

`__init__.py`

```
1  # app/__init__.py
2  from .config import Config
3  from . import models
4  from . import views
5  from . import controllers
6  from . import utils
7
8  __version__ = '1.0.0'
9  __author__ = 'Tu Nombre'
```

config.py

```
1  # app/config.py
2  import os
3  from pathlib import Path
4  from datetime import datetime
5
6  class Config:
7      # Información de la aplicación
8      APP_NAME = 'Sistema de Gestión de CVs'
9      VERSION = '1.0.0'
10     DEBUG = True
11
12     # Rutas base
13     BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
14     APP_DIR = os.path.join(BASE_DIR, 'app')
15
16     # Directorios de recursos
17     RESOURCES_DIR = os.path.join(APP_DIR, 'resources')
18     TEMPLATES_DIR = os.path.join(RESOURCES_DIR, 'templates')
19     IMAGES_DIR = os.path.join(RESOURCES_DIR, 'images')
20     STYLES_DIR = os.path.join(RESOURCES_DIR, 'styles')
21
22     # Directorios de datos
23     DATA_DIR = os.path.join(BASE_DIR, 'data')
24     OUTPUT_DIR = os.path.join(DATA_DIR, 'output')
25     TEMP_DIR = os.path.join(DATA_DIR, 'temp')
26
27     # Rutas de archivos
28     DB_PATH = os.path.join(DATA_DIR, 'cv_database.db')
29     CV_TEMPLATE_PATH = os.path.join(TEMPLATES_DIR, 'cv_template.docx')
30     LOGO_PATH = os.path.join(IMAGES_DIR, 'logo.png')
31
32     # Credenciales de demo
33     DEMO_USER = 'admin'
34     DEMO_PASSWORD = 'admin123'
35
36     # Configuraciones de la aplicación
37     AREAS = [
38         'Administración',
39         'Oficina Técnica',
40         'Construcción',
41         'Calidad',
42         'SSOMA'
43     ]
44
45     TIPOS_OBRA = [
46         'Planta de Cemento y Cal',
47         'Naves Industriales',
48         'Hidroeléctrica',
49         'Central Termoeléctrica',
50         'Planta Eléctrica',
51         'Puentes',
52         'Planta Petróleo',
53         'Obras Civiles',
54         'Planta de Gas',
55         'Construcción Estructuras',
56         'Montaje de Estructura Pesada',
```

```

57         'Chancadora/Fajas (seca)',
58         'Molinos/Celda de Flotación',
59         'Túneles',
60         'Desmontaje',
61         'Electricidad & Instrumentación',
62         'Tubería Agua/Relaves',
63         'Planta Desalinizadora',
64         'Truckshop - Almacenes (Auxiliares)'
65     ]
66
67     @classmethod
68     def create_directories(cls):
69         """Crea todos los directorios necesarios para la aplicación"""
70         directories = [
71             cls.DATA_DIR,
72             cls.OUTPUT_DIR,
73             cls.TEMP_DIR,
74             cls.RESOURCES_DIR,
75             cls.TEMPLATES_DIR,
76             cls.IMAGES_DIR,
77             cls.STYLES_DIR
78         ]
79
80         for directory in directories:
81             os.makedirs(directory, exist_ok=True)
82
83     @classmethod
84     def get_output_path(cls, filename: str) -> str:
85         """Genera una ruta única para archivos de salida"""
86         timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
87         return os.path.join(cls.OUTPUT_DIR, f'{filename}_{timestamp}.docx')
88
89     @classmethod
90     def clean_temp_files(cls):
91         """Limpia archivos temporales antiguos"""
92         if os.path.exists(cls.TEMP_DIR):
93             for file in os.listdir(cls.TEMP_DIR):
94                 file_path = os.path.join(cls.TEMP_DIR, file)
95                 try:
96                     if os.path.isfile(file_path):
97                         os.unlink(file_path)
98                 except Exception as e:
99                     print(f'Error al eliminar {file_path}: {e}')

```

constants.py

```
1  class TipoDocumentoIdentidad:
2      DNI = 'DNI' # Documento Nacional de Identidad
3      CE = 'CE' # Carnet de Extranjería
4      PAS = 'PAS' # Pasaporte
5      RUC = 'RUC' # Registro Único de Contribuyentes
6      CPP = 'CPP' # Carnet de Permiso Temporal de Permanencia
7
8      @classmethod
9      def choices(cls):
10         return [
11             (cls.DNI, 'DNI'),
12             (cls.CE, 'Carnet de Extranjería'),
13             (cls.PAS, 'Pasaporte'),
14             (cls.RUC, 'RUC'),
15             (cls.CPP, 'Carnet PTP')
16         ]
17
18  class TipoObra:
19      # Industria
20      PLANTA_CEMENTO = 'Planta de Cemento y Cal'
21      NAVES_INDUSTRIALES = 'Naves Industriales'
22      HIDROELECTRICA = 'Hidroeléctrica'
23      CENTRAL_TERMO = 'Central Termoeléctrica'
24      PLANTA_ELECTRICA = 'Planta Eléctrica'
25      PUENTES = 'Puentes'
26      PLANTA_PETROLEO = 'Planta Petróleo'
27      OBRAS_CIVILES = 'Obras Civiles'
28
29      # Minería
30      PLANTA_GAS = 'Planta de Gas'
31      CONST_ESTRUCTURAS = 'Construcción Estructuras'
32      MONTAJE_ESTRUCTURAS = 'Montaje de Estructura Pesada'
33      CHANCADORA_FAJAS = 'Chancadora/Fajas (seca)'
34      MOLINOS_FLOTACION = 'Molinos/Celda de Flotación'
35      TUNELES = 'Túneles'
36      DESMONTAJE = 'Desmontaje'
37
38      # Servicios
39      ELECTRICIDAD_INSTR = 'Electricidad & Instrumentación'
40      TUBERIA_AGUA = 'Tubería Agua/Relaves'
41      PLANTA_DESALINIZADORA = 'Planta Desalinizadora'
42      TRUCKSHOP = 'Truckshop - Almacenes (Auxiliares)'
43
44      @classmethod
45      def get_all_tipos(cls):
46         return [
47             (cls.PLANTA_CEMENTO, 'Industria'),
48             (cls.NAVES_INDUSTRIALES, 'Industria'),
49             (cls.HIDROELECTRICA, 'Industria'),
50             (cls.CENTRAL_TERMO, 'Industria'),
51             (cls.PLANTA_ELECTRICA, 'Industria'),
52             (cls.PUENTES, 'Industria'),
53             (cls.PLANTA_PETROLEO, 'Industria'),
54             (cls.OBRAS_CIVILES, 'Industria'),
55             (cls.PLANTA_GAS, 'Minería'),
56             (cls.CONST_ESTRUCTURAS, 'Minería'),
```

```
57         (cls.MONTAJE_ESTRUCTURAS, 'Minería'),
58         (cls.CHANCADORA_FAJAS, 'Minería'),
59         (cls.MOLINOS_FLOTACION, 'Minería'),
60         (cls.TUNELES, 'Minería'),
61         (cls.DESMONTAJE, 'Minería'),
62         (cls.ELECTRICIDAD_INSTR, 'Servicios'),
63         (cls.TUBERIA_AGUA, 'Servicios'),
64         (cls.PLANTA_DESALINIZADORA, 'Servicios'),
65         (cls.TRUCKSHOP, 'Servicios')
66     ]
```

exceptions.py

```
1  # app/exceptions.py
2  class CVSystemException(Exception):
3      """Excepción base para el sistema"""
4      pass
5
6  class DatabaseError(CVSystemException):
7      """Errores relacionados con la base de datos"""
8      pass
9
10 class ValidationError(CVSystemException):
11     """Errores de validación de datos"""
12     pass
13
14 class TemplateError(CVSystemException):
15     """Errores relacionados con las plantillas"""
16     pass
```

logger.py

```
1  # app/logger.py
2  import logging
3  import os
4  from datetime import datetime
5  from .config import Config
6
7  def setup_logger():
8      # Crear directorio de logs si no existe
9      log_dir = os.path.join(Config.DATA_DIR, 'logs')
10     os.makedirs(log_dir, exist_ok=True)
11
12     # Configurar logger
13     log_file = os.path.join(
14         log_dir,
15         f'cv_system_{datetime.now().strftime("%Y%m%d")}.log'
16     )
17
18     logging.basicConfig(
19         level=logging.DEBUG if Config.DEBUG else logging.INFO,
20         format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
21         handlers=[
22             logging.FileHandler(log_file),
23             logging.StreamHandler()
24         ]
25     )
26
27     return logging.getLogger('cv_system')
28
29 logger = setup_logger()
```

main.py

```
1  # app/main.py
2  import sys
3  import os
4  import logging
5  from PyQt5.QtWidgets import QApplication, QMessageBox
6  from PyQt5.QtCore import Qt
7  from app.controllers.login_controller import LoginController
8  from app.controllers.main_controller import MainController
9  from app.config import Config
10 from app.utils.setup_check import verificar_plantilla_cv
11
12
13 def setup_logging():
14     logging.basicConfig(
15         level=logging.DEBUG,
16         format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
17         handlers=[
18             logging.StreamHandler(),
19             logging.FileHandler('app.log')
20         ]
21     )
22
23 class Application:
24     def __init__(self):
25         setup_logging()
26         self.logger = logging.getLogger(__name__)
27         self.logger.debug("Iniciando aplicación...")
28
29         # Crear las carpetas necesarias
30         Config.create_directories()
31
32         # Inicializar la aplicación
33         self.app = QApplication(sys.argv)
34         self.app.setStyle('Fusion')
35
36         # Cargar estilos
37         style_path = os.path.join(Config.STYLES_DIR, 'style.qss')
38         if os.path.exists(style_path):
39             with open(style_path, 'r') as f:
40                 self.app.setStyleSheet(f.read())
41
42         # Mantener referencias a los controladores
43         self.login_controller = None
44         self.main_controller = None
45
46     def start(self):
47         try:
48             # Iniciar con ventana de login
49             self.login_controller = LoginController()
50
51             # Auto-login en modo debug
52             if Config.DEBUG:
53                 self.login_controller.view.usuario_input.setText(Config.DEMO_USER)
54                 self.login_controller.view.password_input.setText(Config.DEMO_PASSWORD)
55
56             self.login_controller.view.show()
```



```

57         self.login_controller.view.loginSuccess.connect(self.on_login_success)
58
59         return self.app.exec_()
60
61     except Exception as e:
62         self.logger.error(f"Error en la aplicación: {str(e)}", exc_info=True)
63         QMessageBox.critical(None, "Error Fatal", f"Error en la aplicación: {str(e)}")
64         return 1
65
66     def on_login_success(self):
67         try:
68             self.logger.debug("Login exitoso, creando ventana principal...")
69             self.main_controller = MainController()
70             self.main_controller.view.show()
71             self.login_controller.view.close()
72         except Exception as e:
73             self.logger.error(f"Error al crear ventana principal: {str(e)}", exc_info=True)
74             QMessageBox.critical(None, "Error Fatal", f"Error al iniciar la aplicación: {str(e)}")
75
76     def main():
77         if not verificar_plantilla_cv():
78             print("La aplicación continuará, pero la generación de CVs no estará disponible.")
79         # Configurar logging
80         setup_logging()
81         logger = logging.getLogger(__name__)
82
83         application = Application()
84         return application.start()
85
86     if __name__ == '__main__':
87         sys.exit(main())

```

controllers__init__.py

```
1  # app/controllers/__init__.py
2  from .login_controller import LoginController
3  from .main_controller import MainController
4
5  __all__ = ['LoginController', 'MainController']
```

controllers\login_controller.py

```
1  # app/controllers/login_controller.py
2  from PyQt5.QtWidgets import QMessageBox
3  from app.views.login_view import LoginView
4  from app.config import Config
5  from app.models.database import Database
6
7  class LoginController:
8      def __init__(self):
9          self.view = LoginView()
10         self.db = Database()
11         self.setup_connections()
12
13     def setup_connections(self):
14         self.view.login_btn.clicked.connect(self.validar_login)
15         # Añadir eventos Enter para una mejor experiencia de usuario
16         self.view.usuario_input.returnPressed.connect(self.validar_login)
17         self.view.password_input.returnPressed.connect(self.validar_login)
18
19     def validar_login(self):
20         try:
21             usuario = self.view.usuario_input.text().strip()
22             password = self.view.password_input.text().strip()
23
24             if not usuario or not password:
25                 QMessageBox.warning(
26                     self.view,
27                     "Error",
28                     "Por favor complete todos los campos"
29                 )
30             return
31
32             # En modo debug usar credenciales de prueba
33             if Config.DEBUG:
34                 if usuario == Config.DEMO_USER and password == Config.DEMO_PASSWORD:
35                     self.view.loginSuccess.emit()
36                 else:
37                     QMessageBox.warning(
38                         self.view,
39                         "Error",
40                         "Credenciales incorrectas"
41                     )
42             else:
43                 # TODO: Implementar validación con base de datos real
44                 pass
45
46         except Exception as e:
47             QMessageBox.critical(
48                 self.view,
49                 "Error",
50                 f"Error en el inicio de sesión: {str(e)}"
51             )
```

controllers/main_controller.py

```
1  # controllers/main_controller.py
2  import logging
3  from PyQt5.QtWidgets import QMessageBox, QTableWidgetItem, QFileDialog
4  from PyQt5.QtCore import Qt
5  from datetime import datetime
6  from app.views.main_view import MainView
7  from app.models.database import Database
8  from app.utils.cv_generator import CVGenerator
9
10 logger = logging.getLogger(__name__)
11
12 class MainController:
13     def __init__(self):
14         try:
15             logger.debug("Iniciando MainController...")
16             self.view = MainView()
17             self.db = Database()
18             self.experiencias_widgets = []
19             self.setup_connections()
20             self.cargar_datos_iniciales()
21             logger.debug("MainController inicializado correctamente")
22         except Exception as e:
23             logger.error(f"Error al inicializar MainController: {str(e)}")
24             raise
25
26     def setup_connections(self):
27         try:
28             logger.debug("Configurando conexiones...")
29
30             # Conexiones del tab de registro
31             self.view.tab_registro.btn_guardar.clicked.connect(self.guardar_registro)
32             self.view.tab_registro.btn_cancelar.clicked.connect(self.cancelar_registro)
33             self.view.tab_registro.btn_agregar_experiencia.clicked.connect(self.agregar_experien
34
35             # Conexiones del tab de búsqueda
36             self.view.tab_busqueda.btn_buscar.clicked.connect(self.realizar_busqueda)
37             self.view.tab_busqueda.btn_limpiar.clicked.connect(self.limpiar_filtros)
38
39             logger.debug("Conexiones configuradas correctamente")
40         except Exception as e:
41             logger.error(f"Error al configurar conexiones: {str(e)}")
42             raise
43
44     def cargar_datos_iniciales(self):
45         try:
46             self.actualizar_tabla_resultados()
47         except Exception as e:
48             logger.error(f"Error al cargar datos iniciales: {str(e)}")
49             QMessageBox.critical(self.view, "Error", f"Error al cargar datos iniciales: {str(e)}")
50
51     def realizar_busqueda(self):
52         try:
53             filtros = {
54                 'texto': self.view.tab_busqueda.busqueda_input.text().strip(),
55                 'area': self.view.tab_busqueda.area_filtro.currentText(),
56                 'tipo_obra': self.view.tab_busqueda.tipo_obra_filtro.currentText(),
```

```

57         'cargo': self.view.tab_busqueda.cargo_filtro.text().strip(),
58         'ex_eimsa': self.view.tab_busqueda.check_eimsa_filtro.isChecked(),
59         'años_exp': self.view.tab_busqueda.años_exp_filtro.currentText()
60     }
61     resultados = self.db.buscar_registros(filtros)
62     self.actualizar_tabla_resultados(resultados)
63 except Exception as e:
64     logger.error(f"Error en búsqueda: {str(e)}")
65     QMessageBox.critical(self.view, "Error", f"Error al realizar la búsqueda: {str(e)}")
66
67 def actualizar_tabla_resultados(self, resultados=None):
68     try:
69         if resultados is None:
70             resultados = self.db.obtener_todos_registros()
71
72         tabla = self.view.tab_busqueda.tabla_resultados
73         tabla.setRowCount(0)
74
75         for row, datos in enumerate(resultados):
76             tabla.insertRow(row)
77
78             # Convertir Row a diccionario si es necesario
79             if not isinstance(datos, dict):
80                 datos = dict(datos)
81
82             items = [
83                 datos.get('codigo', ''),
84                 f"{datos.get('tipo_documento', '')} {datos.get('numero_documento', '')}",
85                 f"{datos.get('nombres', '')} {datos.get('apellidos', '')}",
86                 datos.get('profesion', ''),
87                 datos.get('area', ''),
88                 datos.get('cargo', ''),
89                 datos.get('años_experiencia', '0'),
90                 datos.get('ciudad_residencia', ''),
91                 'Sí' if datos.get('trabajado_eimsa') else 'No',
92                 datos.get('fecha_registro', '')
93             ]
94
95             for col, valor in enumerate(items):
96                 item = QTableWidgetItem(str(valor))
97                 item.setFlags(item.flags() & ~Qt.ItemIsEditable)
98                 tabla.setItem(row, col, item)
99
100         tabla.resizeColumnsToContents()
101
102 except Exception as e:
103     logger.error(f"Error al actualizar tabla: {str(e)}")
104     QMessageBox.critical(self.view, "Error", f"Error al actualizar tabla: {str(e)}")
105
106 def limpiar_filtros(self):
107     try:
108         tab_busqueda = self.view.tab_busqueda
109         tab_busqueda.busqueda_input.clear()
110         tab_busqueda.area_filtro.setCurrentIndex(0)
111         tab_busqueda.tipo_obra_filtro.setCurrentIndex(0)
112         tab_busqueda.cargo_filtro.clear()
113         tab_busqueda.check_eimsa_filtro.setChecked(False)
114         tab_busqueda.años_exp_filtro.setCurrentIndex(0)

```

```

115         self.actualizar_tabla_resultados()
116     except Exception as e:
117         logger.error(f"Error al limpiar filtros: {str(e)}")
118         QMessageBox.critical(self.view, "Error", f"Error al limpiar filtros: {str(e)}")
119
120     def ver_detalle(self):
121         try:
122             registro_seleccionado = self.obtener_registro_seleccionado()
123             if registro_seleccionado:
124                 # Cambiar a la pestaña de registro en modo lectura
125                 self.view.tab_widget.setCurrentWidget(self.view.tab_registro)
126                 self.cargar_datos_registro(registro_seleccionado, modo_lectura=True)
127         except Exception as e:
128             logger.error(f"Error al ver detalle: {str(e)}")
129             QMessageBox.critical(self.view, "Error", f"Error al ver detalle: {str(e)}")
130
131     def editar_registro(self):
132         try:
133             registro_seleccionado = self.obtener_registro_seleccionado()
134             if registro_seleccionado:
135                 # Cambiar a la pestaña de registro en modo edición
136                 self.view.tab_widget.setCurrentWidget(self.view.tab_registro)
137                 self.cargar_datos_registro(registro_seleccionado, modo_lectura=False)
138         except Exception as e:
139             logger.error(f"Error al editar registro: {str(e)}")
140             QMessageBox.critical(self.view, "Error", f"Error al editar registro: {str(e)}")
141
142     def obtener_registro_seleccionado(self):
143         tabla = self.view.tab_busqueda.tabla_resultados
144         indices_seleccionados = tabla.selectedItems()
145         if indices_seleccionados:
146             fila = indices_seleccionados[0].row()
147             codigo = tabla.item(fila, 0).text()
148             return self.db.obtener_registro(codigo)
149         return None
150
151     def guardar_registro(self):
152         try:
153             datos = self.obtener_datos_formulario()
154             if self.validar_datos(datos):
155                 self.db.guardar_registro(datos)
156                 QMessageBox.information(self.view, "Éxito", "Registro guardado correctamente")
157                 self.view.tab_widget.setCurrentWidget(self.view.tab_busqueda)
158                 self.actualizar_tabla_resultados()
159         except Exception as e:
160             logger.error(f"Error al guardar registro: {str(e)}")
161             QMessageBox.critical(self.view, "Error", f"Error al guardar registro: {str(e)}")
162
163     def cancelar_registro(self):
164         self.view.tab_widget.setCurrentWidget(self.view.tab_busqueda)
165
166     def generar_cv(self):
167         try:
168             registro_seleccionado = self.obtener_registro_seleccionado()
169             if registro_seleccionado:
170                 # Generar nombre del archivo
171                 nombre_archivo = f"CV_{registro_seleccionado.get('nombres', 'sin_nombre')}_{'date'
172                 ruta_salida, _ = QFileDialog.getSaveFileName(

