==========

TGA DESKTOP AUTOMATION - SISTEMA ALEMÃO COMPLETO

Sistema de Automação Arquitetônica com Normas Alemãs Online

Versão: German Standards Edition 1.0

==========

“”“

ESTRUTURA FOCADA EM NORMAS ALEMÃS:

```
TGA_Desktop_Automation/
 ├── 📂 main.py              # Arquivo principal
 ├── 📂 requirements.txt        # Dependências
 ├── 📂 config/
 │   ├── german_standards.yaml   # Configurações normas alemãs
 │   └── web_sources.yaml       # Fontes web de normas
 ├── 📂 src/
 │   ├── 📂 ui/              # Interface PyQt6
 │   │   ├── main_window.py      # Janela principal alemã
 │   │   └── standards_browser.py # Navegador normas
```

```
│   ├── 📂 core/            # Engine TGA alemão
│   │   ├── tga_german_engine.py # Motor alemão
│   │   └── standards_manager.py # Gerenciador normas
│   ├── 📂 apis/            # APIs CAD/BIM
│   │   ├── autocad_real.py     # AutoCAD COM
│   │   ├── revit_real.py       # Revit API
│   │   └── oda_reader.py       # ODA SDK
│   ├── 📂 generators/        # Geradores TGA alemães
│   │   ├── german_architectural.py   # AR alemão
│   │   ├── german_structural.py      # ST alemão
│   │   ├── german_hydraulic.py       # HY alemão
│   │   ├── german_electrical.py      # EL alemão
│   │   ├── german_hvac.py            # HV alemão
│   │   ├── german_fire.py            # FP alemão
│   │   └── german_automation.py      # BA alemão
│   ├── 📂 standards/         # Normas alemãs online
│   │   ├── din_scraper.py      # Scraper DIN
│   │   ├── vdi_scraper.py      # Scraper VDI
│   │   ├── vob_scraper.py      # Scraper VOB
│   │   ├── dibt_scraper.py     # Scraper DIBt
│   │   └── standards_cache.py  # Cache local
│   └── 📂 ai_local/          # IA Local alemã
│       ├── ollama_german.py    # LLM alemão
```

```
│   └── german_prompts.py   # Prompts alemães
└── 📂 data/
 ├── 📂 german_templates/    # Templates alemães
 ├── 📂 german_components/   # Componentes alemães
 └── 📂 standards_cache/     # Cache normas
"""

# ==================================================================
# 1. ARQUIVO PRINCIPAL ALEMÃO - main.py

# ==================================================================

import sys
import os
import asyncio
from pathlib import Path

# Adicionar src ao path

sys.path.insert(0, str(Path(**file**).parent / "src"))

from PyQt6.QtWidgets import QApplication, QMessageBox, QSplashScreen
from PyQt6.QtCore import Qt, QTimer
from PyQt6.QtGui import QPixmap, QFont
```

from src.ui.main_window import TGAGermanMainWindow

from src.core.standards_manager import GermanStandardsManager

from src.utils.logger import setup_logger

class TGAGermanApp(QApplication):

“”“Aplicação TGA focada em normas alemãs”””

```

def __init__(self, argv):

    super().__init__(argv)

    self.setApplicationName("TGA German Desktop Automation")

    self.setApplicationVersion("1.0.0")

    self.setOrganizationName("TGA German Systems")

    # Configurar fonte alemã

    font = QFont("Segoe UI", 9)

    self.setFont(font)

    # Setup logging

    self.logger = setup_logger()

    self.logger.info("=== TGA German Desktop Automation gestartet ===")

def show_splash(self):

    """Mostrar splash screen"""

    splash_text = """

    TGA GERMAN DESKTOP AUTOMATION
```

```
    Technische Gebäudeausrüstung nach deutschen Normen
    DIN • VDI • VOB • DIBt

    Laden der Anwendung...
    """

    # Criar splash screen simples
    splash = QSplashScreen()
    splash.showMessage(splash_text, Qt.AlignmentFlag.AlignCenter)
    splash.show()

    # Timer para fechar splash
    QTimer.singleShot(3000, splash.close)
    return splash
```

```
def main():
"""Função principal"""
app = TGAGermanApp(sys.argv)

```

```
# Mostrar splash
splash = app.show_splash()

# Verificar dependências
missing_deps = check_german_dependencies()
if missing_deps:
    QMessageBox.critical(
```

```
                self.selected_standards.append(standard_number)

            # Mostrar detalhes da norma
            asyncio.create_task(self.show_standard_details(standard_number))
        else:
            standard_number = item.text(0)
            if standard_number in self.selected_standards:
                self.selected_standards.remove(standard_number)

    async def show_standard_details(self, standard_number):
        """Mostrar detalhes da norma selecionada"""
        try:
            standard = await
self.standards_manager.get_standard_details(standard_number)
            if standard:
                details_text = f"""
                <h3>{standard.number}: {standard.title}</h3>
                <p><b>Typ:</b> {standard.type}</p>
                <p><b>Kategorie:</b> {standard.category}</p>
                <p><b>Inhalt:</b></p>
                <p>{standard.content[:500]}...</p>
                """
                self.standard_details.setHtml(details_text)
        except Exception as e:
            self.logger.warning(f"Fehler beim Laden von Norm-Details: {e}")

    def start_german_processing(self):
        """Iniciar processamento alemão"""
```

```
    # Validações alemãs
    if not self.dwg_path_edit.text():
        QMessageBox.warning(self, "Fehler", "Bitte wählen Sie zuerst eine DWG-Datei
aus!")
        return

    if not Path(self.dwg_path_edit.text()).exists():
        QMessageBox.warning(self, "Fehler", "DWG-Datei nicht gefunden!")
        return

    # Verificar disciplinas selecionadas
    selected_disciplines = [
        code for code, checkbox in self.discipline_checks.items()
        if checkbox.isChecked()
    ]

    if not selected_disciplines:
        QMessageBox.warning(self, "Fehler", "Bitte wählen Sie mindestens ein Gewerk
aus!")
        return

    # Configurar projeto alemão
    german_config = {
        'dwg_path': self.dwg_path_edit.text(),
        'project_type': self.project_type.currentText(),
        'total_area': self.total_area.value(),
        'floors_count': self.floors_count.value(),
        'german_state': self.german_state.currentText(),
        'disciplines': selected_disciplines,
```

```
        'selected_standards': self.selected_standards,
        'language': 'german'
    }

    # Iniciar thread alemã
    self.processing_thread = GermanProcessingThread(self.tga_engine,
german_config)
    self.processing_thread.progress_updated.connect(self.update_progress)
    self.processing_thread.status_updated.connect(self.update_status)
    self.processing_thread.log_updated.connect(self.log_message)
    self.processing_thread.finished.connect(self.processing_finished)

    # UI updates alemãs
    self.process_btn.setEnabled(False)
    self.progress_bar.setVisible(True)
    self.progress_bar.setValue(0)

    self.log_message("=== STARTE TGA-VERARBEITUNG NACH DEUTSCHEN
NORMEN ===")
    self.processing_thread.start()

def update_progress(self, value):
    """Atualizar progresso"""
    self.progress_bar.setValue(value)

def update_status(self, status):
    """Atualizar status alemão"""
    self.status_label.setText(status)
```

```
def log_message(self, message):
    """Log em alemão"""
    from datetime import datetime
    timestamp = datetime.now().strftime("%H:%M:%S")
    formatted_message = f"[{timestamp}] {message}"

    self.log_text.append(formatted_message)
    self.log_text.ensureCursorVisible()

    self.logger.info(message)

def processing_finished(self, success, result_data):
    """Processamento alemão finalizado"""
    self.process_btn.setEnabled(True)
    self.progress_bar.setVisible(False)

    if success:
        self.status_label.setText("✅ Verarbeitung erfolgreich abgeschlossen!")
        self.status_label.setStyleSheet("color: green; padding: 5px; background-color: #d5f4e6; border-radius: 4px;")

        # Atualizar resultados
        self.update_results_display(result_data)

        # Habilitar botões
        self.open_folder_btn.setEnabled(True)
        self.open_autocad_btn.setEnabled(True)
        self.open_revit_btn.setEnabled(True)
```

```
        self.log_message("=== VERARBEITUNG ERFOLGREICH ABGESCHLOSSEN ===")

        QMessageBox.information(
            self, "Erfolg",
            f"TGA-Projekt erfolgreich nach deutschen Normen erstellt!\\n\\nDateien gespeichert in:\\n{result_data.get('output_path', 'N/A')}"
        )
    else:
        self.status_label.setText("❌ Fehler bei der Verarbeitung")
        self.status_label.setStyleSheet("color: red; padding: 5px; background-color: #f8d7da; border-radius: 4px;")

        error_msg = result_data.get('error', 'Unbekannter Fehler')
        self.log_message(f"FEHLER: {error_msg}")

        QMessageBox.critical(self, "Fehler", f"Verarbeitung fehlgeschlagen:\\n{error_msg}")

def update_results_display(self, result_data):
    """Atualizar exibição alemã"""
    self.results_list.clear()

    if 'files' in result_data:
        for file_info in result_data['files']:
            self.results_list.addItem(f"{file_info['type']}: {file_info['name']}")

    if 'compliance_report' in result_data:
```

```
        self.compliance_text.clear()
        compliance = result_data['compliance_report']

        self.compliance_text.append("=== DEUTSCHE NORMEN COMPLIANCE ===\\n")

        for standard, status in compliance.items():
            icon = "✅" if status['compliant'] else "❌"
            self.compliance_text.append(f"{icon} {standard}: {status['message']}")

    self.current_project = result_data

def open_results_folder(self):
    """Abrir pasta alemã"""
    if self.current_project and 'output_path' in self.current_project:
        import os
        os.startfile(self.current_project['output_path'])

def open_in_autocad(self):
    """Abrir no AutoCAD alemão"""
    if self.current_project and 'autocad_files' in self.current_project:
        try:
            for file_path in self.current_project['autocad_files']:
                self.autocad.open_drawing(file_path)
            self.log_message("Dateien in AutoCAD geöffnet")
        except Exception as e:
            QMessageBox.warning(self, "Fehler", f"Fehler beim Öffnen in AutoCAD:\\n{e}")
```

```
def open_in_revit(self):
    """Abrir no Revit alemão"""
    if self.current_project and 'revit_files' in self.current_project:
        try:
            for file_path in self.current_project['revit_files']:
                self.revit.open_model(file_path)
            self.log_message("Modelle in Revit geöffnet")
        except Exception as e:
            QMessageBox.warning(self, "Fehler", f"Fehler beim Öffnen in Revit:\\n{e}")
```

```
class GermanProcessingThread(QThread):
“”“Thread alemã para processamento”””
progress_updated = pyqtSignal(int)
status_updated = pyqtSignal(str)
log_updated = pyqtSignal(str)
finished = pyqtSignal(bool, dict)

