosegundos
if (now - lastUpdate >= 16_666_666) { // ~60fps
updateMapOverlay();
lastUpdate = now;
}
}
};
renderLoop.start();
// Para detener: renderLoop.stop();
```

7.4 Efectos visuales — DropShadow, Blur, Glow

```
// DropShadow — sombra en tarjetas
```

```
DropShadow cardShadow = new DropShadow();
cardShadow.setRadius(12);
cardShadow.setOffsetY(4);
cardShadow.setColor(Color.color(0, 0, 0, 0.15));
card.setEffect(cardShadow);

// Glow — resaltar elemento seleccionado en mapa
Glow glow = new Glow(0.8);
InnerShadow innerShadow = new InnerShadow(6, Color.BLUE);
// Combinar efectos con chain
glow.setInput(innerShadow);
selectedMarker.setEffect(glow);

// GaussianBlur — fondo difuminado para modal
GaussianBlur blur = new GaussianBlur(8);
mainContent.setEffect(blur);
// Quitar al cerrar modal:
// mainContent.setEffect(null);

// Animar el blur de 0 a 8
DoubleProperty blurAmount = new SimpleDoubleProperty(0);
GaussianBlur animBlur = new GaussianBlur();
animBlur.radiusProperty().bind(blurAmount);
mainContent.setEffect(animBlur);

Timeline blurIn = new Timeline(
new KeyFrame(Duration.millis(200),
new KeyValue(blurAmount, 8, Interpolator.EASE_IN))
);
blurIn.play();
```

8. Conexión con Base de Datos

Para ViMap se recomienda una arquitectura de acceso a datos en capas: Controller → Service → Repository → DataSource. Nunca se accede a la BD directamente desde los controladores. Se cubren dos enfoques: JDBC puro para máximo control, e Hibernate/JPA para proyectos más grandes.

8.1 Configuración de la conexión (patrón Singleton)

```
public class DatabaseConfig {

private static final String URL = "jdbc:postgresql://localhost:5432/vimap";

private static final String USER = System.getenv("DB_USER");

private static final String PASS = System.getenv("DB_PASS");


// HikariCP — pool de conexiones de alto rendimiento

private static HikariDataSource dataSource;


static {

HikariConfig config = new HikariConfig();

config.setJdbcUrl(URL);

config.setUsername(USER);

config.setPassword(PASS);

config.setMaximumPoolSize(10);

config.setMinimumIdle(2);

config.setConnectionTimeout(30000);

config.setIdleTimeout(600000);

config.setMaxLifetime(1800000);

config.setAutoCommit(false);

dataSource = new HikariDataSource(config);

}
```

```
public static Connection getConnection() throws SQLException {
return dataSource.getConnection();
}

public static void close() {
if (dataSource != null) dataSource.close();
}
}
```

8.2 Patrón Repository con JDBC

```
public class PointRepository {

private static final String INSERT =
"INSERT INTO map_points (name, latitude, longitude, category, created_at) " +
"VALUES (?, ?, ?, ?, ?) RETURNING id";

private static final String FIND_ALL =
"SELECT id, name, latitude, longitude, category, created_at " +
"FROM map_points ORDER BY created_at DESC";

public MapPoint save(MapPoint point) throws SQLException {
try (Connection conn = DatabaseConfig.getConnection();
PreparedStatement ps = conn.prepareStatement(INSERT)) {

ps.setString(1, point.getName());
ps.setDouble(2, point.getLatitude());
ps.setDouble(3, point.getLongitude());
ps.setString(4, point.getCategory().name());
ps.setTimestamp(5, Timestamp.valueOf(LocalDateTime.now()));

ResultSet rs = ps.executeQuery();
```

```
if (rs.next()) {
point.setId(rs.getLong("id"));
}
conn.commit();
return point;
}
}

public List<MapPoint> findAll() throws SQLException {
List<MapPoint> points = new ArrayList<>();
try (Connection conn = DatabaseConfig.getConnection();
PreparedStatement ps = conn.prepareStatement(FIND_ALL);
ResultSet rs = ps.executeQuery()) {

while (rs.next()) {
points.add(mapRow(rs));
}
}
return points;
}

private MapPoint mapRow(ResultSet rs) throws SQLException {
return new MapPoint(
rs.getLong("id"),
rs.getString("name"),
rs.getDouble("latitude"),
rs.getDouble("longitude"),
Category.valueOf(rs.getString("category")),
rs.getTimestamp("created_at").toLocalDateTime()
);
```

```
}

}
```

8.3 Service Layer — Lógica de negocio

```
public class PointService {

private final PointRepository repository;

private final ObservableList<MapPoint> points =

FXCollections.observableArrayList();


public PointService(PointRepository repository) {

this.repository = repository;