```

```

173         self.view,
174         "Guardar CV",
175         nombre_archivo,
176         "Documentos Word (*.docx)"
177     )
178
179     if ruta_salida:
180         try:
181             generator = CVGenerator()
182             experiencias = self.db.obtener_experiencias(registro_seleccionado['codigo'])
183             if generator.generar(registro_seleccionado, experiencias, ruta_salida):
184                 QMessageBox.information(
185                     self.view,
186                     "Éxito",
187                     f"CV generado correctamente en:\n{ruta_salida}"
188                 )
189                 # Abrir el archivo generado
190                 from os import startfile
191                 startfile(ruta_salida)
192             else:
193                 QMessageBox.warning(
194                     self.view,
195                     "Advertencia",
196                     "No se pudo generar el CV. Verifique que los datos estén completos"
197                 )
198             except Exception as e:
199                 QMessageBox.critical(
200                     self.view,
201                     "Error",
202                     f"Error al generar CV: {str(e)}"
203                 )
204         except Exception as e:
205             logger.error(f"Error al generar CV: {str(e)}")
206             QMessageBox.critical(self.view, "Error", f"Error al generar CV: {str(e)}")
207
208     def obtener_datos_formulario(self):
209         """Obtiene todos los datos del formulario de registro"""
210         try:
211             tab_registro = self.view.tab_registro
212             return {
213                 'codigo': tab_registro.codigo_input.text(),
214                 'tipo_documento': tab_registro.tipo_doc_combo.currentText(),
215                 'numero_documento': tab_registro.num_doc_input.text(),
216                 'nombres': tab_registro.nombres_input.text(),
217                 'apellidos': tab_registro.apellidos_input.text(),
218                 'profesion': tab_registro.profesion_input.text(),
219                 'fecha_nacimiento': tab_registro.fecha_nac_input.date().toString('yyyy-MM-dd'),
220                 'lugar_nacimiento': tab_registro.lugar_nac_input.text(),
221                 'registro_cip': tab_registro.registro_cip_input.text(),
222                 'area': tab_registro.area_combo.currentText(),
223                 'cargo': tab_registro.cargo_input.text(),
224                 'residencia': tab_registro.residencia_input.text(),
225                 'telefono': tab_registro.telefono_input.text(),
226                 'correo': tab_registro.correo_input.text(),
227                 'trabajado_eimsa': tab_registro.check_eimsa.isChecked(),
228                 'experiencias': self.obtener_experiencias_formulario()
229             }
230         except Exception as e:

```

```

231         logger.error(f"Error al obtener datos del formulario: {str(e)}")
232         raise
233
234     def obtener_experiencias_formulario(self):
235         """Obtiene la lista de experiencias del formulario"""
236         experiencias = []
237         for widget in self.experiencias_widgets:
238             experiencias.append(widget.obtener_datos())
239         return experiencias
240
241     def agregar_experiencia(self):
242         try:
243             dialogo = ExperienciaWidget(parent=self.view)
244             if dialogo.exec_(): # Usar exec_ en lugar de exec() para Qt
245                 datos = dialogo.obtener_datos()
246                 self.view.tab_registro.experiencias_layout.addWidget(
247                     ExperienciaWidget(datos=datos, parent=self.view.tab_registro)
248                 )
249         except Exception as e:
250             logger.error(f"Error al agregar experiencia: {str(e)}")
251             QMessageBox.critical(self.view, "Error", f"Error al agregar experiencia: {str(e)}")
252
253     def eliminar_experiencia(self, widget):
254         """Elimina una experiencia del formulario"""
255         try:
256             widget.setParent(None)
257             widget.deleteLater()
258             self.experiencias_widgets.remove(widget)
259         except Exception as e:
260             logger.error(f"Error al eliminar experiencia: {str(e)}")
261             QMessageBox.critical(self.view, "Error", f"Error al eliminar experiencia: {str(e)}")
262
263     def actualizar_experiencia(self, datos):
264         """Actualiza los datos de una experiencia"""
265         try:
266             # Aquí puedes agregar lógica adicional si necesitas
267             # hacer algo cuando se actualiza una experiencia
268             pass
269         except Exception as e:
270             logger.error(f"Error al actualizar experiencia: {str(e)}")
271             QMessageBox.critical(self.view, "Error", f"Error al actualizar experiencia: {str(e)}")
272
273     def cargar_datos_registro(self, datos, modo_lectura=False):
274         """Carga los datos en el formulario de registro"""
275         try:
276             tab_registro = self.view.tab_registro
277
278             # Limpiar experiencias existentes
279             for widget in self.experiencias_widgets:
280                 widget.setParent(None)
281                 widget.deleteLater()
282             self.experiencias_widgets.clear()
283
284             # Cargar datos básicos
285             tab_registro.codigo_input.setText(datos.get('codigo', ''))
286             tab_registro.tipo_doc_combo.setCurrentText(datos.get('tipo_documento', 'DNI'))
287             tab_registro.num_doc_input.setText(datos.get('numero_documento', ''))
288             tab_registro.nombres_input.setText(datos.get('nombres', ''))

```



```

289         tab_registro.apellidos_input.setText(datos.get('apellidos', ''))
290         tab_registro.profesion_input.setText(datos.get('profesion', ''))
291         # ... (continuar con el resto de campos)
292
293         # Cargar experiencias
294         experiencias = self.db.obtener_experiencias(datos['codigo'])
295         for exp in experiencias:
296             widget = ExperienciaWidget(datos=exp, parent=self.view.tab_registro)
297             widget.eliminado.connect(lambda w=widget: self.eliminar_experiencia(w))
298             widget.actualizado.connect(self.actualizar_experiencia)
299             self.experiencias_widgets.append(widget)
300             self.view.tab_registro.experiencias_layout.addWidget(widget)
301
302         # Configurar modo lectura si es necesario
303         if modo_lectura:
304             self.set_modos_lectura(True)
305
306     except Exception as e:
307         logger.error(f"Error al cargar datos en registro: {str(e)}")
308         QMessageBox.critical(self.view, "Error", f"Error al cargar datos: {str(e)}")
309
310     def set_modos_lectura(self, enabled=True):
311         """Configura el formulario en modo lectura"""
312         tab_registro = self.view.tab_registro
313         for widget in tab_registro.findChildren((QLineEdit, QComboBox, QDateEdit)):
314             widget.setReadOnly(enabled)
315             widget.setEnabled(not enabled)
316         tab_registro.btn_guardar.setVisible(not enabled)
317         # Deshabilitar botones de experiencia en modo lectura
318         for exp_widget in self.experiencias_widgets:
319             exp_widget.set_modos_lectura(enabled)
320
321     def __init__(self):
322         try:
323             logger.debug("Iniciando MainController...")
324             self.view = MainView()
325             self.db = Database()
326             self.experiencias_widgets = [] # Lista para mantener referencia a widgets de experi
327             self.setup_connections()
328             self.cargar_datos_iniciales()
329             logger.debug("MainController inicializado correctamente")
330         except Exception as e:
331             logger.error(f"Error al inicializar MainController: {str(e)}", exc_info=True)
332             raise

```

models__init__.py

```
1  from .code_generator import CodeGenerator
2  from .obra_types import TipoObra, CalculadorExperiencia, CategoriaObra, ExperienciaObra
3
4  __all__ = [
5      'CodeGenerator',
6      'TipoObra',
7      'CalculadorExperiencia',
8      'CategoriaObra',
9      'ExperienciaObra'
10 ]
```

models\code_generator.py

```
1  # utils/code_generator.py
2  from datetime import datetime
3  import re
4  from typing import Optional
5
6  class CodeGenerator:
7      def __init__(self):
8          self._counter = 0
9          self._used_codes = set()
10
11      def generate_code(self, tipo: str = 'O') -> str:
12          """
13          Genera un código único con el formato O/A-000001-EI-PE
14          Args:
15              tipo: 'O' para Obra, 'A' para Administración
16          Returns:
17              Código único generado
18          """
19          if tipo not in ['O', 'A']:
20              raise ValueError("Tipo debe ser 'O' para Obra o 'A' para Administración")
21
22          self._counter += 1
23          code = f"{tipo}-{self._counter:06d}-EI-PE"
24
25          while code in self._used_codes:
26              self._counter += 1
27              code = f"{tipo}-{self._counter:06d}-EI-PE"
28
29          self._used_codes.add(code)
30          return code
31
32      def validate_code(self, code: str) -> bool:
33          """
34          Valida el formato del código
35          """
36          pattern = r'^[OA]-\d{6}-EI-PE$'
37          return bool(re.match(pattern, code))
38
39  class ExperienceCalculator:
40      @staticmethod
41      def calculate_years(fecha_inicio: str, fecha_fin: Optional[str] = None) -> float:
42          """
43          Calcula los años de experiencia entre dos fechas
44          """
45          inicio = datetime.strptime(fecha_inicio, '%Y-%m-%d')
46          fin = datetime.strptime(fecha_fin, '%Y-%m-%d') if fecha_fin else datetime.now()
47
48          diff = fin - inicio
49          return round(diff.days / 365.25, 2)
50
51      @staticmethod
52      def calculate_total_experience(experiencias: list) -> float:
53          """
54          Calcula la experiencia total sumando todas las experiencias
55          """
56          total = 0.0
```

```

57         for exp in experiencias:
58             total += ExperienceCalculator.calculate_years(
59                 exp['fecha_inicio'],
60                 exp.get('fecha_fin')
61             )
62         return round(total, 2)
63
64     class MontoValidator:
65         @staticmethod
66         def format_monto(monto: float) -> str:
67             """
68             Formatea un monto en millones
69             """
70             if monto < 1_000_000:
71                 return f"{monto:,.2f}"
72
73             millones = monto / 1_000_000
74             return f"{millones:,.2f}MM"
75
76         @staticmethod
77         def get_range(monto: float) -> str:
78             """
79             Obtiene el rango del monto del proyecto
80             """
81             if monto < 5_000_000:
82                 return "< 5 MM"
83             elif monto < 20_000_000:
84                 return "5 - 20 MM"
85             elif monto < 50_000_000:
86                 return "20 - 50 MM"
87             else:
88                 return "> 50 MM"

```

models\database.py

```
1  # app/models/database.py
2  import sqlite3
3  from pathlib import Path
4  from app.config import Config
5  import os
6  import logging
7  from datetime import datetime
8
9  class MontoRango:
10     MENOR_5MM = 1
11     ENTRE_5_20MM = 2
12     ENTRE_20_50MM = 3
13     ENTRE_50_100MM = 4
14
15     @staticmethod
16     def obtener_rango(monto):
17         """Determina el rango de un monto en millones"""
18         try:
19             monto = float(monto)
20             if monto < 5_000_000:
21                 return MontoRango.MENOR_5MM
22             elif monto < 20_000_000:
23                 return MontoRango.ENTRE_5_20MM
24             elif monto < 50_000_000:
25                 return MontoRango.ENTRE_20_50MM
26             else:
27                 return MontoRango.ENTRE_50_100MM
28         except:
29             return MontoRango.MENOR_5MM
30
31     @staticmethod
32     def obtener_texto_rango(rango):
33         rangos = {
34             MontoRango.MENOR_5MM: "< 5 MM",
35             MontoRango.ENTRE_5_20MM: "5 - 20 MM",
36             MontoRango.ENTRE_20_50MM: "20 - 50 MM",
37             MontoRango.ENTRE_50_100MM: "50 - 100 MM"
38         }
39         return rangos.get(rango, "No especificado")
40
41     logger = logging.getLogger(__name__)
42
43     class Database:
44         _instance = None
45
46         def __new__(cls):
47             if cls._instance is None:
48                 try:
49                     cls._instance = super().__new__(cls)
50                     db_path = Path(Config.DB_PATH)
51                     db_path.parent.mkdir(parents=True, exist_ok=True)
52                     cls._instance.conn = sqlite3.connect(str(db_path))
53                     cls._instance.conn.row_factory = sqlite3.Row
54                     logger.info(f"Conexión a base de datos establecida: {db_path}")
55                     cls._instance.crear_tablas()
56                 except Exception as e:
```

```

57         logger.error(f"Error al crear instancia de base de datos: {str(e)}")
58         raise
59     return cls._instance
60
61     def crear_tablas(self):
62         """Crea todas las tablas necesarias en la base de datos"""
63         try:
64             with self.conn:
65                 # Tabla de datos personales
66                 self.conn.execute('''
67                 CREATE TABLE IF NOT EXISTS datos_personales (
68                     codigo TEXT PRIMARY KEY,
69                     tipo_documento TEXT,
70                     numero_documento TEXT,
71                     nombres TEXT NOT NULL,
72                     apellidos TEXT,
73                     profesion TEXT NOT NULL,
74                     fecha_nacimiento TEXT,
75                     lugar_nacimiento TEXT,
76                     registro_cip TEXT,
77                     area TEXT NOT NULL,
78                     cargo TEXT NOT NULL,
79                     residencia TEXT,
80                     telefono TEXT,
81                     correo TEXT,
82                     trabajo_previo_eimsa BOOLEAN DEFAULT 0,
83                     fecha_registro TIMESTAMP DEFAULT CURRENT_TIMESTAMP
84                 )
85                 ''')
86
87                 # Tabla de imágenes actualizada para almacenar BLOB
88                 self.conn.execute('''
89                 CREATE TABLE IF NOT EXISTS imagenes_cv (
90                     id INTEGER PRIMARY KEY AUTOINCREMENT,
91                     codigo_personal TEXT,
92                     experiencia_id INTEGER,
93                     nombre_archivo TEXT,
94                     contenido_imagen BLOB,
95                     tipo_imagen TEXT,
96                     fecha_subida TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
97                     FOREIGN KEY (codigo_personal) REFERENCES datos_personales(codigo),
98                     FOREIGN KEY (experiencia_id) REFERENCES experiencia_laboral(id)
99                 )
100                ''')
101
102
103                 # Tabla de experiencia laboral
104                 self.conn.execute('''
105                 CREATE TABLE IF NOT EXISTS experiencia_laboral (
106                     id INTEGER PRIMARY KEY AUTOINCREMENT,
107                     codigo_personal TEXT NOT NULL,
108                     empresa TEXT NOT NULL,
109                     obra TEXT NOT NULL,
110                     detalle_obra TEXT,
111                     tipo_obra TEXT NOT NULL,
112                     monto_proyecto TEXT,
113                     cargo TEXT NOT NULL,
114                     fecha_inicio TEXT NOT NULL,

```

```

115         fecha_fin TEXT,
116         propietario TEXT,
117         funciones TEXT,
118         FOREIGN KEY (codigo_personal) REFERENCES datos_personales(codigo) ON DELETE
119     )
120     '''
121
122     # Tabla para documentos
123     self.conn.execute('''
124     CREATE TABLE IF NOT EXISTS documentos (
125         id INTEGER PRIMARY KEY AUTOINCREMENT,
126         codigo_personal TEXT,
127         experiencia_id INTEGER NULL,
128         tipo_documento TEXT,
129         nombre_archivo TEXT,
130         ruta_archivo TEXT,
131         descripcion TEXT,
132         fecha_subida TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
133         FOREIGN KEY (codigo_personal) REFERENCES datos_personales(codigo),
134         FOREIGN KEY (experiencia_id) REFERENCES experiencia_laboral(id)
135     )
136     '''
137
138
139     # Tabla de tipos de obra actualizada
140     self.conn.execute('''
141     CREATE TABLE IF NOT EXISTS tipos_obra (
142         id TEXT PRIMARY KEY,
143         categoria TEXT NOT NULL,
144         nombre TEXT NOT NULL,
145         descripcion TEXT
146     )
147     '''
148
149     # Insertar tipos de obra predefinidos
150     tipos_obra = [
151         # Industria
152         ('PCAL', 'Industria', 'Planta de Cemento y Cal', 'Plantas de procesamiento'),
153         ('NIND', 'Industria', 'Naves Industriales', 'Construcción de naves'),
154         ('HIDRO', 'Industria', 'Hidroeléctrica', 'Centrales hidroeléctricas'),
155         ('TERMO', 'Industria', 'Central Termoeléctrica', 'Centrales térmicas'),
156         ('PELEC', 'Industria', 'Planta Eléctrica', 'Plantas eléctricas'),
157         ('PUENT', 'Industria', 'Puentes', 'Construcción de puentes'),
158         ('PETRO', 'Industria', 'Planta Petróleo', 'Plantas petroleras'),
159         ('CIVIL', 'Industria', 'Obras Civiles', 'Obras civiles'),
160
161         # Minería
162         ('PGAS', 'Minería', 'Planta de Gas', 'Plantas de gas'),
163         ('ESTRU', 'Minería', 'Construcción Estructuras', 'Estructuras mineras'),
164         ('MPESA', 'Minería', 'Montaje de Estructura Pesada', 'Montajes pesados'),
165         ('CHANC', 'Minería', 'Chancadora/Fajas (seca)', 'Sistemas de chancado'),
166         ('MOLIN', 'Minería', 'Molinos/Celda de Flotación', 'Sistemas de molienda'),
167         ('TUNEL', 'Minería', 'Túneles', 'Construcción de túneles'),
168         ('DESMO', 'Minería', 'Desmontaje', 'Trabajos de desmontaje'),
169
170         # Servicios
171         ('ELEC', 'Servicios', 'Electricidad & Instrumentación', 'Sistemas eléctricos'),
172         ('TUBE', 'Servicios', 'Tubería Agua/Relaves', 'Sistemas de tubería'),

```

```

173         ('DESAL', 'Servicios', 'Planta Desalinizadora', 'Desalinización'),
174         ('TRUCK', 'Servicios', 'Truckshop - Almacenes', 'Talleres y almacenes')
175     ]
176
177     # Primero, limpiar la tabla de tipos de obra
178     self.conn.execute('DELETE FROM tipos_obra')
179
180     # Insertar los nuevos tipos de obra
181     self.conn.executemany('''
182     INSERT INTO tipos_obra (id, categoria, nombre, descripcion)
183     VALUES (?, ?, ?, ?)
184     ''', tipos_obra)
185
186     logger.info("Todas las tablas creadas/verificadas exitosamente")
187
188 except Exception as e:
189     logger.error(f"Error al crear tablas: {str(e)}")
190     raise
191
192 def inicializar_tipos_obra(self):
193     try:
194         tipos_obra = [
195             ('IND_CEM', 'Planta de Cemento y Cal', 'Industria de cemento y cal'),
196             ('NAV_IND', 'Naves Industriales', 'Construcción de naves industriales'),
197             ('HIDRO', 'Hidroeléctrica', 'Centrales hidroeléctricas'),
198             ('PUENTE', 'Puentes', 'Construcción de puentes'),
199             ('PET', 'Plantas de Petróleo', 'Infraestructura petrolera'),
200             ('TUB', 'Tuberías', 'Sistemas de tuberías'),
201             ('GAS', 'Planta de Gas', 'Plantas procesadoras de gas'),
202             ('MIN', 'Minería', 'Proyectos mineros'),
203             ('TUNEL', 'Túneles', 'Construcción de túneles'),
204             ('DESM', 'Desmontaje', 'Trabajos de desmontaje'),
205             ('CIV', 'Obras Civiles', 'Construcciones civiles'),
206             ('ELEC', 'Electricidad e Instrumentación', 'Sistemas eléctricos'),
207             ('DESAL', 'Planta Desalinizadora', 'Plantas de desalinización'),
208             ('TRUCK', 'Truck Shop', 'Talleres y almacenes')
209         ]
210
211         with self.conn:
212             for tipo in tipos_obra:
213                 self.conn.execute('''
214                     INSERT OR IGNORE INTO tipos_obra (id, nombre, descripcion)
215                     VALUES (?, ?, ?)
216                     ''', tipo)
217             logger.debug("Tipos de obra inicializados correctamente")
218     except Exception as e:
219         logger.error(f"Error al inicializar tipos de obra: {str(e)}")
220         raise
221
222 def guardar_registro(self, datos):
223     try:
224         with self.conn:
225             cursor = self.conn.cursor()
226             logger.debug(f"Iniciando guardado de registro para código: {datos.get('codigo',
227
228             # Si hay monto_proyecto, convertir a decimal
229             if 'monto_proyecto' in datos:
230                 monto_str = datos['monto_proyecto']