```

```
def __init__(self, tga_engine, german_config):
    super().__init__()
    self.tga_engine = tga_engine
    self.german_config = german_config

def run(self):
    """Executar processamento alemão"""
    try:
        loop = asyncio.new_event_loop()
```

```
        asyncio.set_event_loop(loop)

        result = loop.run_until_complete(
            self.tga_engine.process_german_project_complete(
                self.german_config,
                progress_callback=self.progress_updated.emit,
                status_callback=self.status_updated.emit,
                log_callback=self.log_updated.emit
            )
        )

        self.finished.emit(True, result)

    except Exception as e:
        self.log_updated.emit(f"FEHLER: {str(e)}")
        self.finished.emit(False, {"error": str(e)})
```

```
# ==================================================================
# 7. ENGINE TGA ALEMÃO - src/core/tga_german_engine.py

# ==================================================================

import asyncio
```

```
from pathlib import Path
from datetime import datetime
from typing import Dict, List, Callable, Optional

from src.core.standards_manager import GermanStandardsManager
from src.apis.autocad_real import AutoCADController
from src.apis.revit_real import RevitController
from src.apis.oda_reader import ODAController
from src.generators.german_architectural import GermanArchitecturalGenerator
from src.generators.german_structural import GermanStructuralGenerator
from src.generators.german_hydraulic import GermanHydraulicGenerator
from src.generators.german_electrical import GermanElectricalGenerator
from src.generators.german_hvac import GermanHVACGenerator
from src.generators.german_fire import GermanFireGenerator
from src.generators.german_automation import GermanAutomationGenerator
from src.ai_local.ollama_german import GermanOllamaInterface
from src.utils.logger import get_logger

class TGAGermanEngine:
"""Engine TGA especializado em normas alemãs"""