}


// Carga inicial en segundo plano

public Task<Void> loadAllAsync() {

return new Task<>() {

@Override

protected Void call() throws Exception {

List<MapPoint> loaded = repository.findAll();

// Actualizar ObservableList en hilo de UI

Platform.runLater(() -> points.setAll(loaded));

return null;

}

};

}


public Task<MapPoint> saveAsync(MapPoint point) {

return new Task<>() {

@Override
```

```
protected MapPoint call() throws Exception {
// Validar
if (point.getName() == null || point.getName().isBlank())
throw new ValidationException("El nombre no puede estar vacío");
// Guardar
MapPoint saved = repository.save(point);
Platform.runLater(() -> points.add(saved));
return saved;
}
};
}

public ObservableList<MapPoint> getPoints() { return points; }
}
```

8.4 Integración en el Controller

```
@FXML
private void handleSavePoint(ActionEvent event) {
MapPoint point = buildPointFromForm();
Task<MapPoint> saveTask = pointService.saveAsync(point);

saveBtn.disableProperty().bind(saveTask.runningProperty());
progressBar.visibleProperty().bind(saveTask.runningProperty());

saveTask.setOnSucceeded(e -> {
showSuccessNotification("Punto guardado correctamente");
closeDialog();
});

saveTask.setOnFailed(e -> {
```

```
Throwable error = saveTask.getException();

if (error instanceof ValidationException) {

showFieldError(error.getMessage());

} else {

showErrorDialog("Error al guardar", error.getMessage());

}

});


executor.submit(saveTask);

}
```

■ **Advertencia:** Nunca uses credenciales de BD en el código fuente. Usa variables de entorno (System.getenv()) o un archivo de configuración externo (.env, application.properties) que esté en el .gitignore del proyecto.

9. Buenas Prácticas y Patrones de Diseño

9.1 Checklist de calidad para ViMap

✓	**UI en hilo correcto**	Toda actualización de UI va en Platform.runLater() o en callbacks de Task
✓	**Sin lógica en FXML**	Los Controllers solo orquestan; la lógica va en Services
✓	**Bindings reactivos**	Usar propiedades JavaFX en lugar de setters manuales
✓	**CSS externalizado**	Ningún setStyle() inline en producción
✓	**Pool de conexiones**	Usar HikariCP, nunca crear conexiones directas
✓	**Manejo de errores**	Task.setOnFailed() siempre definido
✓	**Resources cerrados**	try-with-resources en JDBC
✓	**Tests unitarios**	Services y Repositories testeados con JUnit 5

9.2 Patrón ViewModel para ViMap

```
// ViewModel centraliza el estado de la UI

public class MapViewModel {

private final PointService pointService;


// Estado observable

private final ObservableList<MapPoint> points =

FXCollections.observableArrayList();

private final StringProperty searchQuery =

new SimpleStringProperty("");

private final ObjectProperty<MapPoint> selectedPoint =

new SimpleObjectProperty<>();

private final BooleanProperty loading =
```

```
new SimpleBooleanProperty(false);

public MapViewModel(PointService pointService) {
this.pointService = pointService;
// Filtrado automático cuando cambia la búsqueda
searchQuery.addListener((obs, old, query) -> filterPoints(query));
}

public void loadPoints() {
Task<Void> task = pointService.loadAllAsync();
loading.bind(task.runningProperty());
new Thread(task).start();
}

// Getters de propiedades (no valores)
public ObservableList<MapPoint> getPoints() { return points; }
public StringProperty searchQueryProperty() { return searchQuery; }
public ObjectProperty<MapPoint> selectedPointProperty() { return selectedPoint; }
public BooleanProperty loadingProperty() { return loading; }
}
```

9.3 Errores comunes y cómo evitarlos

■ Memory leaks con listeners

Registrar un listener en una propiedad global sin desregistrarlo. Solución: guardar la referencia y llamar removeListener() al destruir el controller, o usar WeakChangeListener.

■ Bloquear el hilo de UI

Hacer Thread.sleep() o IO dentro de un EventHandler. Solución: siempre usar Task o Service para operaciones lentas.

■ FXMLLoader.load() en el constructor

El FXML no está completamente inicializado en el constructor. Solución: toda lógica de inicialización va en initialize() o en el método show().

■ Crear nodos UI fuera del JavaFX thread

Instanciar Label, Button, etc. en un Task.call(). Solución: construir los datos en el Task, crear los nodos en Platform.runLater().

✦ Recursos recomendados para JavaFX avanzado

Documentación oficial: openjfx.io | CSS Reference: docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html | Ejemplos: github.com/openjfx/samples | Comunidad: stackoverflow.com/questions/tagged/javafx

---