```



```

231         # Limpiar el string de moneda
232         monto_num = ''.join(filter(str.isdigit, monto_str))
233         if monto_num:
234             datos['monto_decimal'] = float(monto_num)
235
236     # Extraer las experiencias
237     experiencias = datos.pop('experiencias', [])
238
239     if datos['codigo']: # Actualización
240         cursor.execute('''
241             UPDATE datos_personales SET
242                 nombres=?, profesion=?, fecha_nacimiento=?,
243                 lugar_nacimiento=?, registro_cip=?, area=?,
244                 cargo=?, residencia=?, telefono=?, correo=?
245             WHERE codigo=?
246             ''', (
247                 datos['nombres'], datos['profesion'], datos['fecha_nacimiento'],
248                 datos['lugar_nacimiento'], datos['registro_cip'], datos['area'],
249                 datos['cargo'], datos['residencia'], datos['telefono'],
250                 datos['correo'], datos['codigo']
251             ))
252         logger.debug(f"Actualizado registro existente: {datos['codigo']}")
253     else: # Nuevo registro
254         datos['codigo'] = self.generar_codigo(datos['area'])
255         cursor.execute('''
256             INSERT INTO datos_personales (
257                 codigo, nombres, profesion, fecha_nacimiento,
258                 lugar_nacimiento, registro_cip, area, cargo,
259                 residencia, telefono, correo, fecha_registro
260             ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, CURRENT_TIMESTAMP)
261             ''', (
262                 datos['codigo'], datos['nombres'], datos['profesion'],
263                 datos['fecha_nacimiento'], datos['lugar_nacimiento'],
264                 datos['registro_cip'], datos['area'], datos['cargo'],
265                 datos['residencia'], datos['telefono'], datos['correo']
266             ))
267         logger.debug(f"Creado nuevo registro: {datos['codigo']}")
268
269     # Manejar las experiencias
270     experiencias_actuales = set()
271     logger.debug("Iniciando procesamiento de experiencias")
272
273     for exp in experiencias:
274         if 'id' in exp: # Experiencia existente
275             experiencias_actuales.add(exp['id'])
276             cursor.execute('''
277                 UPDATE experiencia_laboral SET
278                     empresa=?, obra=?, tipo_obra=?,
279                     detalle_obra=?, monto_proyecto=?, cargo=?,
280                     fecha_inicio=?, fecha_fin=?, propietario=?, funciones=?
281                 WHERE id=? AND codigo_personal=?
282                 ''', (
283                     exp['empresa'], exp['obra'], exp['tipo_obra'],
284                     exp['detalle_obra'], exp['monto_proyecto'], exp['cargo'],
285                     exp['fecha_inicio'], exp['fecha_fin'], exp['propietario'],
286                     ';' + exp['funciones'] if isinstance(exp['funciones'], list) else
287                     exp['id'], datos['codigo']
288                 ))

```

```

289         exp_id = exp['id']
290         logger.debug(f"Actualizada experiencia ID: {exp_id}")
291     else: # Nueva experiencia
292         cursor.execute('''
293             INSERT INTO experiencia_laboral (
294                 codigo_personal, empresa, obra, tipo_obra,
295                 detalle_obra, monto_proyecto, cargo,
296                 fecha_inicio, fecha_fin, propietario, funciones
297             ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
298         ''', (
299             datos['codigo'], exp['empresa'], exp['obra'],
300             exp['tipo_obra'], exp['detalle_obra'], exp['monto_proyecto'],
301             exp['cargo'], exp['fecha_inicio'], exp['fecha_fin'],
302             exp['propietario'], ';'.join(exp['funciones']) if isinstance(exp['fu
303         ))
304         exp_id = cursor.lastrowid
305         experiencias_actuales.add(exp_id)
306         logger.debug(f"Creada nueva experiencia ID: {exp_id}")
307
308     # Manejar documentos/imágenes
309     if 'documentos' in exp and exp['documentos']:
310         logger.debug(f"Procesando documentos para experiencia ID: {exp_id}")
311         for doc in exp['documentos']:
312             with open(doc['ruta'], 'rb') as img_file:
313                 contenido_imagen = img_file.read()
314
315             cursor.execute('''
316                 INSERT INTO imagenes_cv (
317                     codigo_personal, experiencia_id, nombre_archivo,
318                     contenido_imagen, tipo_imagen
319                 ) VALUES (?, ?, ?, ?, ?)
320             ''', (
321                 datos['codigo'],
322                 exp_id,
323                 doc['nombre'],
324                 contenido_imagen,
325                 'experiencia'
326             ))
327             logger.debug(f"Guardada imagen: {doc['nombre']}")
328
329     # Eliminar experiencias que ya no existen
330     if experiencias_actuales:
331         cursor.execute('''
332             DELETE FROM experiencia_laboral
333             WHERE codigo_personal = ? AND id NOT IN ({})
334             '''.format(','.join('? ' * len(experiencias_actuales)),
335                 [datos['codigo']] + list(experiencias_actuales))
336         logger.debug("Eliminadas experiencias obsoletas")
337
338     logger.info(f"Registro guardado exitosamente: {datos['codigo']}")
339     return datos['codigo']
340
341 except Exception as e:
342     logger.error(f"Error al guardar registro: {str(e)}")
343     raise Exception(f"Error al guardar en la base de datos: {str(e)}")
344
345 def obtener_registro(self, codigo):
346     try:

```

```

347         cursor = self.conn.cursor()
348         logger.debug(f"Buscando registro con código: {codigo}")
349
350         # Obtener datos personales
351         cursor.execute('SELECT * FROM datos_personales WHERE codigo = ?', (codigo,))
352         datos_personales = cursor.fetchone()
353
354         if datos_personales:
355             datos = dict(datos_personales)
356             logger.debug(f"Datos personales encontrados para código: {codigo}")
357
358             # Obtener experiencias
359             cursor.execute('''
360                 SELECT * FROM experiencia_laboral
361                 WHERE codigo_personal = ?
362                 ORDER BY fecha_inicio DESC
363             ''', (codigo,))
364
365             experiencias = []
366             for exp in cursor.fetchall():
367                 exp_dict = dict(exp)
368                 logger.debug(f"Procesando experiencia ID: {exp_dict['id']}")
369
370                 # Obtener imágenes para esta experiencia
371                 cursor.execute('''
372                     SELECT id, nombre_archivo, contenido_imagen, tipo_imagen
373                     FROM imagenes_cv
374                     WHERE codigo_personal = ? AND experiencia_id = ?
375                 ''', (codigo, exp_dict['id']))
376
377                 documentos = []
378                 for img in cursor.fetchall():
379                     # Guardar temporalmente la imagen
380                     temp_path = os.path.join(Config.TEMP_DIR, img['nombre_archivo'])
381                     with open(temp_path, 'wb') as f:
382                         f.write(img['contenido_imagen'])
383
384                     documentos.append({
385                         'nombre': img['nombre_archivo'],
386                         'ruta': temp_path,
387                         'tipo': img['tipo_imagen']
388                     })
389                 logger.debug(f"Imagen procesada: {img['nombre_archivo']}")
390                 exp_dict['documentos'] = documentos
391                 experiencias.append(exp_dict)
392
393             datos['experiencias'] = experiencias
394             logger.info(f"Registro obtenido exitosamente: {codigo}")
395             return datos
396
397         logger.warning(f"No se encontró registro para código: {codigo}")
398         return None
399
400     except Exception as e:
401         logger.error(f"Error al obtener registro {codigo}: {str(e)}")
402         raise Exception(f"Error al obtener registro: {str(e)}")
403
404     def generar_codigo(self, area):

```

```

405         try:
406             cursor = self.conn.cursor()
407             cursor.execute('''
408                 SELECT codigo FROM datos_personales
409                 WHERE codigo LIKE 'O-%'
410                 ORDER BY codigo DESC LIMIT 1
411             ''')
412             result = cursor.fetchone()
413
414             if result:
415                 ultimo_num = int(result[0].split('-')[1])
416                 nuevo_num = ultimo_num + 1
417             else:
418                 nuevo_num = 1
419
420             nuevo_codigo = f"O-{nuevo_num:06d}-EI-PE"
421             logger.debug(f"Generado nuevo código: {nuevo_codigo}")
422             return nuevo_codigo
423         except Exception as e:
424             logger.error(f"Error al generar código: {str(e)}")
425             raise
426
427     def obtener_todos_registros(self):
428         """Obtiene todos los registros de la base de datos"""
429         try:
430             cursor = self.conn.cursor()
431             cursor.execute('''
432                 SELECT dp.*, COUNT(el.id) as total_experiencias
433                 FROM datos_personales dp
434                 LEFT JOIN experiencia_laboral el ON dp.codigo = el.codigo_personal
435                 GROUP BY dp.codigo
436                 ORDER BY dp.fecha_registro DESC
437             ''')
438             logger.debug("Obteniendo todos los registros")
439             return [dict(row) for row in cursor.fetchall()]
440         except Exception as e:
441             logger.error(f"Error al obtener todos los registros: {str(e)}")
442             raise
443
444     def buscar_registros(self, filtros):
445         """
446         Busca registros según los filtros proporcionados
447         filtros: diccionario con los criterios de búsqueda
448         """
449         try:
450             logger.debug(f"Iniciando búsqueda con filtros: {filtros}")
451             query = """
452                 SELECT DISTINCT dp.*
453                 FROM datos_personales dp
454                 LEFT JOIN experiencia_laboral el ON dp.codigo = el.codigo_personal
455                 WHERE 1=1
456             """
457             params = []
458
459             if filtros.get('nombres'):
460                 query += " AND dp.nombres LIKE ?"
461                 params.append(f"%{filtros['nombres']}%")
462

```

```

463         if filtros.get('area') and filtros['area'] != 'Todas las áreas':
464             query += " AND dp.area = ?"
465             params.append(filtros['area'])
466
467         if filtros.get('cargo'):
468             query += " AND dp.cargo LIKE ?"
469             params.append(f"%{filtros['cargo']}%")
470
471         if filtros.get('residencia'):
472             query += " AND dp.residencia LIKE ?"
473             params.append(f"%{filtros['residencia']}%")
474
475         cursor = self.conn.cursor()
476         cursor.execute(query, params)
477         resultados = cursor.fetchall()
478         logger.debug(f"Búsqueda completada. Resultados encontrados: {len(resultados)}")
479         return resultados
480     except Exception as e:
481         logger.error(f"Error en búsqueda de registros: {str(e)}")
482         raise
483
484     def eliminar_registro(self, codigo):
485         """Elimina un registro y todos sus datos relacionados"""
486         try:
487             logger.debug(f"Iniciando eliminación de registro: {codigo}")
488             with self.conn:
489                 # Eliminar imágenes primero
490                 self.conn.execute("""
491                     DELETE FROM imagenes_cv
492                     WHERE codigo_personal = ?
493                     """, (codigo,))
494
495                 # Eliminar experiencias
496                 self.conn.execute("""
497                     DELETE FROM experiencia_laboral
498                     WHERE codigo_personal = ?
499                     """, (codigo,))
500
501                 # Eliminar datos personales
502                 self.conn.execute("""
503                     DELETE FROM datos_personales
504                     WHERE codigo = ?
505                     """, (codigo,))
506
507             logger.info(f"Registro eliminado exitosamente: {codigo}")
508             return True
509         except Exception as e:
510             logger.error(f"Error al eliminar registro {codigo}: {str(e)}")
511             raise
512
513     def verificar_duplicado(self, nombres, registro_cip):
514         """Verifica si ya existe un registro con el mismo nombre o CIP"""
515         try:
516             cursor = self.conn.cursor()
517             cursor.execute("""
518                 SELECT codigo FROM datos_personales
519                 WHERE nombres = ? OR registro_cip = ?
520                 """, (nombres, registro_cip))

```

```

521
522         resultado = cursor.fetchone()
523         if resultado:
524             logger.warning(f"Encontrado registro duplicado: {resultado['codigo']}")
525         return resultado is not None
526     except Exception as e:
527         logger.error(f"Error al verificar duplicados: {str(e)}")
528         raise
529
530 def obtener_experiencias_por_tipo(self, tipo_obra):
531     """Obtiene todas las experiencias de un tipo de obra específico"""
532     try:
533         cursor = self.conn.cursor()
534         cursor.execute("""
535             SELECT dp.nombres, el.*
536             FROM experiencia_laboral el
537             JOIN datos_personales dp ON el.codigo_personal = dp.codigo
538             WHERE el.tipo_obra = ?
539             ORDER BY el.fecha_inicio DESC
540             """, (tipo_obra,))
541
542         logger.debug(f"Obteniendo experiencias para tipo de obra: {tipo_obra}")
543         return cursor.fetchall()
544     except Exception as e:
545         logger.error(f"Error al obtener experiencias por tipo: {str(e)}")
546         raise
547
548 def actualizar_imagenes(self, experiencia_id, imagenes):
549     """Actualiza las imágenes de una experiencia"""
550     try:
551         logger.debug(f"Actualizando imágenes para experiencia ID: {experiencia_id}")
552         with self.conn:
553             # Eliminar imágenes existentes
554             self.conn.execute("""
555                 DELETE FROM imagenes_cv
556                 WHERE experiencia_id = ?
557                 """, (experiencia_id,))
558
559             # Insertar nuevas imágenes
560             for img in imagenes:
561                 with open(img['ruta'], 'rb') as f:
562                     contenido = f.read()
563
564                 self.conn.execute("""
565                     INSERT INTO imagenes_cv (
566                         codigo_personal, experiencia_id,
567                         nombre_archivo, contenido_imagen, tipo_imagen
568                     ) VALUES (?, ?, ?, ?, ?)
569                 """, (
570                     img['codigo_personal'],
571                     experiencia_id,
572                     img['nombre'],
573                     contenido,
574                     img['tipo']
575                 ))
576
577         logger.info(f"Imágenes actualizadas para experiencia ID: {experiencia_id}")
578     except Exception as e:

```

```

579         logger.error(f"Error al actualizar imágenes: {str(e)}")
580         raise
581
582     def exportar_datos(self, ruta_archivo):
583         """Exporta todos los datos a un archivo SQL"""
584         try:
585             logger.debug(f"Iniciando exportación de datos a: {ruta_archivo}")
586             with open(ruta_archivo, 'w') as f:
587                 for line in self.conn.iterdump():
588                     f.write(f'{line}\n')
589             logger.info(f"Datos exportados exitosamente a: {ruta_archivo}")
590             return True
591         except Exception as e:
592             logger.error(f"Error al exportar datos: {str(e)}")
593             raise
594
595     def importar_datos(self, ruta_archivo):
596         """Importa datos desde un archivo SQL"""
597         try:
598             logger.debug(f"Iniciando importación de datos desde: {ruta_archivo}")
599             with open(ruta_archivo, 'r') as f:
600                 sql = f.read()
601             self.conn.executescript(sql)
602             logger.info(f"Datos importados exitosamente desde: {ruta_archivo}")
603             return True
604         except Exception as e:
605             logger.error(f"Error al importar datos: {str(e)}")
606             raise
607
608     def backup_database(self):
609         """Realiza una copia de seguridad de la base de datos"""
610         try:
611             fecha = datetime.now().strftime('%Y%m%d_%H%M%S')
612             backup_path = os.path.join(Config.DATA_DIR, f'backup_{fecha}.db')
613
614             # Crear una copia de la base de datos
615             with open(backup_path, 'wb') as backup_file:
616                 backup_file.write(self.conn.execute('vacuum into ?', (backup_path,)))
617
618             logger.info(f"Backup creado exitosamente: {backup_path}")
619             return backup_path
620         except Exception as e:
621             logger.error(f"Error al crear backup: {str(e)}")
622             raise
623
624     def obtener_experiencias(self, codigo_personal):
625         """Obtiene todas las experiencias de un personal específico"""
626         try:
627             cursor = self.conn.cursor()
628             cursor.execute('''
629             SELECT * FROM experiencia_laboral
630             WHERE codigo_personal = ?
631             ORDER BY fecha_inicio DESC
632             ''', (codigo_personal,))
633
634             return [dict(row) for row in cursor.fetchall()]
635         except Exception as e:
636             logger.error(f"Error al obtener experiencias: {str(e)}")

```


models\experience_filter.py

```
1  # models/experience_filter.py
2  from typing import List, Dict, Any
3  from datetime import datetime
4  from dataclasses import dataclass
5
6  @dataclass
7  class FilterCriteria:
8      area: str = None
9      cargo: str = None
10     tipo_obra: str = None
11     rango_monto: str = None
12     ex_eimsa: bool = False
13     años_experiencia: int = None
14     residencia: str = None
15
16 class ExperienceFilter:
17     def __init__(self, database_connection):
18         self.db = database_connection
19
20     def apply_filters(self, criteria: FilterCriteria) -> List[Dict[str, Any]]:
21         """
22         Aplica los filtros especificados a la búsqueda de experiencias
23         """
24         query = """
25         SELECT DISTINCT
26             p.*,
27             e.tipo_obra,
28             e.monto_proyecto,
29             e.cargo as ultimo_cargo
30         FROM
31             datos_personales p
32         LEFT JOIN experiencia_laboral e ON
33             p.codigo = e.codigo_personal
34         WHERE 1=1
35         """
36         params = []
37
38         if criteria.area:
39             query += " AND p.area = ?"
40             params.append(criteria.area)
41
42         if criteria.cargo:
43             query += " AND (p.cargo_referencia LIKE ? OR e.cargo LIKE ?)"
44             params.extend([f"%{criteria.cargo}%", f"%{criteria.cargo}%"])
45
46         if criteria.tipo_obra:
47             query += " AND e.tipo_obra = ?"
48             params.append(criteria.tipo_obra)
49
50         if criteria.rango_monto:
51             ranges = {
52                 "< 5 MM": (0, 5_000_000),
53                 "5 - 20 MM": (5_000_000, 20_000_000),
54                 "20 - 50 MM": (20_000_000, 50_000_000),
55                 "> 50 MM": (50_000_000, float('inf'))
56             }
```

```

57         min_val, max_val = ranges[criteria.rango_monto]
58         query += " AND e.monto_proyecto >= ? AND e.monto_proyecto < ?"
59         params.extend([min_val, max_val])
60
61     if criteria.ex_eimsa:
62         query += " AND p.trabajo_previo_eimsa = 1"
63
64     if criteria.residencia:
65         query += " AND p.lugar_residencia LIKE ?"
66         params.append(f"%{criteria.residencia}%")
67
68     cursor = self.db.execute(query, params)
69     results = cursor.fetchall()
70
71     # Filtrar por años de experiencia si se especifica
72     if criteria.años_experiencia:
73         filtered_results = []
74         for result in results:
75             total_exp = self._calculate_total_experience(result['codigo'])
76             if total_exp >= criteria.años_experiencia:
77                 result['años_experiencia'] = total_exp
78                 filtered_results.append(result)
79         return filtered_results
80
81     return results
82
83 def _calculate_total_experience(self, codigo_personal: str) -> float:
84     """
85     Calcula la experiencia total para un profesional
86     """
87     query = """
88     SELECT
89         fecha_inicio,
90         fecha_fin
91     FROM
92         experiencia_laboral
93     WHERE
94         codigo_personal = ?
95     """
96     cursor = self.db.execute(query, (codigo_personal,))
97     experiences = cursor.fetchall()
98
99     total = 0.0
100    for exp in experiences:
101        inicio = datetime.strptime(exp['fecha_inicio'], '%Y-%m-%d')
102        fin = datetime.strptime(exp['fecha_fin'], '%Y-%m-%d') if exp['fecha_fin'] else datet
103        total += (fin - inicio).days / 365.25
104
105    return round(total, 2)
106
107 def get_experience_summary(self, codigo_personal: str) -> Dict[str, Any]:
108     """
109     Obtiene un resumen de la experiencia por tipo de obra
110     """
111     query = """
112     SELECT
113         tipo_obra,
114         COUNT(*) as total_proyectos,

```

```

115         SUM(
116             (julianday(COALESCE(fecha_fin, date('now')) -
117                 julianday(fecha_inicio)) / 365.25
118             ) as años_experiencia,
119         MAX(monto_proyecto) as mayor_monto
120     FROM
121         experiencia_laboral
122     WHERE
123         codigo_personal = ?
124     GROUP BY
125         tipo_obra
126     """
127     cursor = self.db.execute(query, (codigo_personal,))
128     return {row['tipo_obra']: dict(row) for row in cursor.fetchall()}