```

```
def __init__(self, standards_manager: GermanStandardsManager):
    self.logger = get_logger(__name__)
    self.standards_manager = standards_manager

    # Controllers
    self.autocad = AutoCADController()
```

```
    self.revit = RevitController()
    self.oda = ODAController()

    # IA alemã
    self.german_ai = GermanOllamaInterface()

    # Geradores alemães por disciplina
    self.german_generators = {
        'AR': GermanArchitecturalGenerator(standards_manager),
        'ST': GermanStructuralGenerator(standards_manager),
        'HZ': GermanHydraulicGenerator(standards_manager),  # Heizung
        'LU': GermanHydraulicGenerator(standards_manager),  # Lüftung
        'SA': GermanHydraulicGenerator(standards_manager),  # Sanitär
        'EL': GermanElectricalGenerator(standards_manager),
        'BS': GermanFireGenerator(standards_manager),      # Brandschutz
        'GA': GermanAutomationGenerator(standards_manager)  # Gebäudeautomation
    }

async def process_german_project_complete(
    self,
    german_config: Dict,
    progress_callback: Callable[[int], None] = None,
    status_callback: Callable[[str], None] = None,
    log_callback: Callable[[str], None] = None
) -> Dict:
    """Processar projeto completo com normas alemãs"""
```

```
def update_progress(value: int):
    if progress_callback:
        progress_callback(value)

def update_status(status: str):
    if status_callback:
        status_callback(status)
    if log_callback:
        log_callback(f"[{datetime.now().strftime('%H:%M:%S')}] {status}")

try:
    update_status("🔍 Analysiere DWG-Datei...")
    update_progress(10)

    # 1. Análise do DWG
    project_data = await self.analyze_german_dwg(german_config, update_status)
    update_progress(20)

    # 2. Buscar normas alemãs relevantes
    update_status("📋 Lade relevante deutsche Normen...")
    relevant_standards = await self.load_relevant_german_standards(
        german_config, project_data, update_status
    )
    update_progress(30)

    # 3. Gerar projetos 2D por disciplina alemã
    update_status("📐 Erstelle 2D-Pläne nach deutschen Normen...")
    german_designs_2d = await self.generate_german_2d_designs(
```

```
    project_data, german_config, relevant_standards, update_status
)
update_progress(65)

# 4. Criar modelo BIM alemão
update_status("🏗️ Erstelle BIM-Modell...")
german_model_3d = await self.generate_german_3d_bim(
    german_designs_2d, german_config, relevant_standards, update_status
)
update_progress(85)

# 5. Validação compliance alemão
update_status("✅ Prüfe deutsche Normen-Compliance...")
compliance_report = await self.validate_german_compliance(
    german_designs_2d, german_model_3d, relevant_standards, update_status
)
update_progress(95)

# 6. Salvar resultados alemães
update_status("💾 Speichere deutsche TGA-Dateien...")
output_path = await self.save_german_results(
    german_designs_2d, german_model_3d, compliance_report, german_config
)
update_progress(100)

update_status("🎉 Deutsche TGA-Projekt erfolgreich erstellt!")

return {
```

```
        'success': True,
        'designs_2d': german_designs_2d,
        'model_3d': german_model_3d,
        'compliance_report': compliance_report,
        'output_path': output_path,
        'standards_used': relevant_standards,
        'files': self.get_generated_files_list(output_path),
        'autocad_files': self.get_autocad_files(output_path),
        'revit_files': self.get_revit_files(output_path)
    }

  except Exception as e:
    update_status(f"❌ Fehler bei der Verarbeitung: {str(e)}")
    raise e

async def analyze_german_dwg(self, german_config: Dict, status_callback):
  """Analisar DWG com foco alemão"""
  status_callback("Extrahiere Entitäten...")

  # Usar ODA SDK para extrair entidades
  entities = await self.oda.extract_entities(german_config['dwg_path'])

  status_callback("Erkenne deutsche Baunormen...")

  # Detectar elementos com IA alemã
  ai_analysis = await self.german_ai.analyze_german_drawing(
    entities, german_config
  )
```

```
    status_callback("Extrahiere Texte und Maße...")

    # OCR alemão
    text_data = await self.oda.extract_german_text_ocr(german_config['dwg_path'])

    return {
        'entities': entities,
        'ai_analysis': ai_analysis,
        'text_data': text_data,
        'bounds': await self.oda.get_drawing_bounds(german_config['dwg_path']),
        'german_specific': await self.extract_german_building_data(entities, text_data)
    }

async def extract_german_building_data(self, entities, text_data):
    """Extrair dados específicos de construção alemã"""
    german_data = {
        'building_type': None,
        'energy_standard': None,  # PassivHaus, KfW, etc.
        'fire_class': None,
        'accessibility': False,
        'federal_state_specific': {}
    }

    # Analisar textos para padrões alemães
    text_content = ' '.join([t.get('text', '') for t in text_data])

    # Detectar padrão energético
```

```
    energy_patterns = ['PassivHaus', 'KfW 40', 'KfW 55', 'EnEV', 'GEG']
    for pattern in energy_patterns:
        if pattern.lower() in text_content.lower():
            german_data['energy_standard'] = pattern
            break

    # Detectar classe de incêndio
    fire_patterns = ['F30', 'F60', 'F90', 'F120', 'F180']
    for pattern in fire_patterns:
        if pattern in text_content:
            german_data['fire_class'] = pattern
            break

    return german_data

async def load_relevant_german_standards(self, german_config, project_data,
status_callback):
    """Carregar normas alemãs relevantes"""
    relevant_standards = {}

    # Normas por disciplina
    discipline_standards = {
        'AR': ['DIN 18599', 'DIN 277'],
        'ST': ['DIN EN 1992', 'DIN 1045'],
        'HZ': ['DIN EN 12831', 'VDI 2078'],
        'LU': ['DIN EN 16798', 'VDI 2052'],
        'SA': ['DIN EN 806', 'VDI 6003'],
        'EL': ['DIN VDE 0100', 'VDI 3834'],
```

```
        'BS': ['DIN 14010', 'VDI 6019'],
        'GA': ['DIN EN 15232', 'VDI 3814']
    }

    for discipline in german_config['disciplines']:
        status_callback(f"Lade {discipline}-Normen...")

        standards_list = discipline_standards.get(discipline, [])
        discipline_standards_data = []

        for standard_number in standards_list:
            standard = await
self.standards_manager.get_standard_details(standard_number)
            if standard:
                discipline_standards_data.append(standard)

        # Buscar normas adicionais baseadas no tipo de projeto
        additional_standards = await
self.standards_manager.get_standards_for_discipline(discipline)
        discipline_standards_data.extend(additional_standards[:3])  # Top 3

        relevant_standards[discipline] = discipline_standards_data

    # Normas específicas do estado alemão
    state_specific = await
self.get_state_specific_standards(german_config['german_state'])
    relevant_standards['state_specific'] = state_specific

    return relevant_standards
```

```
async def get_state_specific_standards(self, german_state):
    """Obter normas específicas do estado alemão"""
    # Regulamentações específicas por estado
    state_regulations = {
        'Bayern': ['BayBO', 'BayLBO'],
        'Baden-Württemberg': ['LBO BW'],
        'Nordrhein-Westfalen': ['BauO NRW'],
        'Berlin': ['BauO Bln'],
        # ... outros estados
    }

    regulations = state_regulations.get(german_state, [])
    state_standards = []

    for regulation in regulations:
        standard = await self.standards_manager.get_standard_details(regulation)
        if standard:
            state_standards.append(standard)

    return state_standards

async def generate_german_2d_designs(self, project_data, german_config,
standards, status_callback):
    """Gerar projetos 2D alemães"""
    german_designs = {}

    for discipline in german_config['disciplines']:
```

```
        status_callback(f"Erstelle {discipline}-Pläne...")

        generator = self.german_generators.get(discipline)
        if not generator:
            continue

        # Gerar design alemão
        design = await generator.generate_german_2d_design(
            base_data=project_data,
            config=german_config,
            standards=standards.get(discipline, []),
            previous_disciplines=german_designs
        )

        german_designs[discipline] = design

    return german_designs

async def generate_german_3d_bim(self, designs_2d, german_config, standards,
status_callback):
    """Gerar modelo BIM alemão"""
    status_callback("Erstelle zentrales BIM-Modell...")

    # Criar modelo central alemão
    central_model = await self.revit.create_german_central_model(german_config)