```

models\experience_manager.py

```
1  # models/experience_manager.py
2  from typing import List, Dict, Any, Optional
3  from datetime import datetime
4  from dataclasses import dataclass
5  import sqlite3
6
7  @dataclass
8  class ExperienciaLaboral:
9      empresa: str
10     obra: str
11     tipo_obra: str
12     cargo: str
13     fecha_inicio: str
14     fecha_fin: Optional[str]
15     propietario: str
16     detalle_obra: str
17     monto_proyecto: float
18     codigo_personal: str
19     id: Optional[int] = None
20
21  class ExperienceManager:
22     def __init__(self, database_connection):
23         self.db = database_connection
24
25     def add_experience(self, experiencia: ExperienciaLaboral) -> int:
26         """
27         Agrega una nueva experiencia laboral
28         Returns:
29             ID de la experiencia creada
30         """
31         query = """
32         INSERT INTO experiencia_laboral (
33             codigo_personal, empresa, obra, tipo_obra,
34             cargo, fecha_inicio, fecha_fin, propietario,
35             detalle_obra, monto_proyecto
36         ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
37         """
38         cursor = self.db.execute(query, (
39             experiencia.codigo_personal,
40             experiencia.empresa,
41             experiencia.obra,
42             experiencia.tipo_obra,
43             experiencia.cargo,
44             experiencia.fecha_inicio,
45             experiencia.fecha_fin,
46             experiencia.propietario,
47             experiencia.detalle_obra,
48             experiencia.monto_proyecto
49         ))
50         self.db.commit()
51         return cursor.lastrowid
52
53     def update_experience(self, experiencia: ExperienciaLaboral) -> bool:
54         """
55         Actualiza una experiencia existente
56         """
```

```

57         if not experiencia.id:
58             raise ValueError("ID de experiencia no proporcionado")
59
60         query = """
61         UPDATE experiencia_laboral SET
62             empresa = ?,
63             obra = ?,
64             tipo_obra = ?,
65             cargo = ?,
66             fecha_inicio = ?,
67             fecha_fin = ?,
68             propietario = ?,
69             detalle_obra = ?,
70             monto_proyecto = ?
71         WHERE id = ? AND codigo_personal = ?
72         """
73         cursor = self.db.execute(query, (
74             experiencia.empresa,
75             experiencia.obra,
76             experiencia.tipo_obra,
77             experiencia.cargo,
78             experiencia.fecha_inicio,
79             experiencia.fecha_fin,
80             experiencia.propietario,
81             experiencia.detalle_obra,
82             experiencia.monto_proyecto,
83             experiencia.id,
84             experiencia.codigo_personal
85         ))
86         self.db.commit()
87         return cursor.rowcount > 0
88
89     def get_experiences(self, codigo_personal: str) -> List[Dict[str, Any]]:
90         """
91         Obtiene todas las experiencias de un profesional
92         """
93         query = """
94         SELECT
95             *,
96             (julianday(COALESCE(fecha_fin, date('now')))) -
97             julianday(fecha_inicio)) / 365.25 as duracion_años
98         FROM
99             experiencia_laboral
100        WHERE
101            codigo_personal = ?
102        ORDER BY
103            fecha_inicio DESC
104        """
105        cursor = self.db.execute(query, (codigo_personal,))
106        return cursor.fetchall()
107
108     def delete_experience(self, id: int, codigo_personal: str) -> bool:
109         """
110         Elimina una experiencia laboral
111         """
112         query = """
113         DELETE FROM experiencia_laboral
114         WHERE id = ? AND codigo_personal = ?

```

```

115         """
116         cursor = self.db.execute(query, (id, codigo_personal))
117         self.db.commit()
118         return cursor.rowcount > 0
119
120     def get_experience_by_type(self, tipo_obra: str) -> List[Dict[str, Any]]:
121         """
122         Obtiene todas las experiencias de un tipo específico
123         """
124         query = """
125         SELECT
126             e.*,
127             p.nombres,
128             p.profesion,
129             (julianday(COALESCE(e.fecha_fin, date('now')))) -
130             julianday(e.fecha_inicio)) / 365.25 as duracion_años
131         FROM
132             experiencia_laboral e
133         JOIN
134             datos_personales p ON e.codigo_personal = p.codigo
135         WHERE
136             e.tipo_obra = ?
137         ORDER BY
138             e.fecha_inicio DESC
139         """
140         cursor = self.db.execute(query, (tipo_obra,))
141         return cursor.fetchall()

```

models\experiencia.py

```
1  # app/models/experiencia.py
2  from dataclasses import dataclass
3  from datetime import datetime
4  from typing import List, Optional
5  from .database import Database
6
7  @dataclass
8  class Experiencia:
9      codigo_personal: str
10     empresa: str
11     obra: str
12     tipo_obra: str
13     cargo: str
14     fecha_inicio: str
15     fecha_fin: Optional[str]
16     detalle_obra: Optional[str] = None
17     monto_proyecto: Optional[str] = None
18     propietario: Optional[str] = None
19     funciones: Optional[str] = None
20     id: Optional[int] = None
21
22     def guardar(self) -> bool:
23         db = Database()
24         try:
25             with db.conn:
26                 cursor = db.conn.cursor()
27                 cursor.execute('''
28                     INSERT INTO experiencia_laboral (
29                         codigo_personal, empresa, obra, detalle_obra,
30                         tipo_obra, monto_proyecto, cargo, fecha_inicio,
31                         fecha_fin, propietario, funciones
32                     ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
33                 ''', (
34                     self.codigo_personal, self.empresa, self.obra,
35                     self.detalle_obra, self.tipo_obra, self.monto_proyecto,
36                     self.cargo, self.fecha_inicio, self.fecha_fin,
37                     self.propietario, self.funciones
38                 ))
39                 self.id = cursor.lastrowid
40                 return True
41         except Exception as e:
42             print(f"Error al guardar experiencia: {e}")
43             return False
44
45     def actualizar(self) -> bool:
46         if not self.id:
47             return False
48
49         db = Database()
50         try:
51             with db.conn:
52                 db.conn.execute('''
53                     UPDATE experiencia_laboral
54                     SET empresa=?, obra=?, detalle_obra=?, tipo_obra=?,
55                     monto_proyecto=?, cargo=?, fecha_inicio=?,
56                     fecha_fin=?, propietario=?, funciones=?
```

```

57         WHERE id=?
58     ''' , (
59         self.empresa, self.obra, self.detalle_obra,
60         self.tipo_obra, self.monto_proyecto, self.cargo,
61         self.fecha_inicio, self.fecha_fin, self.propietario,
62         self.funciones, self.id
63     ))
64     return True
65 except Exception as e:
66     print(f"Error al actualizar experiencia: {e}")
67     return False
68
69 @staticmethod
70 def obtener_por_personal(codigo_personal: str) -> List['Experiencia']:
71     db = Database()
72     cursor = db.conn.cursor()
73
74     cursor.execute('''
75         SELECT * FROM experiencia_laboral
76         WHERE codigo_personal = ?
77         ORDER BY fecha_inicio DESC
78     ''', (codigo_personal,))
79
80     resultados = cursor.fetchall()
81     return [Experiencia(**dict(row)) for row in resultados]
82
83 def calcular_duracion(self) -> int:
84     """Calcula la duración en meses de la experiencia"""
85     inicio = datetime.strptime(self.fecha_inicio, '%Y-%m-%d')
86     fin = datetime.strptime(self.fecha_fin, '%Y-%m-%d') if self.fecha_fin else datetime.now()
87
88     return ((fin.year - inicio.year) * 12 + (fin.month - inicio.month))

```


models\obra_types.py

```
1  from enum import Enum
2  from typing import List, Dict, Optional
3  from dataclasses import dataclass
4
5  @dataclass
6  class ExperienciaObra:
7      """Clase para almacenar la experiencia en un tipo de obra"""
8      años: float
9      meses: int
10     proyectos: List[str]
11
12     class CategoriaObra(Enum):
13         INDUSTRIA = "Industria"
14         MINERIA = "Minería"
15         ENERGIA = "Energía"
16         INFRAESTRUCTURA = "Infraestructura"
17         SERVICIOS = "Servicios"
18
19     class TipoObra:
20         def __init__(self):
21             self.tipos = {
22                 # Industria
23                 "CEMENTO_CAL": {
24                     "nombre": "Planta de Cemento y Cal",
25                     "categoria": CategoriaObra.INDUSTRIA
26                 },
27                 "NAVES_INDUSTRIALES": {
28                     "nombre": "Naves Industriales",
29                     "categoria": CategoriaObra.INDUSTRIA
30                 },
31
32                 # Minería
33                 "CHANCADORAS": {
34                     "nombre": "Chancadoras/Fajas",
35                     "categoria": CategoriaObra.MINERIA
36                 },
37                 "MOLINOS": {
38                     "nombre": "Molinos/Celdas de Flotación",
39                     "categoria": CategoriaObra.MINERIA
40                 },
41
42                 # Energía
43                 "HIDROELECTRICA": {
44                     "nombre": "Hidroeléctrica",
45                     "categoria": CategoriaObra.ENERGIA
46                 },
47                 "TERMOELECTRICA": {
48                     "nombre": "Central Termoeléctrica",
49                     "categoria": CategoriaObra.ENERGIA
50                 },
51
52                 # Infraestructura
53                 "TUBERIAS": {
54                     "nombre": "Tubería Agua/Relaves",
55                     "categoria": CategoriaObra.INFRAESTRUCTURA
56                 },
```

```

57         "TRUCK_SHOP": {
58             "nombre": "Truck Shop/Almacenes",
59             "categoria": CategoriaObra.INFRAESTRUCTURA
60         },
61
62         # Servicios
63         "ELECTR_INSTRUM": {
64             "nombre": "Electricidad e Instrumentación",
65             "categoria": CategoriaObra.SERVICIOS
66         },
67         "MONTAJE": {
68             "nombre": "Montaje de Estructuras Pesadas",
69             "categoria": CategoriaObra.SERVICIOS
70         }
71     }
72
73     def get_categorias(self) -> List[CategoriaObra]:
74         """Obtiene todas las categorías disponibles"""
75         return list(set(tipo["categoria"] for tipo in self.tipos.values()))
76
77     def get_tipos_por_categoria(self, categoria: CategoriaObra) -> List[str]:
78         """Obtiene los tipos de obra de una categoría específica"""
79         return [
80             codigo for codigo, datos in self.tipos.items()
81             if datos["categoria"] == categoria
82         ]
83
84     def get_nombre(self, codigo: str) -> Optional[str]:
85         """Obtiene el nombre de un tipo de obra por su código"""
86         if codigo in self.tipos:
87             return self.tipos[codigo]["nombre"]
88         return None
89
90     class CalculadorExperiencia:
91         def __init__(self):
92             self.experiencias: Dict[str, ExperienciaObra] = {}
93
94         def agregar_experiencia(self, tipo_obra: str, años: float, proyecto: str):
95             """
96             Agrega experiencia para un tipo de obra específico
97             """
98             if tipo_obra not in self.experiencias:
99                 self.experiencias[tipo_obra] = ExperienciaObra(
100                     años=años,
101                     meses=int(años * 12),
102                     proyectos=[proyecto]
103                 )
104             else:
105                 exp = self.experiencias[tipo_obra]
106                 exp.años += años
107                 exp.meses = int(exp.años * 12)
108                 exp.proyectos.append(proyecto)
109
110         def get_experiencia_total(self) -> float:
111             """
112             Calcula la experiencia total en años
113             """
114             return sum(exp.años for exp in self.experiencias.values())

```

```
115
116 def get_experiencia_por_categoria(self, categoria: CategoriaObra) -> Dict[str, ExperienciaObra]:
117     """
118     Obtiene la experiencia agrupada por categoría
119     """
120     tipos_obra = TipoObra()
121     resultado = {}
122
123     for tipo, exp in self.experiencias.items():
124         if tipos_obra.tipos[tipo]["categoria"] == categoria:
125             resultado[tipo] = exp
126
127     return resultado
128
```

models\persona.py

```
1  # models/persona.py
2  from dataclasses import dataclass
3  from datetime import date
4  from typing import List, Optional
5  from enum import Enum
6
7  class TipoDocumento(Enum):
8      DNI = "DNI"
9      RUC = "RUC"
10     CE = "CE" # Carnet de Extranjería
11     CPP = "CPP" # Carnet de Permiso Temporal
12     PASAPORTE = "PASAPORTE"
13
14  @dataclass
15  class DatosPersonales:
16      # Campos requeridos (sin valores por defecto)
17      codigo: str
18      tipo_documento: TipoDocumento
19      numero_documento: str
20      nombres: str
21      apellidos: str
22      profesion: str
23      area: str
24      cargo_actual: str
25
26      # Campos opcionales (con valores por defecto)
27      fecha_nacimiento: Optional[date] = None
28      lugar_nacimiento: Optional[str] = None
29      nacionalidad: str = "Peruana"
30      telefono: Optional[str] = None
31      correo: Optional[str] = None
32      direccion: Optional[str] = None
33      ciudad_residencia: Optional[str] = None
34      pais_residencia: str = "Perú"
35      numero_colegiatura: Optional[str] = None
36      años_experiencia: float = 0.0
37      fecha_registro: date = date.today()
38      trabajado_eimsa: bool = False
39      estado_cv: str = "Activo"
40
41  @dataclass
42  class Experiencia:
43      # Campos requeridos
44      empresa: str
45      cargo: str
46      fecha_inicio: date
47      obra: str
48      tipo_obra: str
49      propietario: str
50
51      # Campos opcionales
52      id: Optional[int] = None
53      fecha_fin: Optional[date] = None
54      detalle_obra: str = ""
55      ubicacion: str = ""
56      monto_proyecto: float = 0.0
```

```

57     moneda_proyecto: str = "PEN"
58     funciones: List[str] = None
59     logros: List[str] = None
60     tecnologias_usadas: List[str] = None
61     documentos: List[str] = None
62
63     def __post_init__(self):
64         if self.funciones is None:
65             self.funciones = []
66         if self.logros is None:
67             self.logros = []
68         if self.tecnologias_usadas is None:
69             self.tecnologias_usadas = []
70         if self.documentos is None:
71             self.documentos = []
72
73     @property
74     def duracion_meses(self) -> int:
75         fin = self.fecha_fin or date.today()
76         return (fin.year - self.fecha_inicio.year) * 12 + (fin.month - self.fecha_inicio.month)
77
78 @dataclass
79 class CV:
80     datos_personales: DatosPersonales
81     experiencias: List[Experiencia]
82     habilidades: List[str] = None
83     certificaciones: List[str] = None
84     idiomas: List[dict] = None
85     educacion: List[dict] = None
86
87     def __post_init__(self):
88         if self.habilidades is None:
89             self.habilidades = []
90         if self.certificaciones is None:
91             self.certificaciones = []
92         if self.idiomas is None:
93             self.idiomas = []
94         if self.educacion is None:
95             self.educacion = []

```

models\personal.py

```
1  # app/models/personal.py
2  from dataclasses import dataclass
3  from datetime import datetime
4  from typing import List, Optional
5  from .database import Database
6
7  @dataclass
8  class Personal:
9      codigo: str
10     nombres: str
11     tipo_documento: str
12     numero_documento: str
13     profesion: str
14     area: str
15     cargo: str
16     fecha_nacimiento: Optional[str] = None
17     lugar_nacimiento: Optional[str] = None
18     registro_cip: Optional[str] = None
19     residencia: Optional[str] = None
20     telefono: Optional[str] = None
21     correo: Optional[str] = None
22     fecha_registro: Optional[str] = None
23
24     @staticmethod
25     def generar_codigo(area: str) -> str:
26         db = Database()
27         cursor = db.conn.cursor()
28
29         # Obtener el último código para el área
30         cursor.execute('''
31             SELECT codigo FROM datos_personales
32             WHERE codigo LIKE ?
33             ORDER BY codigo DESC LIMIT 1
34             ''', (f'O-____-EI-PE',))
35
36         ultimo_codigo = cursor.fetchone()
37
38         if ultimo_codigo:
39             # Extraer el número y aumentar en 1
40             num = int(ultimo_codigo[0].split('-')[1]) + 1
41         else:
42             num = 1
43
44         return f'O-{{num:05d}}-EI-PE'
45
46     def guardar(self) -> bool:
47         db = Database()
48         try:
49             with db.conn:
50                 db.conn.execute('''
51                     INSERT INTO datos_personales (
52                         codigo, nombres, profesion, fecha_nacimiento,
53                         lugar_nacimiento, registro_cip, area, cargo,
54                         residencia, telefono, correo
55                     ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
56                     ''', (
```

```

57         self.codigo, self.nombres, self.profesion,
58         self.fecha_nacimiento, self.lugar_nacimiento,
59         self.registro_cip, self.area, self.cargo,
60         self.residencia, self.telefono, self.correo
61     ))
62     return True
63 except Exception as e:
64     print(f"Error al guardar personal: {e}")
65     return False
66
67 def actualizar(self) -> bool:
68     db = Database()
69     try:
70         with db.conn:
71             db.conn.execute('''
72                 UPDATE datos_personales
73                 SET nombres=?, profesion=?, fecha_nacimiento=?,
74                     lugar_nacimiento=?, registro_cip=?, area=?,
75                     cargo=?, residencia=?, telefono=?, correo=?
76                 WHERE codigo=?
77             ''', (
78                 self.nombres, self.profesion, self.fecha_nacimiento,
79                 self.lugar_nacimiento, self.registro_cip, self.area,
80                 self.cargo, self.residencia, self.telefono,
81                 self.correo, self.codigo
82             ))
83     return True
84 except Exception as e:
85     print(f"Error al actualizar personal: {e}")
86     return False
87
88 @staticmethod
89 def buscar(texto: str, area: Optional[str] = None) -> List['Personal']:
90     db = Database()
91     cursor = db.conn.cursor()
92
93     query = '''
94         SELECT * FROM datos_personales
95         WHERE (nombres LIKE ? OR codigo LIKE ?)
96     '''
97     params = [f'%{texto}%', f'%{texto}%']
98
99     if area and area != 'Todas las áreas':
100         query += ' AND area = ?'
101         params.append(area)
102
103     cursor.execute(query, params)
104     resultados = cursor.fetchall()
105
106     return [Personal(**dict(row)) for row in resultados]
107
108 @staticmethod
109 def obtener_por_codigo(codigo: str) -> Optional['Personal']:
110     db = Database()
111     cursor = db.conn.cursor()
112
113     cursor.execute('SELECT * FROM datos_personales WHERE codigo = ?', (codigo,))
114     resultado = cursor.fetchone()

```

```
115
116     if resultado:
117         return Personal(**dict(resultado))
118     return None
```


models\project_filters.py

```
1  # app/models/project_filters.py
2
3  from enum import Enum
4  from typing import List, Dict, Optional
5  from dataclasses import dataclass
6  from decimal import Decimal
7
8  class MontoRango(Enum):
9      """Rangos de montos de proyecto definidos"""
10     MENOR_5MM = "< 5 MM"
11     ENTRE_5_20MM = "5 - 20 MM"
12     ENTRE_20_50MM = "20 - 50 MM"
13     ENTRE_50_100MM = "50 - 100 MM"
14     MAYOR_100MM = "> 100 MM"
15
16 @dataclass
17 class ProyectoMonto:
18     """Clase para manejar el monto de un proyecto"""
19     valor: Decimal
20     moneda: str
21
22     def get_rango(self) -> MontoRango:
23         """Determina el rango del monto del proyecto"""
24         monto_mm = float(self.valor) / 1_000_000
25
26         if monto_mm < 5:
27             return MontoRango.MENOR_5MM
28         elif monto_mm < 20:
29             return MontoRango.ENTRE_5_20MM
30         elif monto_mm < 50:
31             return MontoRango.ENTRE_20_50MM
32         elif monto_mm < 100:
33             return MontoRango.ENTRE_50_100MM
34         else:
35             return MontoRango.MAYOR_100MM
36
37 class ProjectFilters:
38     """Clase para manejar los filtros de búsqueda de proyectos y personal"""
39
40     def __init__(self, db_connection):
41         self.db = db_connection
42
43     def filter_by_residence(self, residence: str) -> List[Dict]:
44         """
45         Filtra personal por lugar de residencia
46         """
47         query = """
48         SELECT * FROM datos_personales
49         WHERE LOWER(residencia) LIKE LOWER(?)
50         """
51         return self.db.execute(query, (f"%{residence}%",)).fetchall()
52
53     def filter_by_name(self, name: str) -> List[Dict]:
54         """
55         Filtra personal por nombre o apellido
56         """
```

```

57         query = """
58         SELECT * FROM datos_personales
59         WHERE LOWER(nombres) LIKE LOWER(?)
60         """
61         return self.db.execute(query, (f"%{name}%",)).fetchall()
62
63     def filter_by_area(self, area: str) -> List[Dict]:
64         """
65         Filtra personal por área
66         """
67         query = """
68         SELECT * FROM datos_personales
69         WHERE area = ?
70         """
71         return self.db.execute(query, (area,)).fetchall()
72
73     def filter_by_reference_position(self, position: str) -> List[Dict]:
74         """
75         Filtra personal por cargo de referencia
76         """
77         query = """
78         SELECT DISTINCT dp.*
79         FROM datos_personales dp
80         JOIN experiencia_laboral el ON dp.codigo = el.codigo_personal
81         WHERE LOWER(el.cargo) LIKE LOWER(?)
82         """
83         return self.db.execute(query, (f"%{position}%",)).fetchall()
84
85     def filter_by_project_amount(self, rango: MontoRango) -> List[Dict]:
86         """
87         Filtra proyectos por rango de monto
88         """
89         ranges = {
90             MontoRango.MENOR_5MM: (0, 5_000_000),
91             MontoRango.ENTRE_5_20MM: (5_000_000, 20_000_000),
92             MontoRango.ENTRE_20_50MM: (20_000_000, 50_000_000),
93             MontoRango.ENTRE_50_100MM: (50_000_000, 100_000_000),
94             MontoRango.MAYOR_100MM: (100_000_000, float('inf'))
95         }
96
97         min_val, max_val = ranges[rango]
98
99         query = """
100         SELECT DISTINCT dp.*
101         FROM datos_personales dp
102         JOIN experiencia_laboral el ON dp.codigo = el.codigo_personal
103         WHERE CAST(REPLACE(REPLACE(el.monto_proyecto, 'S/', ''), '$', '' ) AS DECIMAL)
104         BETWEEN ? AND ?
105         """
106         return self.db.execute(query, (min_val, max_val)).fetchall()
107
108     def filter_previous_eimsa(self) -> List[Dict]:
109         """
110         Filtra personal que trabajó anteriormente en EIMSA
111         """
112         query = """
113         SELECT * FROM datos_personales
114         WHERE trabajo_previo_eimsa = 1

```

```
115         """
116         return self.db.execute(query).fetchall()
117
118     # Actualización de la estructura de la base de datos
119     DB_UPDATES = """
120     -- Agregar campo para trabajo previo en EIMSA
121     ALTER TABLE datos_personales ADD COLUMN trabajo_previo_eimsa BOOLEAN DEFAULT 0;
122
123     -- Actualizar tabla de experiencia_laboral para manejar montos como decimal
124     ALTER TABLE experiencia_laboral ADD COLUMN monto_decimal DECIMAL(15,2);
125     """
```

models\tipo_obra.py

```
1  # app/models/tipo_obra.py
2  from dataclasses import dataclass
3  from typing import List
4  from .database import Database
5  from typing import Optional
6
7
8  @dataclass
9  class TipoObra:
10     id: str
11     nombre: str
12     descripcion: Optional[str] = None
13
14     @staticmethod
15     def obtener_todos() -> List['TipoObra']:
16         db = Database()
17         cursor = db.conn.cursor()
18
19         cursor.execute('SELECT * FROM tipos_obra ORDER BY nombre')
20         resultados = cursor.fetchall()
21
22         return [TipoObra(**dict(row)) for row in resultados]
23
24     @staticmethod
25     def obtener_por_id(id: str) -> Optional['TipoObra']:
26         db = Database()
27         cursor = db.conn.cursor()
28
29         cursor.execute('SELECT * FROM tipos_obra WHERE id = ?', (id,))
30         resultado = cursor.fetchone()
31
32         if resultado:
33             return TipoObra(**dict(resultado))
34         return None
```

utils__init__.py

```
1  # app/utils/__init__.py
2  from .cv_generator import CVGenerator
3  from .validators import validate_email, validate_phone, validate_date
4  from .formatters import format_currency, format_date
5  from .code_generator import generate_code
6
7  __all__ = [
8      'CVGenerator',
9      'validate_email',
10     'validate_phone',
11     'validate_date',
12     'format_currency',
13     'format_date',
14     'generate_code'
15 ]
```

utils\code_generator.py

```
1  # app/utils/code_generator.py
2  import random
3  import string
4  from datetime import datetime
5  from typing import Optional
6  from app.models.database import Database
7
8  def generate_code(area: str = 'O', company: str = 'EI',
9                  country: str = 'PE') -> str:
10     """
11     Genera un código único para un nuevo registro
12     Formato: O-000001-EI-PE
13     """
14     db = Database()
15     cursor = db.conn.cursor()
16
17     # Obtener el último código
18     cursor.execute('''
19         SELECT codigo FROM datos_personales
20         WHERE codigo LIKE ?
21         ORDER BY codigo DESC LIMIT 1
22     ''', (f'{area}-%',))
23
24     result = cursor.fetchone()
25
26     if result:
27         # Extraer el número del último código y aumentar en 1
28         last_number = int(result[0].split('-')[1])
29         new_number = last_number + 1
30     else:
31         new_number = 1
32
33     # Generar el nuevo código
34     return f"{area}-{new_number:06d}-{company}-{country}"
35
36 def generate_temp_code() -> str:
37     """
38     Genera un código temporal para archivos o registros temporales
39     """
40     timestamp = datetime.now().strftime('%Y%m%d%H%M%S')
41     random_chars = ''.join(random.choices(string.ascii_uppercase + string.digits, k=4))
42     return f"TEMP-{timestamp}-{random_chars}"
```

utils\cv_generator.py

```
1  from docx import Document
2  from docx.shared import Pt, Inches, Cm
3  from docx.enum.text import WD_ALIGN_PARAGRAPH
4  from docx.enum.section import WD_ORIENT
5  from datetime import datetime
6  import os
7  from app.config import Config
8  from docx.oxml import parse_xml
9  from PIL import Image
10
11 class CVGenerator:
12     def __init__(self):
13         self.template_path = Config.CV_TEMPLATE_PATH
14         if not os.path.exists(self.template_path):
15             raise FileNotFoundError(f"No se encontró la plantilla en: {self.template_path}")
16
17     def _agregar_imagen(self, doc, ruta_imagen, nombre):
18         """Método para agregar imagen con formato correcto"""
19         try:
20             # Agregar párrafo para la imagen
21             img_paragraph = doc.add_paragraph()
22             img_paragraph.alignment = WD_ALIGN_PARAGRAPH.CENTER
23
24             # Obtener dimensiones de la imagen
25             with Image.open(ruta_imagen) as img:
26                 width, height = img.size
27
28             # Calcular el ancho máximo (15 cm, manteniendo la proporción)
29             max_width = Cm(15)
30             aspect_ratio = height / width
31             width = max_width
32             height = width * aspect_ratio
33
34             # Agregar la imagen centrada
35             run = img_paragraph.add_run()
36             run.add_picture(ruta_imagen, width=width)
37
38             # Espacio después de la imagen
39             doc.add_paragraph()
40
41         except Exception as e:
42             print(f"Error al agregar imagen {nombre}: {str(e)}")
43
44     def generar(self, datos_personales, experiencias, output_path):
45         try:
46             # Usar la plantilla existente en lugar de crear un nuevo documento
47             doc = Document(self.template_path)
48
49             # DATOS PERSONALES
50             titulo_dp = doc.add_paragraph('DATOS PERSONALES')
51             titulo_dp.runs[0].font.bold = True
52             titulo_dp.runs[0].font.size = Pt(16)
53
54             # Datos personales con indentación
55             campos = [
56                 ('APELLIDOS Y NOMBRES', datos_personales['nombres']),
```

```

57         ('PROFESION', datos_personales['profesion']),
58         ('FECHA DE NACIMIENTO', datos_personales['fecha_nacimiento']),
59         ('LUGAR DE NACIMIENTO', datos_personales['lugar_nacimiento']),
60         ('REGISTRO C.I.P.', datos_personales.get('registro_cip', '')),
61         ('RESIDENCIA', datos_personales.get('residencia', '')),
62         ('TELÉFONO', datos_personales.get('telefono', '')),
63         ('CORREO', datos_personales.get('correo', ''))
64     ]
65
66     # Tabla invisible para alineación
67     table = doc.add_table(rows=len(campos), cols=3)
68     table.allow_autofit = True
69
70     # Ajustar ancho y quitar bordes
71     for row in table.rows:
72         for cell in row.cells:
73             cell._tc.get_or_add_tcPr().append(parse_xml(r'<w:tcW xmlns:w="http://schemas
74             cell._element.get_or_add_tcPr().append(parse_xml(r'<w:tcBorders xmlns:w="htt
75
76     # Agregar datos con indentación
77     for i, (label, valor) in enumerate(campos):
78         cells = table.rows[i].cells
79
80         # Primera celda: etiqueta
81         label_para = cells[0].paragraphs[0]
82         label_para.paragraph_format.left_indent = Inches(0.5)
83         run = label_para.add_run(label)
84         run.font.bold = True
85
86         # Segunda celda: dos puntos centrados
87         cells[1].paragraphs[0].alignment = WD_ALIGN_PARAGRAPH.CENTER
88         cells[1].paragraphs[0].add_run(' : ')
89
90         # Tercera celda: valor
91         cells[2].paragraphs[0].add_run(valor)
92
93     doc.add_paragraph() # Espacio entre secciones
94
95     # EXPERIENCIA PROFESIONAL
96     titulo_exp = doc.add_paragraph('EXPERIENCIA PROFESIONAL')
97     titulo_exp.runs[0].font.bold = True
98     titulo_exp.runs[0].font.size = Pt(16)
99
100    # Experiencias
101    for exp in experiencias:
102        campos_exp = [
103            ('Empresa', exp['empresa']),
104            ('Obra', exp['obra']),
105            ('Detalle de la obra', exp['detalle_obra']),
106            ('Monto del proyecto', exp['monto_proyecto']),
107            ('Cargo', exp['cargo']),
108            ('Periodo', f"{exp['fecha_inicio']} - {exp['fecha_fin']}"),
109            ('Propietario', exp['propietario'])
110        ]
111
112    # Tabla invisible para cada experiencia
113    exp_table = doc.add_table(rows=len(campos_exp), cols=3)
114    exp_table.allow_autofit = True

```



```

115
116         # Quitar bordes
117         for row in exp_table.rows:
118             for cell in row.cells:
119                 cell._tc.get_or_add_tcPr().append(parse_xml(r'<w:tcW xmlns:w="http://schemas.microsoft.com/office/word/2012/wordml" w:width="100%"></w:tcW>'))
120                 cell._element.get_or_add_tcPr().append(parse_xml(r'<w:tcBorders xmlns:w="http://schemas.microsoft.com/office/word/2012/wordml" w:top="1" w:bottom="1" w:left="1" w:right="1"></w:tcBorders>'))
121
122         # Agregar datos de experiencia
123         for i, (label, valor) in enumerate(campos_exp):
124             cells = exp_table.rows[i].cells
125
126             # Etiqueta con indentación
127             label_para = cells[0].paragraphs[0]
128             label_para.paragraph_format.left_indent = Inches(0.5)
129             run = label_para.add_run(label)
130             run.font.bold = True
131
132             # Dos puntos centrados
133             cells[1].paragraphs[0].alignment = WD_ALIGN_PARAGRAPH.CENTER
134             cells[1].paragraphs[0].add_run(' : ')
135
136             # Valor
137             cells[2].paragraphs[0].add_run(valor)
138
139         # Funciones
140         if exp.get('funciones'):
141             func_para = doc.add_paragraph()
142             func_para.paragraph_format.left_indent = Inches(0.5)
143             func_para.add_run('Funciones:').bold = True
144
145             funciones = exp['funciones']
146             if isinstance(funciones, str):
147                 funciones = funciones.split(';')
148
149             for funcion in funciones:
150                 if funcion.strip():
151                     bullet_para = doc.add_paragraph()
152                     bullet_para.paragraph_format.left_indent = Inches(0.7)
153                     bullet_para.paragraph_format.space_after = Pt(0)
154                     bullet_para.paragraph_format.space_before = Pt(0)
155                     check = bullet_para.add_run('✓')
156                     check.font.size = Pt(9)
157                     bullet_para.add_run(' ' + funcion.strip())
158
159         # Imágenes de esta experiencia
160         if 'documentos' in exp and exp['documentos']:
161             for img in exp['documentos']:
162                 if os.path.exists(img['ruta']):
163                     self._agregar_imagen(doc, img['ruta'], img['nombre'])
164
165         doc.add_paragraph() # Espacio entre experiencias
166
167         # Documentos generales después de todas las experiencias
168         if 'imagenes' in datos_personales and datos_personales['imagenes']:
169             titulo_img = doc.add_paragraph('DOCUMENTOS ADJUNTOS')
170             titulo_img.runs[0].font.bold = True
171             titulo_img.runs[0].font.size = Pt(16)
172

```

```
173         for img in datos_personales['imagenes']:
174             if os.path.exists(img['ruta']):
175                 self._agregar_imagen(doc, img['ruta'], img['nombre'])
176
177         doc.save(output_path)
178         return True
179
180     except Exception as e:
181         print(f"Error generando CV: {e}")
182         return False
```

utils/formatters.py

```
1  # app/utils/formatters.py
2  from datetime import datetime
3  import locale
4
5  def format_currency(amount: str) -> str:
6      """
7      Formatea un monto monetario
8      """
9      try:
10         # Limpiar el string de caracteres no numéricos
11         clean_amount = ''.join(filter(lambda x: x.isdigit() or x == '.', amount))
12         value = float(clean_amount)
13
14         # Formatear con separadores de miles
15         locale.setlocale(locale.LC_ALL, '')
16         return locale.currency(value, grouping=True, symbol='')
17     except:
18         return amount
19
20 def format_date(date_str: str, input_format: str = '%Y-%m-%d',
21                 output_format: str = '%d/%m/%Y') -> str:
22     """
23     Formatea una fecha al formato deseado
24     """
25     try:
26         if date_str:
27             date_obj = datetime.strptime(date_str, input_format)
28             return date_obj.strftime(output_format)
29         return ""
30     except:
31         return date_str
32
33 def format_phone(phone: str) -> str:
34     """
35     Formatea un número telefónico
36     """
37     if not phone:
38         return ""
39
40     # Limpiar el número
41     clean_number = ''.join(filter(str.isdigit, phone))
42
43     if len(clean_number) == 9: # Número celular Perú
44         return f"{clean_number[:3]} {clean_number[3:6]} {clean_number[6:]}"
45     return phone
```

utils\setup_check.py

```
1  # En utils/setup_check.py
2  import os
3  from app.config import Config
4
5  def verificar_plantilla_cv():
6      if not os.path.exists(Config.CV_TEMPLATE_PATH):
7          print(f"ADVERTENCIA: No se encontró la plantilla de CV en {Config.CV_TEMPLATE_PATH}")
8          print("Creando directorio de templates...")
9          os.makedirs(os.path.dirname(Config.CV_TEMPLATE_PATH), exist_ok=True)
10         print(f"Por favor, coloque su plantilla de CV en: {Config.CV_TEMPLATE_PATH}")
11         return False
12     return True
```

utils\validators.py

```
1  # app/utils/validators.py
2  import re
3  from datetime import datetime
4  from typing import Tuple
5
6  def validate_email(email: str) -> Tuple[bool, str]:
7      """
8      Valida el formato de un correo electrónico
9      """
10     pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
11     if not email:
12         return True, "" # El email es opcional
13     if re.match(pattern, email):
14         return True, ""
15     return False, "Formato de correo electrónico inválido"
16
17 def validate_phone(phone: str) -> Tuple[bool, str]:
18     """
19     Valida el formato de un número telefónico
20     """
21     pattern = r'^\+?[0-9]{9,15}$'
22     if not phone:
23         return True, "" # El teléfono es opcional
24     if re.match(pattern, phone):
25         return True, ""
26     return False, "Formato de teléfono inválido"
27
28 def validate_date(date_str: str) -> Tuple[bool, str]:
29     """
30     Valida el formato de una fecha (YYYY-MM-DD)
31     """
32     try:
33         if date_str:
34             datetime.strptime(date_str, '%Y-%m-%d')
35             return True, ""
36     except ValueError:
37         return False, "Formato de fecha inválido (YYYY-MM-DD)"
38
39 def validate_required(value: str, field_name: str) -> Tuple[bool, str]:
40     """
41     Valida que un campo requerido no esté vacío
42     """
43     if not value or not value.strip():
44         return False, f"El campo {field_name} es requerido"
45     return True, ""
```

views__init__.py

```
1  # app/views/__init__.py
2  from .login_view import LoginView
3  from .main_view import MainView
4  from .personal_view import RegistroPersonalView
5  from .busqueda_view import BusquedaView
6  from .experiencia_view import ExperienciaView
7  from .detalle_view import DetalleView
8
9  __all__ = [
10     'LoginView',
11     'MainView',
12     'RegistroPersonalView',
13     'BusquedaView',
14     'ExperienciaView',
15     'DetalleView'
16 ]
```

views\busqueda_view.py

```
1  # app/views/busqueda_view.py
2  from PyQt5.QtWidgets import (QWidget, QVBoxLayout, QHBoxLayout, QLineEdit,
3      QPushButton, QTableWidgetItem, QLabel, QComboBox,
4      QCheckBox, QGroupBox)
5  from PyQt5.QtCore import pyqtSignal, Qt
6  from PyQt5.QtGui import QColor
7  from app.config import Config
8  from app.models.database import Database # Añadida esta importación
9
10 class BusquedaView(QWidget):
11     registro_seleccionado = pyqtSignal(str)
12
13     def __init__(self):
14         super().__init__()
15         self.db = Database() # Inicializar conexión a base de datos
16         self.setup_ui()
17
18     def setup_ui(self):
19         layout = QVBoxLayout()
20
21         # Grupo de filtros
22         filtros_group = QGroupBox("Filtros de Búsqueda")
23         filtros_layout = QVBoxLayout()
24
25         # Primera fila de filtros
26         filtros_row1 = QHBoxLayout()
27
28         self.busqueda_input = QLineEdit()
29         self.busqueda_input.setPlaceholderText("Buscar por nombre o código...")
30
31         self.area_filter = QComboBox()
32         self.area_filter.addItem(['Todas las areas'] + Config.AREAS)
33
34         filtros_row1.addWidget(QLabel("Buscar:"))
35         filtros_row1.addWidget(self.busqueda_input)
36         filtros_row1.addWidget(QLabel("Área:"))
37         filtros_row1.addWidget(self.area_filter)
38
39         # Segunda fila de filtros
40         filtros_row2 = QHBoxLayout()
41
42         self.tipo_obra_filter = QComboBox()
43         self.tipo_obra_filter.addItem(['Todos los tipos'] + Config.TIPOS_OBRA)
44
45         self.monto_filter = QComboBox()
46         self.monto_filter.addItem(['Todos los montos', '< 5 MM', '5 - 20 MM', '20 - 50 MM', '50 - 100 MM'])
47
48         filtros_row2.addWidget(QLabel("Tipo de Obra:"))
49         filtros_row2.addWidget(self.tipo_obra_filter)
50         filtros_row2.addWidget(QLabel("Rango de Monto:"))
51         filtros_row2.addWidget(self.monto_filter)
52
53         # Checkbox para EIMSA
54         self.eimsa_check = QCheckBox("Ex-empleados EIMSA")
55
56         # Añadir filtros al layout
```

```

57     filtros_layout.addLayout(filtros_row1)
58     filtros_layout.addLayout(filtros_row2)
59     filtros_layout.addWidget(self.eimsa_check)
60     filtros_group.setLayout(filtros_layout)
61     layout.addWidget(filtros_group)
62
63     # Botones de acción
64     actions_layout = QHBoxLayout()
65     self.btn_buscar = QPushButton("Buscar")
66     self.btn_limpiar = QPushButton("Limpiar Filtros")
67     self.btn_ver_detalle = QPushButton("Ver Detalle")
68     self.btn_generar_cv = QPushButton("Generar CV")
69     self.btn_editar = QPushButton("Editar")
70
71     # Estilo para los botones
72     self.btn_buscar.setStyleSheet("background-color: #3498db; color: white;")
73     self.btn_limpiar.setStyleSheet("background-color: #3498db; color: white;")
74     self.btn_ver_detalle.setStyleSheet("background-color: #95a5a6; color: white;")
75     self.btn_generar_cv.setStyleSheet("background-color: #95a5a6; color: white;")
76     self.btn_editar.setStyleSheet("background-color: #95a5a6; color: white;")
77
78     actions_layout.addWidget(self.btn_buscar)
79     actions_layout.addWidget(self.btn_limpiar)
80     actions_layout.addWidget(self.btn_ver_detalle)
81     actions_layout.addWidget(self.btn_generar_cv)
82     actions_layout.addWidget(self.btn_editar)
83
84     layout.addLayout(actions_layout)
85
86     # Tabla de resultados
87     self.label_resultados = QLabel("Resultados:")
88     layout.addWidget(self.label_resultados)
89     self.tabla_resultados = QTableWidgetItem()
90     self.tabla_resultados.setColumnCount(8)
91     self.tabla_resultados.setHorizontalHeaderLabels([
92         "Código", "Nombre", "Profesión", "Área",
93         "Tipo de Obra", "Monto", "Ex-EIMSA",
94         "Última Actualización"
95     ])
96
97     self.tabla_resultados.setAlternatingRowColors(True)
98     self.tabla_resultados.setSelectionBehavior(QTableWidgetItem.SelectRows)
99     self.tabla_resultados.setSelectionMode(QTableWidgetItem.SingleSelection)
100    self.tabla_resultados.horizontalHeader().setStretchLastSection(True)
101
102    layout.addWidget(self.tabla_resultados)
103
104    self.setLayout(layout)
105
106    # Conectar señales
107    self.tabla_resultados.itemSelectionChanged.connect(self.on_selection_change)
108    self.búsqueda_input.returnPressed.connect(self.btn_buscar.click)
109
110    def on_selection_change(self):
111        "" "Habilita/deshabilita botones según selección"" "
112        tiene_seleccion = len(self.tabla_resultados.selectedItems()) > 0
113        self.btn_ver_detalle.setEnabled(tiene_seleccion)
114        self.btn_generar_cv.setEnabled(tiene_seleccion)