    status_callback("Importiere deutsche Gewerke...")
```

```
# Importar disciplinas alemãs
discipline_models = {}
for discipline, design in designs_2d.items():
    model = await self.revit.import_german_discipline_to_bim(
        design, discipline, central_model, standards
    )
    discipline_models[discipline] = model

status_callback("Kollisionsprüfung...")

# Clash detection alemão
clash_results = await self.revit.run_german_clash_detection(
    central_model, discipline_models
)

status_callback("IFC-Export...")

# Export IFC alemão
ifc_model = await self.revit.export_german_ifc(central_model)

return {
    'central_model': central_model,
    'discipline_models': discipline_models,
    'clash_results': clash_results,
    'ifc_model': ifc_model
}
```

```
async def validate_german_compliance(self, designs_2d, model_3d, standards,
status_callback):
    """Validar compliance com normas alemãs"""
    compliance_report = {}

    # Validar cada disciplina
    for discipline, design in designs_2d.items():
        status_callback(f"Prüfe {discipline}-Compliance...")

        discipline_standards = standards.get(discipline, [])
        discipline_compliance = {}

        for standard in discipline_standards:
            compliance_result = await self.validate_against_german_standard(
                design, standard, model_3d
            )
            discipline_compliance[standard.number] = compliance_result

        compliance_report[discipline] = discipline_compliance

    # Validação geral alemã
    status_callback("Prüfe allgemeine deutsche Bauvorschriften...")
    general_compliance = await self.validate_general_german_building_code(
        designs_2d, model_3d
    )
    compliance_report['general'] = general_compliance

    return compliance_report
```

```
async def validate_against_german_standard(self, design, standard, model_3d):
    """Validar contra norma alemã específica"""
    # Usar IA alemã para validação
    validation_result = await self.german_ai.validate_design_against_standard(
        design, standard
    )

    return {
        'compliant': validation_result.get('compliant', False),
        'score': validation_result.get('score', 0.0),
        'violations': validation_result.get('violations', []),
        'recommendations': validation_result.get('recommendations', []),
        'message': validation_result.get('message', 'Geprüft')
    }

async def validate_general_german_building_code(self, designs_2d, model_3d):
    """Validar código geral alemão"""
    general_checks = {
        'accessibility': 'DIN 18040 Barrierefreiheit',
        'fire_safety': 'Bauordnung Brandschutz',
        'energy_efficiency': 'GEG Energieeffizienz',
        'structural_safety': 'Eurocode Tragwerksplanung'
    }

    results = {}
    for check_type, description in general_checks.items():
        # Implementar validações específicas
```

```
        result = await self.perform_general_check(check_type, designs_2d, model_3d)
        results[check_type] = {
            'description': description,
            'compliant': result.get('compliant', True),
            'message': result.get('message', 'Konform')
        }

    return results

async def perform_general_check(self, check_type, designs_2d, model_3d):
    """Realizar verificação geral alemã"""
    # Implementação simplificada
    return {
        'compliant': True,
        'message': f'{check_type.title()} erfüllt deutsche Normen'
    }

async def save_german_results(self, designs_2d, model_3d, compliance_report,
german_config):
    """Salvar resultados alemães"""
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    project_name = Path(german_config['dwg_path']).stem
    output_dir = Path("output") / f"TGA_German_{project_name}_{timestamp}"
    output_dir.mkdir(parents=True, exist_ok=True)

    # Salvar desenhos 2D alemães
    dwg_2d_dir = output_dir / "2D_Plaene_Deutsch"
    dwg_2d_dir.mkdir(exist_ok=True)
```

```
german_discipline_names = {
    'AR': 'Architektur',
    'ST': 'Tragwerk',
    'HZ': 'Heizung',
    'LU': 'Lueftung',
    'SA': 'Sanitaer',
    'EL': 'Elektro',
    'BS': 'Brandschutz',
    'GA': 'Gebaeudeautomation'
}

for discipline, design in designs_2d.items():
    german_name = german_discipline_names.get(discipline, discipline)
    dwg_path = dwg_2d_dir / f"{discipline}_{german_name}.dwg"
    await self.autocad.save_german_drawing(design, dwg_path, german_config)

# Salvar modelos 3D alemães
bim_3d_dir = output_dir / "3D_BIM_Modelle"
bim_3d_dir.mkdir(exist_ok=True)

await self.revit.save_german_models(model_3d, bim_3d_dir, german_config)

# Salvar relatórios alemães
reports_dir = output_dir / "Deutsche_Berichte"
reports_dir.mkdir(exist_ok=True)

await self.save_german_compliance_reports(compliance_report, reports_dir)
```

```
    return str(output_dir)

async def save_german_compliance_reports(self, compliance_report,
reports_dir):
    """Salvar relatórios de compliance alemães"""
    # Relatório HTML alemão
    html_content = self.generate_german_html_report(compliance_report)

    html_path = reports_dir / "Deutsche_Normen_Compliance.html"
    with open(html_path, 'w', encoding='utf-8') as f:
        f.write(html_content)

    # Relatório PDF alemão
    pdf_path = reports_dir / "Deutsche_Normen_Compliance.pdf"
    await self.generate_german_pdf_report(compliance_report, pdf_path)

def generate_german_html_report(self, compliance_report):
    """Gerar relatório HTML alemão"""
    html = """
    <!DOCTYPE html>
    <html lang="de">
    <head>
        <meta charset="UTF-8">
        <title>Deutsche Normen Compliance Bericht</title>
        <style>
            body { font-family: Arial, sans-serif; margin: 20px; }
            .header { background: linear-gradient(90deg, #000, #ff0000, #ffd700);
```

```
            color: white; padding: 20px; text-align: center; }
        .compliant { color: green; }
        .non-compliant { color: red; }
        .section { margin: 20px 0; padding: 15px; border: 1px solid #ddd; }
    </style>
</head>
<body>
    <div class="header">
        <h1>DE DEUTSCHE NORMEN COMPLIANCE BERICHT</h1>
        <h2>TGA nach deutschen Vorschriften</h2>
    </div>
"""

for discipline, standards_compliance in compliance_report.items():
    if discipline == 'general':
        continue

    html += f"""
    <div class="section">
        <h3>{discipline} - Compliance</h3>
    """

    for standard_number, compliance in standards_compliance.items():
        status_class = "compliant" if compliance['compliant'] else "non-compliant"
        icon = "✅" if compliance['compliant'] else "❌"

        html += f"""
        <p class="{status_class}">
```

```
            {icon} <strong>{standard_number}</strong>: {compliance['message']}
        </p>
        """

    html += "</div>"

html += """
</body>
</html>
"""

return html

async def generate_german_pdf_report(self, compliance_report, pdf_path):
    """Gerar relatório PDF alemão"""
    # Implementação simplificada - usar reportlab
    from reportlab.pdfgen import canvas
    from reportlab.lib.pagesizes import A4

    c = canvas.Canvas(str(pdf_path), pagesize=A4)

    # Título alemão
    c.setFont("Helvetica-Bold", 16)
    c.drawString(50, 800, "Deutsche Normen Compliance Bericht")

    y_position = 750
    c.setFont("Helvetica", 12)
```

```
    for discipline, standards_compliance in compliance_report.items():
        if discipline == 'general':
            continue

        c.drawString(50, y_position, f"{discipline} - Compliance:")
        y_position -= 20

        for standard_number, compliance in standards_compliance.items():
            icon = "✓" if compliance['compliant'] else "✗"
            text = f"  {icon} {standard_number}: {compliance['message']}"
            c.drawString(70, y_position, text)
            y_position -= 15

            if, "Fehlende Abhängigkeiten",
        f"Folgende Module fehlen:\\n{', '.join(missing_deps)}\\n\\nInstallieren Sie mit: pip install -r requirements.txt"
    )
    sys.exit(1)

try:
    # Inicializar gerenciador de normas alemãs
    standards_manager = GermanStandardsManager()

    # Criar janela principal
    main_window = TGAGermanMainWindow(standards_manager)

    # Fechar splash e mostrar janela
    splash.finish(main_window)
```

```
    main_window.show()

    # Executar aplicação
    sys.exit(app.exec())

except Exception as e:
    QMessageBox.critical(
        None, "Fataler Fehler",
        f"Fehler beim Initialisieren der Anwendung:\\n{e}"
    )
    sys.exit(1)
```

```
def check_german_dependencies():
"""Verificar dependências específicas do sistema alemão"""
required_modules = [
'PyQt6', 'win32com.client', 'requests', 'beautifulsoup4',
'lxml', 'yaml', 'sqlite3', 'asyncio', 'aiohttp'
]

```

```
missing = []
for module in required_modules:
    try:
        __import__(module)
    except ImportError:
        missing.append(module)
```

return missing

```

if **name** == “**main**”:

main()

#
==========

# 2. REQUIREMENTS.txt ALEMÃO

#
==========

“”” 

PyQt6>=6.6.0

pywin32>=306

pyautocad>=0.2.0

requests>=2.31.0

PyYAML>=6.0

beautifulsoup4>=4.12.0

lxml>=4.9.0

selenium>=4.15.0

aiohttp>=3.8.0

asyncio>=3.4.3

Pillow>=10.0.0

opencv-python>=4.8.0
```

```
ultralytics>=8.0.0
pytesseract>=0.3.10
numpy>=1.24.0
pandas>=2.0.0
matplotlib>=3.7.0
reportlab>=4.0.0
"""

# ==================================================================
# 3. GERENCIADOR DE NORMAS ALEMÃS - src/core/standards_manager.py

# ==================================================================

import asyncio
import aiohttp
import sqlite3
import yaml
import json
from pathlib import Path
from datetime import datetime, timedelta
from typing import Dict, List, Optional
from dataclasses import dataclass

from src.standards.din_scraper import DINScraper
```

```
from src.standards.vdi_scraper import VDIScraper
from src.standards.vob_scraper import VOBScraper
from src.standards.dibt_scraper import DIBtScraper
from src.utils.logger import get_logger

@dataclass
class GermanStandard:
"""Classe para representar uma norma alemã"""
number: str
title: str
type: str  # DIN, VDI, VOB, DIBt
category: str  # TGA, Structural, Fire, etc.
content: str
url: str
last_updated: datetime
relevance_score: float = 0.0

class GermanStandardsManager:
"""Gerenciador de normas alemãs com busca online"""

```
def __init__(self):
    self.logger = get_logger(__name__)
    self.cache_db_path = Path("data/standards_cache/german_standards.db")
    self.cache_db_path.parent.mkdir(parents=True, exist_ok=True)