```



```

115         self.btn_editar.setEnabled(tiene_seleccion)
116
117     def cargar_registros(self, registros=None):
118         try:
119             if registros is None:
120                 registros = self.db.obtener_todos_registros()
121
122                 self.tabla_resultados.setRowCount(0)
123                 for row, datos in enumerate(registros):
124                     self.tabla_resultados.insertRow(row)
125                     items = [
126                         datos['codigo'],
127                         datos['nombres'],
128                         datos['profesion'],
129                         datos['area'],
130                         datos.get('tipo_obra', ''),
131                         datos.get('monto_proyecto', ''),
132                         '✓' if datos.get('trabajo_previo_eimsa') else '',
133                         datos.get('fecha_registro', '')
134                     ]
135
136                     for col, valor in enumerate(items):
137                         item = QTableWidgetItem(str(valor))
138                         item.setFlags(item.flags() & ~Qt.ItemIsEditable)
139                         self.tabla_resultados.setItem(row, col, item)
140
141                 self.tabla_resultados.resizeColumnsToContents()
142         except Exception as e:
143             print(f"Error al cargar registros: {e}")
144
145     def mostrar_resultados(self, resultados):
146         """Muestra los resultados en la tabla"""
147         self.tabla_resultados.setRowCount(0)
148         for row, datos in enumerate(resultados):
149             self.tabla_resultados.insertRow(row)
150             items = [
151                 datos.get('codigo', ''),
152                 datos.get('nombres', ''),
153                 datos.get('profesion', ''),
154                 datos.get('area', ''),
155                 datos.get('cargo', ''),
156                 datos.get('monto_proyecto', ''),
157                 '✓' if datos.get('trabajo_previo_eimsa') else '',
158                 datos.get('fecha_registro', '')
159             ]
160
161             for col, valor in enumerate(items):
162                 item = QTableWidgetItem(str(valor))
163                 item.setFlags(item.flags() & ~Qt.ItemIsEditable)
164                 self.tabla_resultados.setItem(row, col, item)
165
166             self.tabla_resultados.resizeColumnsToContents()
167
168     def obtener_codigo_seleccionado(self):
169         items = self.tabla_resultados.selectedItems()
170         if items:
171             row = items[0].row()
172             return self.tabla_resultados.item(row, 0).text()

```

```
173         return None
174
175     def limpiar_filtros(self):
176         self.búsqueda_input.clear()
177         self.area_filter.setCurrentIndex(0)
178         self.tipo_obra_filter.setCurrentIndex(0)
179         self.monto_filter.setCurrentIndex(0)
180         self.eimsa_check.setChecked(False)
```

views\detalle_view.py

```
1  from PyQt5.QtWidgets import (
2      QWidget, QVBoxLayout, QHBoxLayout, QLabel, QPushButton,
3      QScrollArea, QFrame
4  )
5  from PyQt5.QtCore import Qt
6
7
8  class DetalleView(QWidget):
9      def __init__(self):
10         super().__init__()
11         self.setup_ui()
12
13     def setup_ui(self):
14         layout = QVBoxLayout()
15
16         # Área scrollable para el contenido
17         scroll = QScrollArea()
18         scroll.setWidgetResizable(True)
19
20         # Widget contenedor
21         self.content_widget = QWidget()
22         self.content_layout = QVBoxLayout(self.content_widget)
23
24         # Sección de datos personales
25         self.datos_personales = QFrame()
26         self.datos_personales.setFrameStyle(QFrame.StyledPanel)
27         datos_layout = QVBoxLayout(self.datos_personales)
28
29         self.lbl_titulo_datos = QLabel("Datos Personales")
30         self.lbl_titulo_datos.setStyleSheet("font-size: 16px; font-weight: bold;")
31         datos_layout.addWidget(self.lbl_titulo_datos)
32
33         self.grid_datos = QVBoxLayout()
34         datos_layout.addLayout(self.grid_datos)
35
36         self.content_layout.addWidget(self.datos_personales)
37
38         # Sección de experiencias
39         self.experiencias = QFrame()
40         self.experiencias.setFrameStyle(QFrame.StyledPanel)
41         exp_layout = QVBoxLayout(self.experiencias)
42
43         self.lbl_titulo_exp = QLabel("Experiencia Laboral")
44         self.lbl_titulo_exp.setStyleSheet("font-size: 16px; font-weight: bold;")
45         exp_layout.addWidget(self.lbl_titulo_exp)
46
47         self.experiencias_layout = QVBoxLayout()
48         exp_layout.addLayout(self.experiencias_layout)
49
50         self.content_layout.addWidget(self.experiencias)
51
52         scroll.setWidget(self.content_widget)
53         layout.addWidget(scroll)
54
55         # Botones de acción
56         buttons_layout = QHBoxLayout()
```

```

57         self.btn_editar = QPushButton("Editar")
58         self.btn_generar_cv = QPushButton("Generar CV")
59         self.btn_volver = QPushButton("Volver")
60
61         buttons_layout.addWidget(self.btn_editar)
62         buttons_layout.addWidget(self.btn_generar_cv)
63         buttons_layout.addWidget(self.btn_volver)
64
65         layout.addLayout(buttons_layout)
66
67         self.setLayout(layout)
68
69     def mostrar_datos(self, datos_personales, experiencias):
70         """Muestra los datos del registro en los campos correspondientes"""
71         if not datos_personales:
72             raise ValueError("Datos personales no proporcionados.")
73         if experiencias is None:
74             experiencias = []
75
76         # Limpiar layouts previos
77         self._limpiar_layout(self.grid_datos)
78         self._limpiar_layout(self.experiencias_layout)
79
80         # Mostrar datos personales
81         campos = [
82             ('Código:', datos_personales.get('codigo', '')),
83             ('Apellidos y Nombres:', datos_personales.get('nombres', '')),
84             ('Profesión:', datos_personales.get('profesion', '')),
85             ('Área:', datos_personales.get('area', '')),
86             ('Cargo:', datos_personales.get('cargo', '')),
87             ('Fecha de Nacimiento:', datos_personales.get('fecha_nacimiento', '')),
88             ('Lugar de Nacimiento:', datos_personales.get('lugar_nacimiento', '')),
89             ('Registro CIP:', datos_personales.get('registro_cip', '')),
90             ('Residencia:', datos_personales.get('residencia', '')),
91             ('Teléfono:', datos_personales.get('telefono', '')),
92             ('Correo:', datos_personales.get('correo', ''))
93         ]
94
95         for label_text, valor in campos:
96             if valor:
97                 container = QWidget()
98                 layout = QHBoxLayout(container)
99                 label = QLabel(label_text)
100                 label.setMinimumWidth(150)
101                 label.setStyleSheet("font-weight: bold;")
102                 valor_label = QLabel(str(valor))
103                 layout.addWidget(label)
104                 layout.addWidget(valor_label)
105                 layout.addStretch()
106                 self.grid_datos.addWidget(container)
107
108         # Mostrar experiencias
109         for exp in experiencias:
110             exp_frame = QFrame()
111             exp_frame.setFrameStyle(QFrame.StyledPanel)
112             exp_layout = QVBoxLayout(exp_frame)
113
114             # Título de la experiencia

```

```

115         titulo = QLabel(f"{exp.get('cargo', 'N/A')} en {exp.get('empresa', 'N/A')}")
116         titulo.setStyleSheet("font-weight: bold; font-size: 14px;")
117         exp_layout.addWidget(titulo)
118
119         # Detalles de la experiencia
120         detalles = [
121             ('Obra:', exp.get('obra', 'N/A')),
122             ('Periodo:', f"{exp.get('fecha_inicio', '')} - {exp.get('fecha_fin', 'Actualidad')}"
123             ('Tipo de Obra:', exp.get('tipo_obra', 'N/A')),
124             ('Monto del Proyecto:', exp.get('monto_proyecto', 'N/A')),
125             ('Propietario:', exp.get('propietario', 'N/A'))
126         ]
127
128         for label_text, valor in detalles:
129             if valor:
130                 container = QWidget()
131                 layout = QHBoxLayout(container)
132                 label = QLabel(label_text)
133                 label.setMinimumWidth(120)
134                 label.setStyleSheet("font-weight: bold;")
135                 valor_label = QLabel(str(valor))
136                 layout.addWidget(label)
137                 layout.addWidget(valor_label)
138                 layout.addStretch()
139                 exp_layout.addWidget(container)
140
141         # Funciones
142         if exp.get('funciones'):
143             funciones_label = QLabel("Funciones:")
144             funciones_label.setStyleSheet("font-weight: bold;")
145             exp_layout.addWidget(funciones_label)
146
147             funciones = exp['funciones'].split('\n')
148             for funcion in funciones:
149                 if funcion.strip():
150                     func_label = QLabel(f"• {funcion.strip()}")
151                     exp_layout.addWidget(func_label)
152
153             self.experiencias_layout.addWidget(exp_frame)
154
155         # Agregar espacio al final
156         self.experiencias_layout.addStretch()
157
158     def _limpiar_layout(self, layout):
159         while layout.count():
160             item = layout.takeAt(0)
161             if item.widget():
162                 item.widget().deleteLater()
163             elif item.layout():
164                 self._limpiar_layout(item.layout())
165

```

views\experiencia_view.py

```
1  from PyQt5.QtWidgets import (QWidget, QVBoxLayout, QHBoxLayout, QLineEdit,
2                                  QLabel, QComboBox, QPushButton, QDateEdit,
3                                  QTextEdit, QDialog, QListWidget, QFileDialog,
4                                  QMessageBox, QGroupBox)
5  import os
6  from PyQt5.QtCore import pyqtSignal, Qt, QDate
7  from app.config import Config
8
9  class ExperienciaView(QDialog):
10     guardado_success = pyqtSignal(dict)
11
12     def __init__(self, parent=None):
13         super().__init__(parent)
14         self.setWindowTitle("Agregar Experiencia Laboral")
15         self.setup_ui()
16
17     def setup_ui(self):
18         layout = QVBoxLayout()
19
20         # Campos de la experiencia
21         self.empresa_input = QLineEdit()
22         self.obra_input = QLineEdit()
23         self.tipo_obra_combo = QComboBox()
24         self.tipo_obra_combo.addItem(Config.TIPOS_OBRA)
25         self.cargo_input = QLineEdit()
26
27         self.fecha_inicio = QDateEdit()
28         self.fecha_inicio.setCalendarPopup(True)
29         self.fecha_fin = QDateEdit()
30         self.fecha_fin.setCalendarPopup(True)
31
32         self.detalle_obra = QTextEdit()
33         self.propietario = QLineEdit()
34
35         # Layout para monto con selector de moneda
36         monto_container = QWidget()
37         monto_layout = QHBoxLayout(monto_container)
38         monto_layout.setContentsMargins(0, 0, 0, 0)
39
40         self.moneda_combo = QComboBox()
41         self.moneda_combo.addItem('S/ ', '$')
42         self.monto_input = QLineEdit()
43         self.monto_input.setPlaceholderText("0.00")
44         self.monto_input.textChanged.connect(self.validar_monto)
45
46         monto_layout.addWidget(self.moneda_combo)
47         monto_layout.addWidget(self.monto_input)
48
49         # Agregar campos al layout
50         campos = [
51             ('Empresa:', self.empresa_input),
52             ('Obra:', self.obra_input),
53             ('Tipo de Obra:', self.tipo_obra_combo),
54             ('Cargo:', self.cargo_input),
55             ('Fecha de Inicio:', self.fecha_inicio),
56             ('Fecha de Fin:', self.fecha_fin),
```

```

57         ('Detalle de la Obra:', self.detalle_obra),
58         ('Monto del Proyecto:', monto_container),
59         ('Propietario:', self.propietario)
60     ]
61
62     for label_text, widget in campos:
63         container = QWidget()
64         field_layout = QHBoxLayout(container)
65         label = QLabel(label_text)
66         label.setMinimumWidth(120)
67         field_layout.addWidget(label)
68         field_layout.addWidget(widget)
69         layout.addWidget(container)
70
71     # Sección de funciones
72     funciones_group = QGroupBox("Funciones")
73     funciones_layout = QVBoxLayout()
74
75     nueva_funcion_layout = QHBoxLayout()
76     self.nueva_funcion_input = QLineEdit()
77     self.btn_agregar_funcion = QPushButton("Agregar Función")
78     self.btn_agregar_funcion.clicked.connect(self.agregar_funcion)
79     nueva_funcion_layout.addWidget(self.nueva_funcion_input)
80     nueva_funcion_layout.addWidget(self.btn_agregar_funcion)
81
82     self.funciones_list = QListWidget()
83     self.btn_eliminar_funcion = QPushButton("Eliminar Función")
84     self.btn_eliminar_funcion.clicked.connect(self.eliminar_funcion)
85
86     funciones_layout.addLayout(nueva_funcion_layout)
87     funciones_layout.addWidget(self.funciones_list)
88     funciones_layout.addWidget(self.btn_eliminar_funcion)
89     funciones_group.setLayout(funciones_layout)
90     layout.addWidget(funciones_group)
91
92     # Sección de imágenes
93     images_group = QGroupBox("Imágenes")
94     images_layout = QVBoxLayout()
95
96     self.images_list = QListWidget()
97     self.btn_agregar_imagen = QPushButton("Agregar Imagen")
98     self.btn_agregar_imagen.clicked.connect(self.adjuntar_imagen)
99     self.btn_eliminar_imagen = QPushButton("Eliminar Imagen")
100    self.btn_eliminar_imagen.clicked.connect(self.eliminar_imagen)
101    self.btn_eliminar_imagen.setEnabled(False)
102
103    images_layout.addWidget(self.images_list)
104    images_layout.addWidget(self.btn_agregar_imagen)
105    images_layout.addWidget(self.btn_eliminar_imagen)
106
107    images_group.setLayout(images_layout)
108    layout.addWidget(images_group)
109
110    # Botones de acción
111    buttons_layout = QHBoxLayout()
112    self.btn_guardar = QPushButton("Guardar")
113    self.btn_cancelar = QPushButton("Cancelar")
114    self.btn_guardar.clicked.connect(self.guardar)

```

```

115         self.btn_cancelar.clicked.connect(self.reject)
116
117         buttons_layout.addWidget(self.btn_guardar)
118         buttons_layout.addWidget(self.btn_cancelar)
119         layout.addLayout(buttons_layout)
120
121         self.setLayout(layout)
122
123         # Conexiones para imágenes
124         self.images_list.itemSelectionChanged.connect(
125             lambda: self.btn_eliminar_imagen.setEnabled(self.images_list.currentItem() is not No
126         )
127
128     def agregar_funcion(self):
129         funcion = self.nueva_funcion_input.text().strip()
130         if funcion:
131             self.funciones_list.addItem(funcion)
132             self.nueva_funcion_input.clear()
133
134     def eliminar_funcion(self):
135         current_item = self.funciones_list.currentItem()
136         if current_item:
137             self.funciones_list.takeItem(self.funciones_list.row(current_item))
138
139     def validar_monto(self, texto):
140         texto_limpio = ''.join(c for c in texto if c.isdigit() or c == '.')
141         if texto_limpio.count('.') > 1:
142             texto_limpio = texto_limpio[:texto_limpio.rfind('.')]
143         if '.' in texto_limpio:
144             partes = texto_limpio.split('.')
145             texto_limpio = f"{partes[0]}.{partes[1][:2]}"
146         if texto_limpio != texto:
147             self.monto_input.setText(texto_limpio)
148
149
150     def adjuntar_imagen(self):
151         try:
152             archivos, _ = QFileDialog.getOpenFileNames(
153                 self,
154                 "Seleccionar Imágenes",
155                 "",
156                 "Imágenes (*.png *.jpg *.jpeg *.bmp)"
157             )
158
159             if archivos:
160                 for archivo in archivos:
161                     nombre = os.path.basename(archivo)
162                     # Verificar si ya existe la imagen
163                     existe = False
164                     for i in range(self.images_list.count()):
165                         if self.images_list.item(i).text() == nombre:
166                             existe = True
167                             break
168
169                     if not existe:
170                         self.images_list.addItem(nombre)
171                         if not hasattr(self, 'imagenes'):
172                             self.imagenes = []

```



```

173             self.imagenes.append({
174                 'nombre': nombre,
175                 'ruta': archivo,
176                 'tipo': 'experiencia'
177             })
178     except Exception as e:
179         QMessageBox.critical(self, "Error", f"Error al adjuntar imagen: {str(e)}")
180
181     def eliminar_imagen(self):
182         current_row = self.images_list.currentRow()
183         if current_row >= 0:
184             self.images_list.takeItem(current_row)
185             if hasattr(self, 'imagenes'):
186                 self.imagenes.pop(current_row)
187
188     def guardar(self):
189         datos = self.obtener_datos()
190         self.guardado_success.emit(datos)
191         self.accept()
192
193     def obtener_datos(self):
194         funciones = []
195         for i in range(self.funciones_list.count()):
196             funciones.append(self.funciones_list.item(i).text())
197
198         documentos = []
199         for i in range(self.images_list.count()):
200             nombre = self.images_list.item(i).text()
201             # Buscar la ruta en las imágenes guardadas
202             for img in getattr(self, 'imagenes', []):
203                 if img['nombre'] == nombre:
204                     documentos.append({
205                         'nombre': img['nombre'],
206                         'ruta': img['ruta'],
207                         'tipo': img.get('tipo', 'experiencia')
208                     })
209             break
210
211         return {
212             'empresa': self.empresa_input.text().strip(),
213             'obra': self.obra_input.text().strip(),
214             'tipo_obra': self.tipo_obra_combo.currentText(),
215             'cargo': self.cargo_input.text().strip(),
216             'fecha_inicio': self.fecha_inicio.date().toString('dd/MM/yyyy'),
217             'fecha_fin': self.fecha_fin.date().toString('dd/MM/yyyy'),
218             'detalle_obra': self.detalle_obra.toPlainText().strip(),
219             'monto_proyecto': f"{self.moneda_combo.currentText()} {self.monto_input.text()}",
220             'propietario': self.propietario.text().strip(),
221             'funciones': funciones,
222             'documentos': documentos # Usar la lista de documentos procesada
223         }
224
225     def cargar_datos(self, experiencia):
226         try:
227             self.empresa_input.setText(experiencia['empresa'])
228             self.obra_input.setText(experiencia['obra'])
229
230             index = self.tipo_obra_combo.findText(experiencia['tipo_obra'])