    # Scrapers especializados
    self.din_scraper = DINScraper()
```
```

```
    self.vdi_scraper = VDIScraper()
    self.vob_scraper = VOBScraper()
    self.dibt_scraper = DIBtScraper()

    # Cache em memória
    self.standards_cache: Dict[str, GermanStandard] = {}

    # Inicializar banco de dados
    self.init_cache_database()

    # Carregar configurações
    self.load_german_config()

def init_cache_database(self):
    """Inicializar banco de dados de cache"""
    conn = sqlite3.connect(self.cache_db_path)
    cursor = conn.cursor()

    cursor.execute("""
        CREATE TABLE IF NOT EXISTS german_standards (
            number TEXT PRIMARY KEY,
            title TEXT NOT NULL,
            type TEXT NOT NULL,
            category TEXT NOT NULL,
            content TEXT NOT NULL,
            url TEXT,
            last_updated TIMESTAMP,
            relevance_score REAL DEFAULT 0.0
```

```
        )
    """)

    cursor.execute("""
        CREATE TABLE IF NOT EXISTS search_history (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            query TEXT NOT NULL,
            results_count INTEGER,
            timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
        )
    """)

    conn.commit()
    conn.close()

    self.logger.info("Datenbank für deutsche Normen initialisiert")

def load_german_config(self):
    """Carregar configurações alemãs"""
    config_path = Path("config/german_standards.yaml")
    if config_path.exists():
        with open(config_path, 'r', encoding='utf-8') as f:
            self.config = yaml.safe_load(f)
    else:
        # Configuração padrão
        self.config = {
            'tga_categories': {
                'Heizung': ['DIN EN 12831', 'VDI 2078', 'VDI 6030'],
```

```
            'Lüftung': ['DIN EN 16798', 'VDI 2052', 'VDI 3803'],
            'Sanitär': ['DIN EN 806', 'VDI 6003', 'VDI 6023'],
            'Elektro': ['DIN VDE 0100', 'VDI 3834', 'VDI 3814'],
            'Brandschutz': ['DIN 14010', 'VDI 6019', 'MLAR'],
            'Gebäudeautomation': ['DIN EN 15232', 'VDI 3814', 'VDI 3813']
        },
        'priority_standards': [
            'DIN 18599', 'VDI 2552', 'VOB/C', 'EnEV', 'GEG'
        ],
        'update_frequency_hours': 24,
        'max_cache_age_days': 30
    }

async def search_standards_online(self, query: str, category: str = None) ->
List[GermanStandard]:
    """Buscar normas alemãs online"""
    self.logger.info(f"Suche nach deutschen Normen: '{query}' Kategorie: {category}")

    results = []

    # Buscar em paralelo nos diferentes scrapers
    tasks = [
        self.din_scraper.search(query, category),
        self.vdi_scraper.search(query, category),
        self.vob_scraper.search(query, category),
        self.dibt_scraper.search(query, category)
    ]
```

```
    try:
        scraper_results = await asyncio.gather(*tasks, return_exceptions=True)

        for i, result in enumerate(scraper_results):
            if isinstance(result, Exception):
                self.logger.warning(f"Fehler bei Scraper {i}: {result}")
                continue

            if isinstance(result, list):
                results.extend(result)

    except Exception as e:
        self.logger.error(f"Fehler bei der Online-Suche: {e}")

    # Filtrar e ranquear resultados
    filtered_results = self.filter_and_rank_results(results, query, category)

    # Salvar no cache
    await self.cache_standards(filtered_results)

    # Salvar histórico de busca
    self.save_search_history(query, len(filtered_results))

    return filtered_results

def filter_and_rank_results(self, results: List[GermanStandard], query: str,
category: str) -> List[GermanStandard]:
    """Filtrar e ranquear resultados por relevância"""
```

```
if not results:
    return []

# Calcular score de relevância
for standard in results:
    score = 0.0

    # Score baseado na query
    if query.lower() in standard.title.lower():
        score += 2.0
    if query.lower() in standard.number.lower():
        score += 3.0
    if query.lower() in standard.content.lower():
        score += 1.0

    # Score baseado na categoria
    if category and category.lower() in standard.category.lower():
        score += 2.0

    # Score baseado em prioridade
    if standard.number in self.config.get('priority_standards', []):
        score += 5.0

    # Score baseado na atualidade
    if standard.last_updated:
        days_old = (datetime.now() - standard.last_updated).days
        if days_old < 30:
            score += 1.0
```

```
        elif days_old < 365:
            score += 0.5

    standard.relevance_score = score

# Ordenar por relevância
results.sort(key=lambda x: x.relevance_score, reverse=True)

# Remover duplicatas
seen = set()
unique_results = []
for standard in results:
    if standard.number not in seen:
        seen.add(standard.number)
        unique_results.append(standard)

return unique_results[:20]  # Top 20 resultados

async def cache_standards(self, standards: List[GermanStandard]):
    """Salvar normas no cache local"""
    conn = sqlite3.connect(self.cache_db_path)
    cursor = conn.cursor()

    for standard in standards:
        cursor.execute("""
            INSERT OR REPLACE INTO german_standards
            (number, title, type, category, content, url, last_updated, relevance_score)
            VALUES (?, ?, ?, ?, ?, ?, ?, ?)
```

```
        """, (
            standard.number,
            standard.title,
            standard.type,
            standard.category,
            standard.content,
            standard.url,
            standard.last_updated,
            standard.relevance_score
        ))

        # Adicionar ao cache em memória
        self.standards_cache[standard.number] = standard

    conn.commit()
    conn.close()

    self.logger.info(f"{len(standards)} Normen im Cache gespeichert")

def save_search_history(self, query: str, results_count: int):
    """Salvar histórico de busca"""
    conn = sqlite3.connect(self.cache_db_path)
    cursor = conn.cursor()

    cursor.execute("""
        INSERT INTO search_history (query, results_count)
        VALUES (?, ?)
    """, (query, results_count))
```

```
    conn.commit()
    conn.close()

async def get_standards_for_discipline(self, discipline: str) ->
List[GermanStandard]:
    """Obter normas relevantes para uma disciplina específica"""
    discipline_mapping = {
        'AR': 'Architektur',
        'ST': 'Tragwerk',
        'HY': 'Sanitär',
        'EL': 'Elektro',
        'HV': 'Heizung Lüftung',
        'FP': 'Brandschutz',
        'BA': 'Gebäudeautomation'
    }

    category = discipline_mapping.get(discipline, discipline)

    # Primeiro verificar cache
    cached_standards = self.get_cached_standards_by_category(category)

    if cached_standards and
self.is_cache_valid(cached_standards[0].last_updated):
        self.logger.info(f"Verwende gecachte Normen für {discipline}")
        return cached_standards

    # Buscar online se cache inválido
    self.logger.info(f"Suche online nach Normen für {discipline}")
```

```
    return await self.search_standards_online(category, category)

def get_cached_standards_by_category(self, category: str) ->
List[GermanStandard]:
    """Obter normas do cache por categoria"""
    conn = sqlite3.connect(self.cache_db_path)
    cursor = conn.cursor()

    cursor.execute("""
        SELECT * FROM german_standards
        WHERE category LIKE ?
        ORDER BY relevance_score DESC
    """, (f"%{category}%",))

    rows = cursor.fetchall()
    conn.close()

    standards = []
    for row in rows:
        standard = GermanStandard(
            number=row[0],
            title=row[1],
            type=row[2],
            category=row[3],
            content=row[4],
            url=row[5],
            last_updated=datetime.fromisoformat(row[6]) if row[6] else None,
            relevance_score=row[7]
```

```
        )
        standards.append(standard)

    return standards

def is_cache_valid(self, last_updated: datetime) -> bool:
    """Verificar se cache ainda é válido"""
    if not last_updated:
        return False

    max_age = timedelta(hours=self.config.get('update_frequency_hours', 24))
    return datetime.now() - last_updated < max_age

async def get_standard_details(self, standard_number: str) ->
Optional[GermanStandard]:
    """Obter detalhes completos de uma norma específica"""
    # Verificar cache primeiro
    if standard_number in self.standards_cache:
        return self.standards_cache[standard_number]

    # Buscar no banco
    conn = sqlite3.connect(self.cache_db_path)
    cursor = conn.cursor()

    cursor.execute("""
        SELECT * FROM german_standards WHERE number = ?
    """, (standard_number,))
```

```
    row = cursor.fetchone()
    conn.close()

    if row:
        return GermanStandard(
            number=row[0],
            title=row[1],
            type=row[2],
            category=row[3],
            content=row[4],
            url=row[5],
            last_updated=datetime.fromisoformat(row[6]) if row[6] else None,
            relevance_score=row[7]
        )

    # Se não encontrou, tentar buscar online
    search_results = await self.search_standards_online(standard_number)
    if search_results:
        return search_results[0]

    return None

def get_search_suggestions(self, partial_query: str) -> List[str]:
    """Obter sugestões de busca baseadas no histórico"""
    conn = sqlite3.connect(self.cache_db_path)
    cursor = conn.cursor()

    cursor.execute("""
```

```
        SELECT DISTINCT query FROM search_history
        WHERE query LIKE ?
        ORDER BY timestamp DESC
        LIMIT 10
    """, (f"%{partial_query}%",))

    suggestions = [row[0] for row in cursor.fetchall()]
    conn.close()

    return suggestions
```

```
# ==================================================================
# 4. SCRAPER DIN - src/standards/din_scraper.py

# ==================================================================

import aiohttp
import asyncio
from bs4 import BeautifulSoup
from datetime import datetime
from typing import List, Optional
from urllib.parse import urljoin, quote
```

from src.core.standards_manager import GermanStandard

from src.utils.logger import get_logger

class DINScraper:

“”“Scraper para normas DIN”””

```

def __init__(self):

    self.logger = get_logger(__name__)

    self.base_url = "https://www.din.de"

    self.search_url = "https://www.din.de/de/suche"

    # Headers para parecer um browser real

    self.headers = {

        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36',

        'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8',

        'Accept-Language': 'de,en;q=0.9',

        'Accept-Encoding': 'gzip, deflate',

        'Connection': 'keep-alive',

        'Upgrade-Insecure-Requests': '1',

    }

async def search(self, query: str, category: str = None) -> List[GermanStandard]:

    """Buscar normas DIN"""

    self.logger.info(f"Suche DIN-Normen für: {query}")

    try:
```

```
        async with aiohttp.ClientSession(headers=self.headers) as session:

            # Buscar VDI específicas para TGA
            for vdi_number in vdi_tga_numbers:
                if query.lower() in vdi_number or vdi_number in query:
                    standard = await self.get_vdi_standard(vdi_number, session)
                    if standard:
                        standards.append(standard)

            # Busca geral no site VDI
            general_results = await self.search_vdi_general(query, session)
            standards.extend(general_results)

    except Exception as e:
        self.logger.error(f"Fehler bei VDI-Scraping: {e}")
        return []

    return standards[:10]  # Top 10

async def get_vdi_standard(self, vdi_number: str, session: aiohttp.ClientSession) -
> GermanStandard:
    """Obter detalhes de uma VDI específica"""
    try:
        # URLs conhecidas para VDI importantes
        vdi_urls = {
            '2052': '/vdi-2052-lueftungstechnische-massnahmen',
            '2078': '/vdi-2078-kuehllastberechnung',
            '2166': '/vdi-2166-planung-lueftungsanlagen',
```

```
            '2552': '/vdi-2552-building-information-modeling',
            '3803': '/vdi-3803-zentrale-raumlufttechnik',
            '3814': '/vdi-3814-gebaeudeautomation',
            '6003': '/vdi-6003-trinkwasser-installationen',
            '6019': '/vdi-6019-ingenieurmethoden-brandschutz',
            '6023': '/vdi-6023-hygiene-trinkwasser-installationen'
        }

        if vdi_number not in vdi_urls:
            return None

        url = f"{self.base_url}{vdi_urls[vdi_number]}"

        async with session.get(url) as response:
            if response.status == 200:
                html = await response.text()
                return self.parse_vdi_page(html, vdi_number, url)

    except Exception as e:
        self.logger.warning(f"Fehler beim Laden von VDI {vdi_number}: {e}")

    return None

async def search_vdi_general(self, query: str, session: aiohttp.ClientSession) ->
List[GermanStandard]:
    """Busca geral no site VDI"""
    standards = []
```

```
    try:
        # Simular busca baseada em conhecimento das VDI principais
        vdi_knowledge = {
            'heizung': ['VDI 2078', 'VDI 6030'],
            'lüftung': ['VDI 2052', 'VDI 3803'],
            'sanitär': ['VDI 6003', 'VDI 6023'],
            'elektro': ['VDI 3834'],
            'automation': ['VDI 3814', 'VDI 3813'],
            'bim': ['VDI 2552'],
            'brandschutz': ['VDI 6019']
        }

        query_lower = query.lower()
        for topic, vdi_list in vdi_knowledge.items():
            if topic in query_lower:
                for vdi_num in vdi_list:
                    number = vdi_num.replace('VDI ', '')
                    standard = await self.get_vdi_standard(number, session)
                    if standard:
                        standards.append(standard)

    except Exception as e:
        self.logger.warning(f"Fehler bei VDI-Generalsuche: {e}")

    return standards

def parse_vdi_page(self, html: str, vdi_number: str, url: str) -> GermanStandard:
    """Parse de página VDI específica"""
```

```
soup = BeautifulSoup(html, 'html.parser')

# Extrair título
title_selectors = ['h1', '.page-title', '.vdi-title']
title = f"VDI {vdi_number}"

for selector in title_selectors:
    title_elem = soup.select_one(selector)
    if title_elem:
        title = title_elem.get_text(strip=True)
        break

# Extrair conteúdo
content_selectors = ['.vdi-content', '.main-content', 'article', '.content']
content = ""

for selector in content_selectors:
    content_elem = soup.select_one(selector)
    if content_elem:
        content = content_elem.get_text(separator=' ', strip=True)[:2000]
        break

# Categorizar baseado no número VDI
category = self.categorize_vdi(vdi_number, title, content)

return GermanStandard(
    number=f"VDI {vdi_number}",
    title=title,
```

```
        type="VDI",
        category=category,
        content=content,
        url=url,
        last_updated=datetime.now()
    )

def categorize_vdi(self, vdi_number: str, title: str, content: str) -> str:
    """Categorizar VDI baseada no número e conteúdo"""
    vdi_categories = {
        '2052': 'Lüftung',
        '2078': 'Kühlung',
        '2166': 'Lüftung',
        '2552': 'BIM',
        '3803': 'Lüftung',
        '3814': 'Gebäudeautomation',
        '6003': 'Sanitär',
        '6019': 'Brandschutz',
        '6023': 'Sanitär',
        '6030': 'Heizung'
    }

    return vdi_categories.get(vdi_number, 'TGA')
```

#
==========

6. INTERFACE PRINCIPAL ALEMÃ - src/ui/main_window.py

```
# ==================================================================
==========

import asyncio
from pathlib import Path
from PyQt6.QtWidgets import (
QMainWindow, QVBoxLayout, QHBoxLayout, QWidget, QPushButton,
QLabel, QFileDialog, QTextEdit, QProgressBar, QComboBox,
QSpinBox, QGroupBox, QGridLayout, QMessageBox, QSplitter,
QListWidget, QTabWidget, QCheckBox, QLineEdit, QCompleter,
QTreeWidget, QTreeWidgetItem
)
from PyQt6.QtCore import QThread, pyqtSignal, Qt, QTimer, QStringListModel
from PyQt6.QtGui import QFont, QIcon

from src.core.tga_german_engine import TGAGermanEngine
from src.core.standards_manager import GermanStandardsManager
from src.apis.autocad_real import AutoCADController
from src.apis.revit_real import RevitController
from src.utils.logger import get_logger

class TGAGermanMainWindow(QMainWindow):
"""Janela principal TGA com foco em normas alemãs"""

```

```
def __init__(self, standards_manager: GermanStandardsManager):
```

```
    super().__init__()
    self.standards_manager = standards_manager
    self.logger = get_logger(__name__)
    self.tga_engine = TGAGermanEngine(standards_manager)
    self.autocad = AutoCADController()
    self.revit = RevitController()

    # Estado da aplicação
    self.current_project = None
    self.processing_thread = None
    self.selected_standards = []

    self.setup_ui()
    self.setup_connections()
    self.check_software_status()

def setup_ui(self):
    """Configurar interface alemã"""
    self.setWindowTitle("TGA German Desktop Automation v1.0 - Deutsche
Normen")
    self.setGeometry(100, 100, 1600, 1000)

    # Widget central
    central_widget = QWidget()
    self.setCentralWidget(central_widget)

    # Layout principal
    main_layout = QHBoxLayout()
```

```
    central_widget.setLayout(main_layout)

    # Splitter triplo
    splitter = QSplitter(Qt.Orientation.Horizontal)
    main_layout.addWidget(splitter)

    # Painel esquerdo - Configuração
    left_panel = self.create_left_panel()
    splitter.addWidget(left_panel)

    # Painel central - Normas alemãs
    center_panel = self.create_standards_panel()
    splitter.addWidget(center_panel)

    # Painel direito - Resultados
    right_panel = self.create_right_panel()
    splitter.addWidget(right_panel)

    # Proporções
    splitter.setSizes([400, 600, 600])

    # Status bar alemã
    self.statusBar().showMessage("Bereit für TGA-Projekte nach deutschen
Normen")

def create_left_panel(self):
    """Painel de configuração alemão"""
    widget = QWidget()
```

```
    layout = QVBoxLayout()
    widget.setLayout(layout)

    # Header alemão
    header_label = QLabel("TGA DEUTSCHE AUTOMATION")
    header_label.setFont(QFont("Arial", 16, QFont.Weight.Bold))
    header_label.setStyleSheet("color: #000000; background-color: #FFD700;
padding: 10px; border: 2px solid #FF0000;")
    header_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
    layout.addWidget(header_label)

    # Status conexões
    self.create_german_status_section(layout)

    # Configuração projeto alemão
    self.create_german_project_config(layout)

    # Disciplinas TGA alemãs
    self.create_german_disciplines_section(layout)

    # Botão processar alemão
    self.create_german_process_section(layout)

    layout.addStretch()
    return widget

def create_german_status_section(self, layout):
    """Seção status alemã"""
```

```
    status_group = QGroupBox("Software-Status")
    status_layout = QGridLayout()

    self.autocad_status = QLabel("❌ AutoCAD: Nicht verbunden")
    self.revit_status = QLabel("❌ Revit: Nicht verbunden")
    self.standards_status = QLabel("❌ Deutsche Normen: Laden...")

    status_layout.addWidget(self.autocad_status, 0, 0)
    status_layout.addWidget(self.revit_status, 1, 0)
    status_layout.addWidget(self.standards_status, 2, 0)

    refresh_btn = QPushButton("🔄 Status Prüfen")
    refresh_btn.clicked.connect(self.check_software_status)
    status_layout.addWidget(refresh_btn, 3, 0)

    status_group.setLayout(status_layout)
    layout.addWidget(status_group)

def create_german_project_config(self, layout):
    """Configuração projeto alemão"""
    config_group = QGroupBox("Projektkonfiguration")
    config_layout = QGridLayout()

    # Arquivo DWG
    config_layout.addWidget(QLabel("DWG-Datei:"), 0, 0)
    self.dwg_path_edit = QLineEdit()
    self.dwg_path_edit.setPlaceholderText("Basis-DWG-Datei auswählen...")
    config_layout.addWidget(self.dwg_path_edit, 0, 1)
```

```
self.dwg_browse_btn = QPushButton("📂 Durchsuchen")
self.dwg_browse_btn.clicked.connect(self.select_dwg_file)
config_layout.addWidget(self.dwg_browse_btn, 0, 2)

# Tipo de projeto alemão
config_layout.addWidget(QLabel("Gebäudetyp:"), 1, 0)
self.project_type = QComboBox()
self.project_type.addItems([
    "Einfamilienhaus",
    "Mehrfamilienhaus",
    "Bürogebäude",
    "Einzelhandel",
    "Industriebau",
    "Bildungseinrichtung",
    "Gesundheitswesen",
    "Hotel/Gastronomie"
])
config_layout.addWidget(self.project_type, 1, 1, 1, 2)

# Área alemã
config_layout.addWidget(QLabel("Nutzfläche:"), 2, 0)
self.total_area = QSpinBox()
self.total_area.setRange(20, 100000)
self.total_area.setValue(200)
self.total_area.setSuffix(" m²")
config_layout.addWidget(self.total_area, 2, 1)
```

```
    # Pavimentos
    config_layout.addWidget(QLabel("Geschosse:"), 2, 2)
    self.floors_count = QSpinBox()
    self.floors_count.setRange(1, 50)
    self.floors_count.setValue(1)
    config_layout.addWidget(self.floors_count, 2, 3)

    # Região alemã
    config_layout.addWidget(QLabel("Bundesland:"), 3, 0)
    self.german_state = QComboBox()
    self.german_state.addItems([
        "Baden-Württemberg", "Bayern", "Berlin", "Brandenburg",
        "Bremen", "Hamburg", "Hessen", "Mecklenburg-Vorpommern",
        "Niedersachsen", "Nordrhein-Westfalen", "Rheinland-Pfalz",
        "Saarland", "Sachsen", "Sachsen-Anhalt", "Schleswig-Holstein", "Thüringen"
    ])
    config_layout.addWidget(self.german_state, 3, 1, 1, 2)

    config_group.setLayout(config_layout)
    layout.addWidget(config_group)

def create_german_disciplines_section(self, layout):
    """Disciplinas TGA alemãs"""
    disciplines_group = QGroupBox("TGA-Gewerke")
    disciplines_layout = QGridLayout()

    self.discipline_checks = {}
    german_disciplines = [
```

```
        ("AR", "Architektur", True),
        ("ST", "Tragwerk", True),
        ("HZ", "Heizung", True),
        ("LU", "Lüftung", True),
        ("SA", "Sanitär", True),
        ("EL", "Elektrotechnik", True),
        ("BS", "Brandschutz", True),
        ("GA", "Gebäudeautomation", False)
    ]

    for i, (code, name, checked) in enumerate(german_disciplines):
        checkbox = QCheckBox(f"{code} - {name}")
        checkbox.setChecked(checked)
        self.discipline_checks[code] = checkbox
        disciplines_layout.addWidget(checkbox, i // 2, i % 2)

    disciplines_group.setLayout(disciplines_layout)
    layout.addWidget(disciplines_group)

def create_german_process_section(self, layout):
    """Seção processamento alemão"""
    process_group = QGroupBox("TGA-Verarbeitung")
    process_layout = QVBoxLayout()

    # Botão principal alemão
    self.process_btn = QPushButton("🚀 TGA-PROJEKT NACH DEUTSCHEN NORMEN ERSTELLEN")
    self.process_btn.setFont(QFont("Arial", 11, QFont.Weight.Bold))
```

```
self.process_btn.setStyleSheet("""
    QPushButton {
        background-color: #000000;
        color: #FFD700;
        border: 2px solid #FF0000;
        padding: 15px;
        border-radius: 8px;
        min-height: 40px;
    }
    QPushButton:hover {
        background-color: #333333;
    }
    QPushButton:disabled {
        background-color: #666666;
    }
""")
self.process_btn.clicked.connect(self.start_german_processing)
process_layout.addWidget(self.process_btn)

# Barra de progresso
self.progress_bar = QProgressBar()
self.progress_bar.setVisible(False)
process_layout.addWidget(self.progress_bar)

# Status alemão
self.status_label = QLabel("Bereit zur Verarbeitung")
self.status_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
```

```
    self.status_label.setStyleSheet("padding: 5px; background-color: #f0f0f0; border-radius: 4px;")
    process_layout.addWidget(self.status_label)

    process_group.setLayout(process_layout)
    layout.addWidget(process_group)

def create_standards_panel(self):
    """Painel de normas alemãs"""
    widget = QWidget()
    layout = QVBoxLayout()
    widget.setLayout(layout)

    # Header normas
    standards_label = QLabel("DEUTSCHE NORMEN & VORSCHRIFTEN")
    standards_label.setFont(QFont("Arial", 14, QFont.Weight.Bold))
    standards_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
    standards_label.setStyleSheet("background-color: #f0f0f0; padding: 8px; border-radius: 4px;")
    layout.addWidget(standards_label)

    # Busca de normas
    search_layout = QHBoxLayout()

    self.standards_search = QLineEdit()
    self.standards_search.setPlaceholderText("DIN, VDI, VOB oder Stichwort suchen...")
    self.standards_search.textChanged.connect(self.on_standards_search)
```

```
# Autocompletar
self.setup_standards_completer()

search_btn = QPushButton("🔍 Suchen")
search_btn.clicked.connect(self.search_standards)

search_layout.addWidget(self.standards_search)
search_layout.addWidget(search_btn)
layout.addLayout(search_layout)

# Árvore de normas
self.standards_tree = QTreeWidget()
self.standards_tree.setHeaderLabels(["Norm", "Titel", "Kategorie"])
self.standards_tree.itemChanged.connect(self.on_standard_selected)
layout.addWidget(self.standards_tree)

# Detalhes da norma selecionada
details_group = QGroupBox("Norm-Details")
details_layout = QVBoxLayout()

self.standard_details = QTextEdit()
self.standard_details.setMaximumHeight(150)
self.standard_details.setReadOnly(True)
details_layout.addWidget(self.standard_details)

details_group.setLayout(details_layout)
layout.addWidget(details_group)
```

```
    # Carregar normas padrão
    self.load_default_standards()

    return widget

def setup_standards_completer(self):
    """Configurar autocompletar para normas"""
    common_standards = [
        "DIN 18599", "DIN EN 12831", "DIN EN 16798",
        "VDI 2052", "VDI 2078", "VDI 2552", "VDI 3814",
        "VOB/A", "VOB/B", "VOB/C",
        "Heizung", "Lüftung", "Sanitär", "Elektro"
    ]

    completer = QCompleter(common_standards)
    completer.setFilterMode(Qt.MatchFlag.MatchContains)
    self.standards_search.setCompleter(completer)

def load_default_standards(self):
    """Carregar normas alemãs padrão"""
    default_categories = {
        "DIN Normen": [
            ("DIN 18599", "Energetische Bewertung von Gebäuden", "Energie"),
            ("DIN EN 12831", "Heizungsanlagen - Verfahren zur Berechnung der Norm-Heizlast", "Heizung"),
            ("DIN EN 16798", "Energetische Bewertung von Gebäuden - Lüftung", "Lüftung")
        ],
        "VDI Richtlinien": [
```

```
        ("VDI 2052", "Raumlufttechnische Anlagen für Küchen", "Lüftung"),
        ("VDI 2078", "Berechnung der Kühllast", "Kühlung"),
        ("VDI 2552", "Building Information Modeling", "BIM"),
        ("VDI 3814", "Gebäudeautomation", "Automation")
    ],
    "VOB": [
        ("VOB/A", "Allgemeine Bestimmungen für die Vergabe von Bauleistungen",
"Vergabe"),
        ("VOB/B", "Allgemeine Vertragsbedingungen", "Vertrag"),
        ("VOB/C", "Allgemeine Technische Vertragsbedingungen", "Technik")
    ]
  }

  for category, standards in default_categories.items():
    category_item = QTreeWidgetItem([category])
    category_item.setFlags(Qt.ItemFlag.ItemIsEnabled)

    for number, title, subcategory in standards:
        standard_item = QTreeWidgetItem([number, title, subcategory])
        standard_item.setFlags(Qt.ItemFlag.ItemIsEnabled |
Qt.ItemFlag.ItemIsUserCheckable)
        standard_item.setCheckState(0, Qt.CheckState.Unchecked)
        category_item.addChild(standard_item)

    self.standards_tree.addTopLevelItem(category_item)

  self.standards_tree.expandAll()

def create_right_panel(self):
```

```
"""Painel direito de resultados"""