```

```

231         if index >= 0:
232             self.tipo_obra_combo.setCurrentIndex(index)
233
234         self.cargo_input.setText(experiencia['cargo'])
235         self.fecha_inicio.setDate(QDate.fromString(experiencia['fecha_inicio'], 'dd-MM-yyyy'))
236
237         if experiencia['fecha_fin']:
238             self.fecha_fin.setDate(QDate.fromString(experiencia['fecha_fin'], 'dd-MM-yyyy'))
239
240         self.detalle_obra.setPlainText(experiencia['detalle_obra'])
241
242         # Separar monto y moneda
243         monto = experiencia['monto_proyecto']
244         if monto.startswith('S/'):
245             self.moneda_combo.setCurrentText('S/')
246             self.monto_input.setText(monto[2:].strip())
247         elif monto.startswith('$'):
248             self.moneda_combo.setCurrentText('$')
249             self.monto_input.setText(monto[1:].strip())
250         else:
251             self.monto_input.setText(monto)
252
253         self.propietario.setText(experiencia['propietario'])
254
255         # Cargar funciones
256         self.funciones_list.clear()
257         if isinstance(experiencia['funciones'], list):
258             for funcion in experiencia['funciones']:
259                 self.funciones_list.addItem(funcion)
260         elif isinstance(experiencia['funciones'], str):
261             for funcion in experiencia['funciones'].split(';'):
262                 if funcion.strip():
263                     self.funciones_list.addItem(funcion.strip())
264
265         # Cargar imágenes
266         self.imagenes_list.clear()
267         if 'documentos' in experiencia and experiencia['documentos']:
268             self.imagenes = experiencia['documentos'] # Guardamos la lista completa de imágenes
269             for doc in experiencia['documentos']:
270                 self.imagenes_list.addItem(doc['nombre'])
271
272     except Exception as e:
273         QMessageBox.critical(self, "Error", f"Error al cargar experiencia: {str(e)}")
274
275
276

```

views/login_view.py

```
1  # app/views/login_view.py
2  from PyQt5.QtWidgets import (QWidget, QVBoxLayout, QHBoxLayout, QLineEdit,
3                                QPushButton, QLabel)
4  from PyQt5.QtCore import pyqtSignal, Qt
5  from PyQt5.QtGui import QPixmap
6  from app.config import Config
7
8  class LoginView(QWidget):
9      loginSuccess = pyqtSignal()
10
11      def __init__(self):
12          super().__init__()
13          self.setWindowTitle(f"{Config.APP_NAME} - Login")
14          self.setup_ui()
15
16      def setup_ui(self):
17          layout = QVBoxLayout()
18
19          # Logo
20          logo_label = QLabel()
21          pixmap = QPixmap(Config.LOGO_PATH)
22          logo_label.setPixmap(pixmap.scaled(200, 200))
23          logo_label.setAlignment(Qt.AlignCenter)
24
25          # Título
26          title = QLabel(Config.APP_NAME)
27          title.setStyleSheet("font-size: 24px; font-weight: bold;")
28          title.setAlignment(Qt.AlignCenter)
29
30          # Campos de login
31          self.usuario_input = QLineEdit()
32          self.usuario_input.setPlaceholderText("Usuario")
33          self.password_input = QLineEdit()
34          self.password_input.setPlaceholderText("Contraseña")
35          self.password_input.setEchoMode(QLineEdit.Password)
36
37          # Botón de login
38          self.login_btn = QPushButton("Iniciar Sesión")
39
40          layout.addWidget(logo_label)
41          layout.addWidget(title)
42          layout.addWidget(self.usuario_input)
43          layout.addWidget(self.password_input)
44          layout.addWidget(self.login_btn)
45
46          self.setLayout(layout)
47
48          # Estilo
49          self.setStyleSheet("""
50              QWidget {
51                  background-color: white;
52              }
53              QPushButton {
54                  background-color: #4CAF50;
55                  color: white;
56                  padding: 8px;
```

```
57         border-radius: 4px;
58     }
59     QLineEdit {
60         padding: 8px;
61         border: 1px solid #ccc;
62         border-radius: 4px;
63     }
64     """)
```

views/main_view.py

```
1  # views/main_view.py
2  from PyQt5.QtWidgets import (
3      QMainWindow, QWidget, QTabWidget, QVBoxLayout, QHBoxLayout,
4      QLabel, QLineEdit, QComboBox, QDateEdit, QCheckBox,
5      QPushButton, QGroupBox, QScrollArea, QTableWidgetItem,
6      QTableWidgetItem, QFileDialog, QMessageBox
7  )
8  from PyQt5.QtCore import Qt, QDate
9  from app.views.widgets import ExperienciaWidget
10 from app.models.persona import TipoDocumento
11 from app.config import Config
12
13 class MainView(QMainWindow):
14     def __init__(self):
15         super().__init__()
16         self.setWindowTitle("Sistema de Gestión de CVs")
17         self.setMinimumSize(1200, 800)
18         # Eliminar la línea: self.setup_registro_tab()
19         self.setup_ui()
20
21     def setup_ui(self):
22         central_widget = QWidget()
23         self.setCentralWidget(central_widget)
24         layout = QVBoxLayout(central_widget)
25
26         # Crear tabs
27         self.tab_widget = QTabWidget()
28         self.tab_busqueda = self.crear_tab_busqueda()
29         self.tab_registro = self.crear_tab_registro()
30
31         self.tab_widget.addTab(self.tab_busqueda, "Búsqueda")
32         self.tab_widget.addTab(self.tab_registro, "Registro/Edición")
33
34         layout.addWidget(self.tab_widget)
35
36     def crear_tab_registro(self):
37         tab = QWidget()
38         layout = QVBoxLayout(tab)
39         scroll = QScrollArea()
40         scroll.setWidgetResizable(True)
41         content = QWidget()
42         content_layout = QVBoxLayout(content)
43
44         # Grupo de Identificación
45         grupo_id = QGroupBox("Identificación")
46         grupo_id_layout = QHBoxLayout()
47
48         self.tipo_doc_combo = QComboBox()
49         self.tipo_doc_combo.addItem([t.value for t in TipoDocumento])
50         self.num_doc_input = QLineEdit()
51         self.codigo_input = QLineEdit()
52         self.codigo_input.setReadOnly(True)
53
54         grupo_id_layout.addWidget(QLabel("Tipo Documento:"))
55         grupo_id_layout.addWidget(self.tipo_doc_combo)
56         grupo_id_layout.addWidget(QLabel("Número:"))
```

```

57     grupo_id_layout.addWidget(self.num_doc_input)
58     grupo_id_layout.addWidget(QLabel("Código:"))
59     grupo_id_layout.addWidget(self.codigo_input)
60     grupo_id.setLayout(grupo_id_layout)
61
62     # Grupo de Datos Personales
63     grupo_personal = QGroupBox("Datos Personales")
64     grupo_personal_layout = QVBoxLayout()
65
66     campos_personales = [
67         ('Nombres:', QLineEdit()),
68         ('Apellidos:', QLineEdit()),
69         ('Fecha Nacimiento:', QDateEdit()),
70         ('Lugar Nacimiento:', QLineEdit()),
71         ('Nacionalidad:', QLineEdit()),
72     ]
73
74     for label, widget in campos_personales:
75         fila = QHBoxLayout()
76         fila.addWidget(QLabel(label))
77         fila.addWidget(widget)
78         grupo_personal_layout.addLayout(fila)
79
80     grupo_personal.setLayout(grupo_personal_layout)
81
82     # Grupo de Contacto
83     grupo_contacto = QGroupBox("Datos de Contacto")
84     grupo_contacto_layout = QVBoxLayout()
85
86     campos_contacto = [
87         ('Teléfono:', QLineEdit()),
88         ('Correo:', QLineEdit()),
89         ('Dirección:', QLineEdit()),
90         ('Ciudad:', QLineEdit()),
91         ('País:', QLineEdit('Perú')),
92     ]
93
94     for label, widget in campos_contacto:
95         fila = QHBoxLayout()
96         fila.addWidget(QLabel(label))
97         fila.addWidget(widget)
98         grupo_contacto_layout.addLayout(fila)
99
100     grupo_contacto.setLayout(grupo_contacto_layout)
101
102     # Grupo Profesional
103     grupo_prof = QGroupBox("Datos Profesionales")
104     grupo_prof_layout = QVBoxLayout()
105
106     self.area_combo = QComboBox()
107     self.area_combo.addItem(Config.AREAS)
108
109     campos_prof = [
110         ('Profesión:', QLineEdit()),
111         ('Nº Colegiatura:', QLineEdit()),
112         ('Área:', self.area_combo),
113         ('Cargo Actual:', QLineEdit()),
114     ]

```

```

115
116     for label, widget in campos_prof:
117         fila = QHBoxLayout()
118         fila.addWidget(QLabel(label))
119         fila.addWidget(widget)
120         grupo_prof_layout.addLayout(fila)
121
122     self.check_eimsa = QCheckBox("¿Ha trabajado en EIMSA?")
123     grupo_prof_layout.addWidget(self.check_eimsa)
124     grupo_prof.setLayout(grupo_prof_layout)
125
126     # Grupo de Experiencias
127     grupo_exp = QGroupBox("Experiencias Laborales")
128     tab.experiencias_layout = QVBoxLayout() # Guardar referencia al layout
129     tab.btn_agregar_experiencia = QPushButton("Agregar Experiencia") # Crear botón como atr
130     tab.btn_agregar_experiencia.setStyleSheet("""
131         QPushButton {
132             background-color: #4CAF50;
133             color: white;
134             padding: 8px;
135             border-radius: 4px;
136         }
137         QPushButton:hover {
138             background-color: #45a049;
139         }
140     """)
141
142     # Crear el layout de experiencias
143     experiencias_container = QVBoxLayout()
144     experiencias_container.addWidget(tab.btn_agregar_experiencia)
145     tab.experiencias_layout = QVBoxLayout()
146     experiencias_container.addLayout(tab.experiencias_layout)
147     grupo_exp.setLayout(experiencias_container)
148
149     # Botones de acción
150     botones_layout = QHBoxLayout()
151     tab.btn_guardar = QPushButton("Guardar")
152     tab.btn_cancelar = QPushButton("Cancelar")
153     botones_layout.addWidget(tab.btn_guardar)
154     botones_layout.addWidget(tab.btn_cancelar)
155
156     # Agregar todos los grupos al layout principal
157     content_layout.addWidget(grupo_exp)
158     content_layout.addLayout(botones_layout)
159
160     scroll.setWidget(content)
161     layout.addWidget(scroll)
162     return tab
163
164 def crear_tab_busqueda(self):
165     tab = QWidget()
166     layout = QVBoxLayout(tab)
167
168     # Filtros
169     grupo_filtros = QGroupBox("Filtros de Búsqueda")
170     filtros_layout = QVBoxLayout()
171
172     # Primera fila de filtros

```

```

173     fila1 = QHBoxLayout()
174     self.busqueda_input = QLineEdit()
175     self.busqueda_input.setPlaceholderText("Buscar por nombre, documento o código...")
176     self.area_filtro = QComboBox()
177     self.area_filtro.addItem(['Todas las Áreas'] + Config.AREAS)
178
179     fila1.addWidget(QLabel("Búsqueda:"))
180     fila1.addWidget(self.busqueda_input)
181     fila1.addWidget(QLabel("Área:"))
182     fila1.addWidget(self.area_filtro)
183
184     # Segunda fila de filtros
185     fila2 = QHBoxLayout()
186     self.tipo_obra_filtro = QComboBox()
187     self.tipo_obra_filtro.addItem(['Todos'] + Config.TIPOS_OBRA)
188     self.cargo_filtro = QLineEdit()
189     self.cargo_filtro.setPlaceholderText("Filtrar por cargo...")
190
191     fila2.addWidget(QLabel("Tipo de Obra:"))
192     fila2.addWidget(self.tipo_obra_filtro)
193     fila2.addWidget(QLabel("Cargo:"))
194     fila2.addWidget(self.cargo_filtro)
195
196     # Tercera fila de filtros
197     fila3 = QHBoxLayout()
198     self.check_eimsa_filtro = QCheckBox("Ex-empleados EIMSA")
199     self.años_exp_filtro = QComboBox()
200     self.años_exp_filtro.addItem(['Cualquier experiencia', '0-2 años', '2-5 años', '5-10 años'])
201
202     fila3.addWidget(self.check_eimsa_filtro)
203     fila3.addWidget(QLabel("Años de Experiencia:"))
204     fila3.addWidget(self.años_exp_filtro)
205
206     # Botones de filtro
207     botones_filtro = QHBoxLayout()
208     self.btn_buscar = QPushButton("Buscar")
209     self.btn_limpiar = QPushButton("Limpiar Filtros")
210
211     botones_filtro.addWidget(self.btn_buscar)
212     botones_filtro.addWidget(self.btn_limpiar)
213
214     # Agregar todas las filas al grupo de filtros
215     filtros_layout.addLayout(fila1)
216     filtros_layout.addLayout(fila2)
217     filtros_layout.addLayout(fila3)
218     filtros_layout.addLayout(botones_filtro)
219     grupo_filtros.setLayout(filtros_layout)
220
221     # Tabla de resultados
222     self.tabla_resultados = QTableWidgetItem()
223     self.tabla_resultados.setColumnCount(10)
224     self.tabla_resultados.setHorizontalHeaderLabels([
225         "Código", "Documento", "Nombres", "Profesión", "Área",
226         "Cargo", "Años Exp.", "Ciudad", "Ex-EIMSA", "Última Act."
227     ])
228
229     # Botones de acción
230     acciones = QHBoxLayout()

```



```

231         self.btn_ver = QPushButton("Ver Detalle")
232         self.btn_editar = QPushButton("Editar")
233         self.btn_generar_cv = QPushButton("Generar CV")
234
235         acciones.addWidget(self.btn_ver)
236         acciones.addWidget(self.btn_editar)
237         acciones.addWidget(self.btn_generar_cv)
238
239         # Agregar todo al layout principal
240         layout.addWidget(grupo_filtros)
241         layout.addWidget(self.tabla_resultados)
242         layout.addLayout(acciones)
243
244         return tab
245
246     def crear_tab_busqueda(self):
247         tab = QWidget()
248         layout = QVBoxLayout(tab)
249
250         # Filtros
251         grupo_filtros = QGroupBox("Filtros de Búsqueda")
252         filtros_layout = QVBoxLayout()
253
254         # Primera fila de filtros
255         fila1 = QHBoxLayout()
256         tab.busqueda_input = QLineEdit()
257         tab.busqueda_input.setPlaceholderText("Buscar por nombre, documento o código...")
258         tab.area_filtro = QComboBox()
259         tab.area_filtro.addItem(['Todas las Áreas'] + Config.AREAS)
260
261         fila1.addWidget(QLabel("Búsqueda:"))
262         fila1.addWidget(tab.busqueda_input)
263         fila1.addWidget(QLabel("Área:"))
264         fila1.addWidget(tab.area_filtro)
265
266         # Segunda fila de filtros
267         fila2 = QHBoxLayout()
268         tab.tipo_obra_filtro = QComboBox()
269         tab.tipo_obra_filtro.addItem(['Todos'] + Config.TIPOS_OBRA)
270         tab.cargo_filtro = QLineEdit()
271         tab.cargo_filtro.setPlaceholderText("Filtrar por cargo...")
272
273         fila2.addWidget(QLabel("Tipo de Obra:"))
274         fila2.addWidget(tab.tipo_obra_filtro)
275         fila2.addWidget(QLabel("Cargo:"))
276         fila2.addWidget(tab.cargo_filtro)
277
278         # Tercera fila de filtros
279         fila3 = QHBoxLayout()
280         tab.check_eimsa_filtro = QCheckBox("Ex-empleados EIMSA")
281         tab.años_exp_filtro = QComboBox()
282         tab.años_exp_filtro.addItem(['Cualquier experiencia', '0-2 años', '2-5 años', '5-10 años'])
283
284         fila3.addWidget(tab.check_eimsa_filtro)
285         fila3.addWidget(QLabel("Años de Experiencia:"))
286         fila3.addWidget(tab.años_exp_filtro)
287
288         # Botones de filtro

```

```

289     botones_filtro = QHBoxLayout()
290     tab.btn_buscar = QPushButton("Buscar")
291     tab.btn_limpiar = QPushButton("Limpiar Filtros")
292
293     botones_filtro.addWidget(tab.btn_buscar)
294     botones_filtro.addWidget(tab.btn_limpiar)
295
296     # Agregar todas las filas al grupo de filtros
297     filtros_layout.addLayout(fila1)
298     filtros_layout.addLayout(fila2)
299     filtros_layout.addLayout(fila3)
300     filtros_layout.addLayout(botones_filtro)
301     grupo_filtros.setLayout(filtros_layout)
302
303     # Tabla de resultados
304     tab.tabla_resultados = QTableWidgetItem()
305     tab.tabla_resultados.setColumnCount(10)
306     tab.tabla_resultados.setHorizontalHeaderLabels([
307         "Código", "Documento", "Nombres", "Profesión", "Área",
308         "Cargo", "Años Exp.", "Ciudad", "Ex-EIMSA", "Última Act."
309     ])
310
311     # Botones de acción
312     acciones = QHBoxLayout()
313     tab.btn_ver = QPushButton("Ver Detalle")
314     tab.btn_editar = QPushButton("Editar")
315     tab.btn_generar_cv = QPushButton("Generar CV")
316
317     acciones.addWidget(tab.btn_ver)
318     acciones.addWidget(tab.btn_editar)
319     acciones.addWidget(tab.btn_generar_cv)
320
321     # Agregar todo al layout principal
322     layout.addWidget(grupo_filtros)
323     layout.addWidget(tab.tabla_resultados)
324     layout.addLayout(acciones)
325
326     return tab

```

views\personal_view.py

```
1  # app/views/personal_view.py
2  from PyQt5.QtWidgets import (QWidget, QVBoxLayout, QHBoxLayout, QLineEdit,
3                                QLabel, QComboBox, QPushButton, QDateEdit,
4                                QMainWindow, QScrollArea, QMessageBox)
5  from PyQt5.QtCore import pyqtSignal, Qt, QDate
6  from app.config import Config
7  from app.utils.validators import validate_email, validate_phone, validate_date
8  from .experiencia_view import ExperienciaView
9  from .widgets.experiencia_widget import ExperienciaWidget
10 import re
11
12 class RegistroPersonalView(QWidget):
13     # Definir señales
14     guardado_success = pyqtSignal()
15     experiencia_agregada = pyqtSignal(dict)
16     experiencia_eliminada = pyqtSignal(int)
17     guardado_success = pyqtSignal(dict) # Modificado para enviar los datos
18
19     def __init__(self):
20         super().__init__()
21         self.experiencias = []
22         self.setup_ui()
23
24     def validar_datos(self):
25         """
26         Valida los campos del formulario antes de guardar
27         """
28         try:
29             # Validar campos requeridos
30             if not self.nombres_input.text().strip():
31                 QMessageBox.warning(self, "Error", "El campo Apellidos y Nombres es requerido")
32                 self.nombres_input.setFocus()
33                 return False
34
35             if not self.profesion_input.text().strip():
36                 QMessageBox.warning(self, "Error", "El campo Profesión es requerido")
37                 self.profesion_input.setFocus()
38                 return False
39
40             if not self.cargo_input.text().strip():
41                 QMessageBox.warning(self, "Error", "El campo Cargo es requerido")
42                 self.cargo_input.setFocus()
43                 return False
44
45             # Validar formato de correo si se ha ingresado
46             correo = self.correo_input.text().strip()
47             if correo and not self._validar_formato_correo(correo):
48                 QMessageBox.warning(self, "Error", "El formato del correo electrónico no es válido")
49                 self.correo_input.setFocus()
50                 return False
51
52             # Validar formato de teléfono si se ha ingresado
53             telefono = self.telefono_input.text().strip()
54             if telefono and not self._validar_formato_telefono(telefono):
55                 QMessageBox.warning(self, "Error", "El formato del teléfono no es válido")
56                 self.telefono_input.setFocus()
```

```

57         return False
58
59         # Si todas las validaciones pasan, emitir señal con los datos
60         datos = self.obtener_datos()
61         self.guardado_success.emit(datos)
62         return True
63
64     except Exception as e:
65         QMessageBox.critical(self, "Error", f"Error al validar datos: {str(e)}")
66         return False
67
68     def _validar_formato_correo(self, correo):
69         """
70         Valida el formato del correo electrónico
71         """
72         patron = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
73         return bool(re.match(patron, correo))
74
75     def _validar_formato_telefono(self, telefono):
76         """
77         Valida el formato del teléfono
78         """
79         # Acepta números con o sin código de país, mínimo 9 dígitos
80         patron = r'^\+?[0-9]{9,15}$'
81         return bool(re.match(patron, telefono))
82
83     def obtener_datos(self):
84         """
85         Recopila todos los datos del formulario
86         """
87         return {
88             'codigo': self.codigo_input.text().strip(),
89             'nombres': self.nombres_input.text().strip(),
90             'profesion': self.profesion_input.text().strip(),
91             'fecha_nacimiento': self.fecha_nac_input.date().toString('dd-MM-yyyy'),
92             'lugar_nacimiento': self.lugar_nac_input.text().strip(),
93             'registro_cip': self.registro_cip_input.text().strip(),
94             'area': self.area_combo.currentText(),
95             'cargo': self.cargo_input.text().strip(),
96             'residencia': self.residencia_input.text().strip(),
97             'telefono': self.telefono_input.text().strip(),
98             'correo': self.correo_input.text().strip(),
99             'experiencias': self.experiencias
100         }
101
102     def agregar_experiencia(self):
103         """
104         Abre el diálogo para agregar una nueva experiencia laboral
105         """
106         try:
107             dialogo = ExperienciaView(self)
108             if dialogo.exec_():
109                 # Si el diálogo se cerró con Aceptar
110                 datos_experiencia = dialogo.obtener_datos()
111                 self.experiencias.append(datos_experiencia)
112                 self.actualizar_lista_experiencias()
113         except Exception as e:
114             QMessageBox.critical(