tabs = QTabWidget()

# Tab 1: Log alemão
log_widget = QWidget()
log_layout = QVBoxLayout()

log_label = QLabel("Verarbeitungsprotokoll:")
log_label.setFont(QFont("Arial", 10, QFont.Weight.Bold))
log_layout.addWidget(log_label)

self.log_text = QTextEdit()
self.log_text.setReadOnly(True)
self.log_text.setFont(QFont("Consolas", 9))
log_layout.addWidget(self.log_text)

log_widget.setLayout(log_layout)
tabs.addTab(log_widget, "📋 Protokoll")

# Tab 2: Resultados alemães
results_widget = QWidget()
results_layout = QVBoxLayout()

results_label = QLabel("Generierte Dateien:")
results_label.setFont(QFont("Arial", 10, QFont.Weight.Bold))
results_layout.addWidget(results_label)

self.results_list = QListWidget()
```

```
results_layout.addWidget(self.results_list)

# Botões alemães
buttons_layout = QHBoxLayout()

self.open_folder_btn = QPushButton("📂 Ordner öffnen")
self.open_folder_btn.clicked.connect(self.open_results_folder)
self.open_folder_btn.setEnabled(False)

self.open_autocad_btn = QPushButton("📐 AutoCAD")
self.open_autocad_btn.clicked.connect(self.open_in_autocad)
self.open_autocad_btn.setEnabled(False)

self.open_revit_btn = QPushButton("🏗️ Revit")
self.open_revit_btn.clicked.connect(self.open_in_revit)
self.open_revit_btn.setEnabled(False)

buttons_layout.addWidget(self.open_folder_btn)
buttons_layout.addWidget(self.open_autocad_btn)
buttons_layout.addWidget(self.open_revit_btn)

results_layout.addLayout(buttons_layout)
results_widget.setLayout(results_layout)
tabs.addTab(results_widget, "📂 Ergebnisse")

# Tab 3: Compliance alemão
compliance_widget = QWidget()
compliance_layout = QVBoxLayout()
```

```
    compliance_label = QLabel("Normen-Compliance:")
    compliance_label.setFont(QFont("Arial", 10, QFont.Weight.Bold))
    compliance_layout.addWidget(compliance_label)

    self.compliance_text = QTextEdit()
    self.compliance_text.setReadOnly(True)
    compliance_layout.addWidget(self.compliance_text)

    compliance_widget.setLayout(compliance_layout)
    tabs.addTab(compliance_widget, "✅ Compliance")

    return tabs

def setup_connections(self):
    """Configurar conexões alemãs"""
    # Timer para verificação
    self.status_timer = QTimer()
    self.status_timer.timeout.connect(self.check_software_status)
    self.status_timer.start(30000)  # 30 segundos

def check_software_status(self):
    """Verificar status software alemão"""
    # AutoCAD
    try:
        if self.autocad.check_connection():
            self.autocad_status.setText("✅ AutoCAD: Verbunden")
            self.autocad_status.setStyleSheet("color: green;")
```

```
    else:
        self.autocad_status.setText("❌ AutoCAD: Nicht verbunden")
        self.autocad_status.setStyleSheet("color: red;")
except:
    self.autocad_status.setText("❌ AutoCAD: Fehler")
    self.autocad_status.setStyleSheet("color: red;")

# Revit
try:
    if self.revit.check_connection():
        self.revit_status.setText("✅ Revit: Verbunden")
        self.revit_status.setStyleSheet("color: green;")
    else:
        self.revit_status.setText("❌ Revit: Nicht verbunden")
        self.revit_status.setStyleSheet("color: red;")
except:
    self.revit_status.setText("❌ Revit: Fehler")
    self.revit_status.setStyleSheet("color: red;")

# Normas alemãs
try:
    self.standards_status.setText("✅ Deutsche Normen: Verfügbar")
    self.standards_status.setStyleSheet("color: green;")
except:
    self.standards_status.setText("❌ Deutsche Normen: Fehler")
    self.standards_status.setStyleSheet("color: red;")
```

```
def select_dwg_file(self):
    """Selecionar arquivo DWG alemão"""
    file_path, _ = QFileDialog.getOpenFileName(
        self,
        "DWG-Datei auswählen",
        "",
        "DWG-Dateien (*.dwg);;Alle Dateien (*)"
    )

    if file_path:
        self.dwg_path_edit.setText(file_path)
        self.log_message(f"Datei ausgewählt: {Path(file_path).name}")

def on_standards_search(self, text):
    """Busca de normas em tempo real"""
    if len(text) >= 3:
        # Buscar normas alemãs
        asyncio.create_task(self.search_standards_async(text))

async def search_standards_async(self, query):
    """Busca assíncrona de normas"""
    try:
        results = await self.standards_manager.search_standards_online(query)
        self.update_standards_tree(results)
    except Exception as e:
        self.logger.warning(f"Fehler bei Normen-Suche: {e}")

def search_standards(self):
```

```
    """Buscar normas alemãs"""
    query = self.standards_search.text().strip()
    if not query:
        return

    self.log_message(f"Suche deutsche Normen für: {query}")

    # Executar busca assíncrona
    asyncio.create_task(self.search_standards_async(query))

def update_standards_tree(self, standards):
    """Atualizar árvore de normas"""
    # Limpar resultados anteriores
    search_items = []
    for i in range(self.standards_tree.topLevelItemCount()):
        item = self.standards_tree.topLevelItem(i)
        if item.text(0) == "Suchergebnisse":
            search_items.append(item)

    for item in search_items:
        self.standards_tree.takeTopLevelItem(
            self.standards_tree.indexOfTopLevelItem(item)
        )

    if not standards:
        return

    # Adicionar novos resultados
```

```
    search_category = QTreeWidgetItem(["Suchergebnisse"])

    for standard in standards:
        standard_item = QTreeWidgetItem([
            standard.number,
            standard.title,
            standard.category
        ])
        standard_item.setFlags(Qt.ItemFlag.ItemIsEnabled |
Qt.ItemFlag.ItemIsUserCheckable)
        standard_item.setCheckState(0, Qt.CheckState.Unchecked)
        search_category.addChild(standard_item)

    self.standards_tree.insertTopLevelItem(0, search_category)
    self.standards_tree.expandItem(search_category)

def on_standard_selected(self, item, column):
    """Norma selecionada"""
    if item.checkState(0) == Qt.CheckState.Checked:
        standard_number = item.text(0)
        if standard_number not in self.selected_standards:headers=self.headers) as
session:
            # Construir URL de busca
            search_params = {
                'query': query,
                'type': 'norm',
                'lang': 'de'
            }
```

```
        if category:
            search_params['category'] = category

        # Fazer busca
        async with session.get(self.search_url, params=search_params) as response:
            if response.status == 200:
                html = await response.text()
                return await self.parse_search_results(html, session)
            else:
                self.logger.warning(f"DIN-Suche fehlgeschlagen: Status {response.status}")
                return []

    except Exception as e:
        self.logger.error(f"Fehler bei DIN-Scraping: {e}")
        return []

async def parse_search_results(self, html: str, session: aiohttp.ClientSession) -> List[GermanStandard]:
    """Parse dos resultados de busca DIN"""
    soup = BeautifulSoup(html, 'html.parser')
    standards = []

    # Procurar resultados (adaptar seletores conforme site atual)
    result_items = soup.find_all('div', class_='search-result-item')

    for item in result_items[:10]:  # Limitar a 10 resultados
        try:
```

```
# Extrair informações básicas
title_elem = item.find('h3') or item.find('a')
if not title_elem:
    continue

title = title_elem.get_text(strip=True)
link = title_elem.get('href', '')

if link and not link.startswith('http'):
    link = urljoin(self.base_url, link)

# Extrair número da norma
number = self.extract_din_number(title)
if not number:
    continue

# Buscar conteúdo detalhado
content = await self.get_standard_content(link, session) if link else ""

# Criar objeto standard
standard = GermanStandard(
    number=number,
    title=title,
    type="DIN",
    category=self.categorize_din_standard(title, content),
    content=content,
    url=link,
    last_updated=datetime.now()
```

```
            )

            standards.append(standard)

            # Delay para evitar sobrecarga
            await asyncio.sleep(0.5)

        except Exception as e:
            self.logger.warning(f"Fehler beim Parsen von DIN-Ergebnis: {e}")
            continue

    self.logger.info(f"{len(standards)} DIN-Normen gefunden")
    return standards

def extract_din_number(self, title: str) -> Optional[str]:
    """Extrair número da norma DIN do título"""
    import re

    # Padrões para números DIN
    patterns = [
        r'DIN\s+(\d+(?:-\d+)*)',
        r'DIN\s+EN\s+(\d+(?:-\d+)*)',
        r'DIN\s+ISO\s+(\d+(?:-\d+)*)',
        r'DIN\s+VDE\s+(\d+(?:-\d+)*)'
    ]

    for pattern in patterns:
        match = re.search(pattern, title, re.IGNORECASE)
```

```
        if match:
            return f"DIN {match.group(1)}"

    return None

async def get_standard_content(self, url: str, session: aiohttp.ClientSession) -> str:
    """Obter conteúdo detalhado da norma"""
    try:
        async with session.get(url) as response:
            if response.status == 200:
                html = await response.text()
                soup = BeautifulSoup(html, 'html.parser')

                # Extrair texto principal (adaptar seletores)
                content_selectors = [
                    '.standard-content',
                    '.norm-content',
                    '.main-content',
                    'article',
                    '.content'
                ]

                for selector in content_selectors:
                    content_elem = soup.select_one(selector)
                    if content_elem:
                        # Limpar HTML e retornar texto
                        text = content_elem.get_text(separator=' ', strip=True)
                        return text[:2000]  # Limitar tamanho
```

```
            # Fallback: texto de todo o body
            body = soup.find('body')
            if body:
                return body.get_text(separator=' ', strip=True)[:1000]

    except Exception as e:
        self.logger.warning(f"Fehler beim Laden von DIN-Inhalt: {e}")

    return ""

def categorize_din_standard(self, title: str, content: str) -> str:
    """Categorizar norma DIN baseada no título e conteúdo"""
    text = f"{title} {content}".lower()

    categories = {
        'TGA': ['heizung', 'lüftung', 'sanitär', 'haustechnik', 'gebäudetechnik'],
        'Brandschutz': ['brandschutz', 'feuerschutz', 'rauchmelder', 'sprinkler'],
        'Elektro': ['elektro', 'elektrisch', 'installation', 'verkabelung'],
        'Tragwerk': ['tragwerk', 'statik', 'beton', 'stahl', 'holzbau'],
        'Energie': ['energie', 'dämmung', 'wärme', 'effizienz', 'verbrauch'],
        'BIM': ['bim', 'cad', 'digital', 'modell', 'daten']
    }

    for category, keywords in categories.items():
        if any(keyword in text for keyword in keywords):
            return category
```

```
    return 'Allgemein'
```

```
# ==================================================================
===========

# 5. SCRAPER VDI - src/standards/vdi_scraper.py

# ==================================================================
===========

import aiohttp
from bs4 import BeautifulSoup
from datetime import datetime
from typing import List
import re

from src.core.standards_manager import GermanStandard
from src.utils.logger import get_logger

class VDIScraper:
"""Scraper para diretrizes VDI"""

def __init__(self):
    self.logger = get_logger(__name__)
    self.base_url = "https://www.vdi.de"
```

```
    self.search_url = "https://www.vdi.de/richtlinien"

    self.headers = {
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36',
        'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
        'Accept-Language': 'de,en;q=0.9',
    }

async def search(self, query: str, category: str = None) -> List[GermanStandard]:
    """Buscar diretrizes VDI"""
    self.logger.info(f"Suche VDI-Richtlinien für: {query}")

    try:
        # VDI tem estrutura específica - focar em TGA
        vdi_tga_numbers = [
            '2052', '2078', '2166', '2552', '3803', '3814', '6003', '6019', '6023'
        ]

        standards = []

        async with aiohttp.ClientSession(
```