```

```

115         self,
116         "Error",
117         f"Error al agregar experiencia: {str(e)}"
118     )
119
120     def actualizar_lista_experiencias(self):
121         """
122         Actualiza la visualización de la lista de experiencias
123         """
124         try:
125             # Limpiar el layout de experiencias existente
126             while self.experiencias_layout.count():
127                 item = self.experiencias_layout.takeAt(0)
128                 if item.widget():
129                     item.widget().deleteLater()
130
131             # Agregar las experiencias actualizadas
132             for i, exp in enumerate(self.experiencias):
133                 exp_widget = QWidget()
134                 exp_layout = QVBoxLayout(exp_widget)
135
136                 # Mostrar información resumida de la experiencia
137                 titulo = QLabel(f"{exp['cargo']} en {exp['empresa']}")
138                 titulo.setStyleSheet("font-weight: bold;")
139                 periodo = QLabel(f"Periodo: {exp['fecha_inicio']} - {exp['fecha_fin']} or 'Actual")
140
141                 exp_layout.addWidget(titulo)
142                 exp_layout.addWidget(periodo)
143
144                 # Botones de acción
145                 botones_layout = QHBoxLayout()
146                 btn_editar = QPushButton("Editar")
147                 btn_eliminar = QPushButton("Eliminar")
148
149                 btn_editar.clicked.connect(lambda checked, idx=i: self.editar_experiencia(idx))
150                 btn_eliminar.clicked.connect(lambda checked, idx=i: self.eliminar_experiencia(id
151
152                 botones_layout.addWidget(btn_editar)
153                 botones_layout.addWidget(btn_eliminar)
154                 exp_layout.addLayout(botones_layout)
155
156                 self.experiencias_layout.addWidget(exp_widget)
157         except Exception as e:
158             QMessageBox.critical(
159                 self,
160                 "Error",
161                 f"Error al actualizar lista de experiencias: {str(e)}"
162             )
163
164     def editar_experiencia(self, index):
165         """
166         Abre el diálogo para editar una experiencia existente
167         """
168         try:
169             dialogo = ExperienciaView(self)
170             dialogo.cargar_datos(self.experiencias[index])
171             if dialogo.exec_():
172                 self.experiencias[index] = dialogo.obtener_datos()

```

```

173         self.actualizar_lista_experiencias()
174     except Exception as e:
175         QMessageBox.critical(
176             self,
177             "Error",
178             f"Error al editar experiencia: {str(e)}"
179         )
180
181     def eliminar_experiencia(self, index):
182         """
183         Elimina una experiencia de la lista
184         """
185         try:
186             respuesta = QMessageBox.question(
187                 self,
188                 "Confirmar eliminación",
189                 "¿Está seguro de que desea eliminar esta experiencia?",
190                 QMessageBox.Yes | QMessageBox.No
191             )
192
193             if respuesta == QMessageBox.Yes:
194                 self.experiencias.pop(index)
195                 self.actualizar_lista_experiencias()
196         except Exception as e:
197             QMessageBox.critical(
198                 self,
199                 "Error",
200                 f"Error al eliminar experiencia: {str(e)}"
201             )
202
203     def setup_validations(self):
204         """Configurar validaciones de campos"""
205         # Validación de correo electrónico
206         self.correo_input.textChanged.connect(self.validar_correo)
207         # Validación de teléfono
208         self.telefono_input.textChanged.connect(self.validar_telefono)
209         # No permitir fechas futuras
210         self.fecha_nac_input.setMaximumDate(QDate.currentDate())
211
212     def setup_ui(self):
213         main_layout = QHBoxLayout()
214
215         # Panel izquierdo: Datos personales
216         left_panel = QWidget()
217         left_layout = QVBoxLayout(left_panel)
218
219         # Campos de datos personales
220         self.codigo_input = QLineEdit()
221         self.codigo_input.setReadOnly(True)
222         self.nombres_input = QLineEdit()
223         self.profesion_input = QLineEdit()
224         self.fecha_nac_input = QDateEdit()
225         self.fecha_nac_input.setCalendarPopup(True)
226         self.lugar_nac_input = QLineEdit()
227         self.registro_cip_input = QLineEdit()
228         self.area_combo = QComboBox()
229         self.area_combo.addItems(Config.AREAS)
230         self.cargo_input = QLineEdit()

```

```

231 self.residencia_input = QLineEdit()
232 self.telefono_input = QLineEdit()
233 self.correo_input = QLineEdit()
234
235 # Agregar campos al layout izquierdo
236 campos = [
237     ('Código:', self.codigo_input),
238     ('Apellidos y Nombres:', self.nombres_input),
239     ('Profesión:', self.profesion_input),
240     ('Fecha de Nacimiento:', self.fecha_nac_input),
241     ('Lugar de Nacimiento:', self.lugar_nac_input),
242     ('Registro CIP:', self.registro_cip_input),
243     ('Área:', self.area_combo),
244     ('Cargo:', self.cargo_input),
245     ('Residencia:', self.residencia_input),
246     ('Teléfono:', self.telefono_input),
247     ('Correo:', self.correo_input)
248 ]
249
250 for label_text, widget in campos:
251     container = QWidget()
252     layout = QHBoxLayout(container)
253     label = QLabel(label_text)
254     label.setMinimumWidth(150)
255     layout.addWidget(label)
256     layout.addWidget(widget)
257     left_layout.addWidget(container)
258
259 # Crear layout para botones
260 buttons_layout = QHBoxLayout()
261 self.btn_guardar = QPushButton("Guardar")
262 self.btn_cancelar = QPushButton("Cancelar") # Nuevo botón
263 self.btn_cancelar.clicked.connect(self.cancelar) # Nueva conexión
264 buttons_layout.addWidget(self.btn_guardar)
265 buttons_layout.addWidget(self.btn_cancelar)
266 left_layout.addLayout(buttons_layout)
267
268 # Panel derecho: Experiencias laborales
269 right_panel = QWidget()
270 right_layout = QVBoxLayout(right_panel)
271
272 # Título de experiencias
273 titulo_exp = QLabel("Experiencias Laborales")
274 titulo_exp.setStyleSheet("font-weight: bold; font-size: 14px;")
275 right_layout.addWidget(titulo_exp)
276
277 # Contenedor scrollable para experiencias
278 scroll = QScrollArea()
279 experiencias_container = QWidget()
280 self.experiencias_layout = QVBoxLayout(experiencias_container)
281 scroll.setWidget(experiencias_container)
282 scroll.setWidgetResizable(True)
283 right_layout.addWidget(scroll)
284
285 # Botón agregar experiencia
286 self.btn_agregar_experiencia = QPushButton("Agregar Experiencia")
287 self.btn_agregar_experiencia.setStyleSheet("")
288     QPushButton {

```

```

289         background-color: #4CAF50;
290         color: white;
291         padding: 8px;
292         border-radius: 4px;
293     }
294     QPushButton:hover {
295         background-color: #45a049;
296     }
297 """
298 right_layout.addWidget(self.btn_agregar_experiencia)
299
300 # Agregar paneles al layout principal
301 main_layout.addWidget(left_panel, stretch=1)
302 main_layout.addWidget(right_panel, stretch=1)
303
304 self.setLayout(main_layout)
305
306 # Conexiones
307 self.btn_guardar.clicked.connect(self.validar_datos)
308 self.btn_agregar_experiencia.clicked.connect(self.agregar_experiencia)
309
310 def limpiar_campos(self):
311     """Limpia todos los campos del formulario"""
312     self.codigo_input.clear()
313     self.nombres_input.clear()
314     self.profesion_input.clear()
315     self.fecha_nac_input.setDate(QDate.currentDate())
316     self.lugar_nac_input.clear()
317     self.registro_cip_input.clear()
318     self.area_combo.setCurrentIndex(0)
319     self.cargo_input.clear()
320     self.residencia_input.clear()
321     self.telefono_input.clear()
322     self.correo_input.clear()
323
324     # Limpiar experiencias
325     self.experiencias = []
326     while self.experiencias_layout.count():
327         item = self.experiencias_layout.takeAt(0)
328         if item.widget():
329             item.widget().deleteLater()
330
331 def cargar_datos(self, datos):
332     """Carga los datos del registro en el formulario"""
333     try:
334         self.codigo_input.setText(datos['codigo'])
335         self.nombres_input.setText(datos['nombres'])
336         self.profesion_input.setText(datos['profesion'])
337         self.lugar_nac_input.setText(datos['lugar_nacimiento'])
338         self.registro_cip_input.setText(datos['registro_cip'])
339
340         # Convertir la fecha de nacimiento a QDate
341         fecha_nac = QDate.fromString(datos['fecha_nacimiento'], 'yyyy-MM-dd')
342         self.fecha_nac_input.setDate(fecha_nac)
343
344         # Seleccionar el área correcta
345         index = self.area_combo.findText(datos['area'])
346         if index >= 0:

```



```

347         self.area_combo.setCurrentIndex(index)
348
349         self.cargo_input.setText(datos['cargo'])
350         self.residencia_input.setText(datos['residencia'])
351         self.telefono_input.setText(datos['telefono'])
352         self.correo_input.setText(datos['correo'])
353
354         # Cargar experiencias
355         self.experiencias = datos.get('experiencias', [])
356         self.actualizar_lista_experiencias()
357
358     except Exception as e:
359         QMessageBox.critical(self, "Error", f"Error al cargar datos: {str(e)}")
360
361     def cancelar(self):
362         """Cancela el registro y regresa a la vista de búsqueda"""
363         # Subir hasta encontrar MainView
364         parent = self.parent()
365         while parent and not isinstance(parent, QMainWindow):
366             parent = parent.parent()
367
368         if parent:
369             parent.stack.setCurrentWidget(parent.búsqueda_view)
370             self.limpiar_campos()
371
372

```

views\widgets__init__.py

```
1  # views/widgets/__init__.py
2  from .experiencia_widget import ExperienciaWidget
3
4  __all__ = ['ExperienciaWidget']
```

views\widgets\experiencia_widget.py

```
1  # views/widgets/experiencia_widget.py
2  from PyQt5.QtWidgets import (
3      QWidget, QVBoxLayout, QHBoxLayout, QLabel,
4      QLineEdit, QDateEdit, QTextEdit, QPushButton,
5      QComboBox, QGroupBox, QFileDialog, QListWidget
6  )
7  from PyQt5.QtCore import Qt, QDate, pyqtSignal
8  from PyQt5.QtWidgets import QDialog
9
10 class ExperienciaWidget(QDialog):
11     eliminado = pyqtSignal(QDialog)
12     actualizado = pyqtSignal(dict)
13
14     def __init__(self, datos=None, parent=None):
15         super().__init__(parent)
16         self.setWindowTitle("Experiencia Laboral")
17         self.setup_ui()
18         if datos:
19             self.cargar_datos(datos)
20
21         # Agregar botones de aceptar/cancelar
22         buttons_layout = QHBoxLayout()
23         self.btn_aceptar = QPushButton("Aceptar")
24         self.btn_cancelar = QPushButton("Cancelar")
25
26         self.btn_aceptar.clicked.connect(self.accept)
27         self.btn_cancelar.clicked.connect(self.reject)
28
29         buttons_layout.addWidget(self.btn_aceptar)
30         buttons_layout.addWidget(self.btn_cancelar)
31
32         # Asegurarse de que el layout principal sea QVBoxLayout
33         if not hasattr(self, 'layout'):
34             self.setLayout(QVBoxLayout())
35         self.layout().addLayout(buttons_layout)
36
37     def setup_ui(self):
38         layout = QVBoxLayout(self)
39         self.setStyleSheet = lambda: None # Para compatibilidad con QFrame
40
41         # Crear grupo principal
42         grupo = QGroupBox("Experiencia Laboral")
43         grupo_layout = QVBoxLayout()
44
45         # Primera fila: Empresa y Cargo
46         fila1 = QHBoxLayout()
47         self.empresa_input = QLineEdit()
48         self.cargo_input = QLineEdit()
49
50         fila1.addWidget(QLabel("Empresa:"))
51         fila1.addWidget(self.empresa_input)
52         fila1.addWidget(QLabel("Cargo:"))
53         fila1.addWidget(self.cargo_input)
54
55         # Segunda fila: Fechas
56         fila2 = QHBoxLayout()
```

```

57     self.fecha_inicio = QDateEdit()
58     self.fecha_inicio.setCalendarPopup(True)
59     self.fecha_fin = QDateEdit()
60     self.fecha_fin.setCalendarPopup(True)
61
62     fila2.addWidget(QLabel("Fecha Inicio:"))
63     fila2.addWidget(self.fecha_inicio)
64     fila2.addWidget(QLabel("Fecha Fin:"))
65     fila2.addWidget(self.fecha_fin)
66
67     # Tercera fila: Obra y Tipo
68     fila3 = QHBoxLayout()
69     self.obra_input = QLineEdit()
70     self.tipo_obra_combo = QComboBox()
71     # Agregar tipos de obra desde Config
72     self.tipo_obra_combo.addItem('Planta de Cemento', 'Minería', 'Hidroeléctrica'))
73
74     fila3.addWidget(QLabel("Obra:"))
75     fila3.addWidget(self.obra_input)
76     fila3.addWidget(QLabel("Tipo:"))
77     fila3.addWidget(self.tipo_obra_combo)
78
79     # Cuarta fila: Propietario y Monto
80     fila4 = QHBoxLayout()
81     self.propietario_input = QLineEdit()
82     self.monto_input = QLineEdit()
83     self.moneda_combo = QComboBox()
84     self.moneda_combo.addItem('S/', 'USD'))
85
86     fila4.addWidget(QLabel("Propietario:"))
87     fila4.addWidget(self.propietario_input)
88     fila4.addWidget(QLabel("Monto:"))
89     fila4.addWidget(self.moneda_combo)
90     fila4.addWidget(self.monto_input)
91
92     # Detalle de la obra
93     self.detalle_obra = QTextEdit()
94     self.detalle_obra.setMaximumHeight(100)
95     detalle_container = QVBoxLayout()
96     detalle_container.addWidget(QLabel("Detalle de la Obra:"))
97     detalle_container.addWidget(self.detalle_obra)
98
99     # Funciones
100    funciones_grupo = QGroupBox("Funciones")
101    funciones_layout = QVBoxLayout()
102    self.funciones_list = QListWidget()
103    self.funcion_input = QLineEdit()
104    btn_agregar_funcion = QPushButton("Agregar Función")
105    btn_eliminar_funcion = QPushButton("Eliminar Función")
106
107    funciones_layout.addWidget(self.funcion_input)
108    funciones_layout.addWidget(btn_agregar_funcion)
109    funciones_layout.addWidget(self.funciones_list)
110    funciones_layout.addWidget(btn_eliminar_funcion)
111    funciones_grupo.setLayout(funciones_layout)
112
113    # Botones de acción
114    botones = QHBoxLayout()

```

```

115         self.btn_eliminar = QPushButton("Eliminar Experiencia")
116         botones.addWidget(self.btn_eliminar)
117
118         # Conectar señales
119         self.btn_eliminar.clicked.connect(lambda: self.eliminado.emit(self))
120         btn_agregar_funcion.clicked.connect(self.agregar_funcion)
121         btn_eliminar_funcion.clicked.connect(self.eliminar_funcion)
122
123         # Agregar todo al layout principal
124         grupo_layout.addLayout(fila1)
125         grupo_layout.addLayout(fila2)
126         grupo_layout.addLayout(fila3)
127         grupo_layout.addLayout(fila4)
128         grupo_layout.addLayout(detalle_container)
129         grupo_layout.addWidget(funciones_grupo)
130         grupo_layout.addLayout(botones)
131
132         grupo.setLayout(grupo_layout)
133         layout.addWidget(grupo)
134
135     def agregar_funcion(self):
136         texto = self.funcion_input.text().strip()
137         if texto:
138             self.funciones_list.addItem(texto)
139             self.funcion_input.clear()
140             self.actualizado.emit(self.obtener_datos())
141
142     def eliminar_funcion(self):
143         item = self.funciones_list.currentItem()
144         if item:
145             self.funciones_list.takeItem(self.funciones_list.row(item))
146             self.actualizado.emit(self.obtener_datos())
147
148     def obtener_datos(self):
149         funciones = []
150         for i in range(self.funciones_list.count()):
151             funciones.append(self.funciones_list.item(i).text())
152
153         return {
154             'empresa': self.empresa_input.text(),
155             'cargo': self.cargo_input.text(),
156             'fecha_inicio': self.fecha_inicio.date().toString('dd/MM/yyyy'),
157             'fecha_fin': self.fecha_fin.date().toString('dd/MM/yyyy'),
158             'obra': self.obra_input.text(),
159             'tipo_obra': self.tipo_obra_combo.currentText(),
160             'propietario': self.propietario_input.text(),
161             'monto_proyecto': f"{self.moneda_combo.currentText()} {self.monto_input.text()}",
162             'detalle_obra': self.detalle_obra.toPlainText(),
163             'funciones': funciones
164         }
165
166     def cargar_datos(self, datos):
167         self.empresa_input.setText(datos.get('empresa', ''))
168         self.cargo_input.setText(datos.get('cargo', ''))
169
170         # Cargar fechas
171         if 'fecha_inicio' in datos:
172             self.fecha_inicio.setDate(QDate.fromString(datos['fecha_inicio'], 'dd/MM/yyyy'))

```

```
173         if 'fecha_fin' in datos:
174             self.fecha_fin.setDate(QDate.fromString(datos['fecha_fin'], 'dd/MM/yyyy'))
175
176         self.obra_input.setText(datos.get('obra', ''))
177
178         # Cargar tipo de obra
179         tipo_obra = datos.get('tipo_obra', '')
180         index = self.tipo_obra_combo.findText(tipo_obra)
181         if index >= 0:
182             self.tipo_obra_combo.setCurrentIndex(index)
183
184         self.propietario_input.setText(datos.get('propietario', ''))
185
186         # Cargar monto y moneda
187         monto = datos.get('monto_proyecto', '')
188         if monto:
189             if monto.startswith('S/'):
190                 self.moneda_combo.setCurrentText('S/')
191                 self.monto_input.setText(monto[2:].strip())
192             elif monto.startswith('USD'):
193                 self.moneda_combo.setCurrentText('USD')
194                 self.monto_input.setText(monto[3:].strip())
195
196         self.detalle_obra.setPlainText(datos.get('detalle_obra', ''))
197
198         # Cargar funciones
199         self.funciones_list.clear()
200         funciones = datos.get('funciones', [])
201         if isinstance(funciones, str):
202             funciones = funciones.split(';')
203         for funcion in funciones:
204             if funcion.strip():
205                 self.funciones_list.addItem(funcion.strip())
```