"""

SportAI Suite - Enterprise Edition

Production-Ready Sports Facility Management Platform

Version: 3.0.0 Enterprise

Copyright (c) 2025 SportAI Technologies

"""

import os

import sys

import streamlit as st

import json

import hashlib

import secrets

import logging

import time

from datetime import datetime, timedelta

from typing import Dict, Any, Optional, List, Tuple

import uuid

import base64

from pathlib import Path

import re

# Configure logging

def setup\_logging():

"""Setup enterprise-grade logging"""

log\_dir = Path("logs")

log\_dir.mkdir(exist\_ok=True)

logging.basicConfig(

level=logging.INFO,

format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',

handlers=[

logging.FileHandler(log\_dir / f'sportai\_{datetime.now().strftime("%Y%m%d")}.log'),

logging.StreamHandler()

]

)

return logging.getLogger('SportAI')

logger = setup\_logging()

# Add current directory to Python path

BASE\_DIR = os.path.dirname(\_\_file\_\_)

if BASE\_DIR not in sys.path:

sys.path.insert(0, BASE\_DIR)

# ============= SECURITY MODULE =============

class SecurityManager:

"""Enterprise-grade security manager"""

def \_\_init\_\_(self):

self.salt = self.\_get\_or\_create\_salt()

self.pepper = "SportAI\_Secure\_2025" # Application-specific pepper

self.max\_login\_attempts = 5

self.lockout\_duration = 900 # 15 minutes in seconds

self.session\_timeout = 3600 # 1 hour in seconds

self.password\_requirements = {

'min\_length': 8,

'require\_uppercase': True,

'require\_lowercase': True,

'require\_digit': True,

'require\_special': True

}

def \_get\_or\_create\_salt(self) -> str:

"""Get or create application salt"""

salt\_file = Path(".salt")

if salt\_file.exists():

with open(salt\_file, 'r') as f:

return f.read().strip()

else:

salt = secrets.token\_hex(32)

with open(salt\_file, 'w') as f:

f.write(salt)

return salt

def hash\_password(self, password: str) -> str:

"""Hash password with salt and pepper"""

combined = f"{password}{self.salt}{self.pepper}"

return hashlib.pbkdf2\_hmac('sha256',

combined.encode(),

self.salt.encode(),

100000).hex()

def verify\_password(self, stored\_hash: str, provided\_password: str) -> bool:

"""Verify password against stored hash"""

return stored\_hash == self.hash\_password(provided\_password)

def validate\_password\_strength(self, password: str) -> Tuple[bool, List[str]]:

"""Validate password meets security requirements"""

errors = []

if len(password) < self.password\_requirements['min\_length']:

errors.append(f"Password must be at least {self.password\_requirements['min\_length']} characters")

if self.password\_requirements['require\_uppercase'] and not any(c.isupper() for c in password):

errors.append("Password must contain at least one uppercase letter")

if self.password\_requirements['require\_lowercase'] and not any(c.islower() for c in password):

errors.append("Password must contain at least one lowercase letter")

if self.password\_requirements['require\_digit'] and not any(c.isdigit() for c in password):

errors.append("Password must contain at least one digit")

if self.password\_requirements['require\_special'] and not re.search(r'[!@#$%^&\*(),.?":{}|<>]', password):

errors.append("Password must contain at least one special character")

return len(errors) == 0, errors

def generate\_session\_token(self) -> str:

"""Generate secure session token"""

return secrets.token\_urlsafe(32)

def create\_audit\_log(self, user\_email: str, action: str, details: str = ""):

"""Create audit log entry"""

audit\_dir = Path("audit\_logs")

audit\_dir.mkdir(exist\_ok=True)

log\_entry = {

'timestamp': datetime.now().isoformat(),

'user': user\_email,

'action': action,

'details': details,

'ip': 'N/A', # Would get actual IP in production

'session\_id': st.session\_state.get('session\_id', 'N/A')

}

with open(audit\_dir / f'audit\_{datetime.now().strftime("%Y%m")}.jsonl', 'a') as f:

f.write(json.dumps(log\_entry) + '\n')

# ============= CONFIGURATION MANAGER =============

class ConfigurationManager:

"""Manage facility-specific configurations"""

def \_\_init\_\_(self, facility\_id: str = None):

self.config\_dir = Path("configurations")

self.config\_dir.mkdir(exist\_ok=True)

self.facility\_id = facility\_id or self.\_get\_facility\_id()

self.config = self.\_load\_config()

def \_get\_facility\_id(self) -> str:

"""Get or create facility ID"""

id\_file = self.config\_dir / "facility\_id.txt"

if id\_file.exists():

return id\_file.read\_text().strip()

else:

facility\_id = str(uuid.uuid4())

id\_file.write\_text(facility\_id)

return facility\_id

def \_load\_config(self) -> Dict:

"""Load facility configuration"""

config\_file = self.config\_dir / f"{self.facility\_id}\_config.json"

if config\_file.exists():

with open(config\_file, 'r') as f:

return json.load(f)

else:

return self.\_create\_default\_config()

def \_create\_default\_config(self) -> Dict:

"""Create default configuration for new facility"""

config = {

'facility': {

'name': 'Sports Complex',

'type': 'multi-sport',

'timezone': 'America/Chicago',

'currency': 'USD',

'language': 'en'

},

'features': {

'ai\_modules': True,

'advanced\_analytics': True,

'multi\_facility': False,

'api\_access': True,

'white\_label': True

},

'limits': {

'max\_users': 100,

'max\_facilities': 5,

'max\_events\_per\_month': 1000,

'max\_members': 10000,

'storage\_gb': 100

},

'branding': {

'primary\_color': '#1E40AF',

'secondary\_color': '#3B82F6',

'logo\_url': None,

'facility\_name': 'Your Sports Complex'

},

'subscription': {

'tier': 'professional', # starter, professional, enterprise

'valid\_until': (datetime.now() + timedelta(days=30)).isoformat(),

'seats': 10

},

'integrations': {

'payment\_gateway': None,

'email\_provider': 'smtp',

'sms\_provider': None,

'calendar\_sync': True,

'accounting\_software': None

}

}

self.\_save\_config(config)

return config

def \_save\_config(self, config: Dict):

"""Save configuration"""

config\_file = self.config\_dir / f"{self.facility\_id}\_config.json"

with open(config\_file, 'w') as f:

json.dump(config, f, indent=2)

def update\_config(self, section: str, key: str, value: Any):

"""Update configuration value"""

if section in self.config and key in self.config[section]:

self.config[section][key] = value

self.\_save\_config(self.config)

logger.info(f"Configuration updated: {section}.{key} = {value}")

# ============= LICENSE MANAGER =============

class LicenseManager:

"""Manage software licensing"""

def \_\_init\_\_(self, config\_manager: ConfigurationManager):

self.config = config\_manager

self.license\_file = Path("license.key")

def validate\_license(self) -> Tuple[bool, str]:

"""Validate software license"""

if not self.license\_file.exists():

return False, "No license file found"

try:

with open(self.license\_file, 'r') as f:

license\_key = f.read().strip()

# In production, this would validate against a license server

# For now, check if subscription is valid

valid\_until = datetime.fromisoformat(

self.config.config['subscription']['valid\_until']

)

if datetime.now() > valid\_until:

return False, f"License expired on {valid\_until.strftime('%Y-%m-%d')}"

days\_remaining = (valid\_until - datetime.now()).days

if days\_remaining < 7:

return True, f"⚠️ License expires in {days\_remaining} days"

return True, f"License valid until {valid\_until.strftime('%Y-%m-%d')}"

except Exception as e:

logger.error(f"License validation error: {e}")

return False, "Invalid license"

def check\_feature\_access(self, feature: str) -> bool:

"""Check if feature is available in current subscription"""

tier = self.config.config['subscription']['tier']

feature\_matrix = {

'starter': ['basic\_management', 'reporting', 'scheduling'],

'professional': ['basic\_management', 'reporting', 'scheduling',

'ai\_modules', 'advanced\_analytics', 'api\_access'],

'enterprise': ['basic\_management', 'reporting', 'scheduling',

'ai\_modules', 'advanced\_analytics', 'api\_access',

'multi\_facility', 'white\_label', 'custom\_integrations']

}

return feature in feature\_matrix.get(tier, [])

# ============= DATABASE MANAGER =============

class DatabaseManager:

"""Manage application database"""

def \_\_init\_\_(self, facility\_id: str):

self.db\_dir = Path("database")

self.db\_dir.mkdir(exist\_ok=True)

self.facility\_id = facility\_id

self.users\_file = self.db\_dir / f"{facility\_id}\_users.json"

self.ensure\_database()

def ensure\_database(self):

"""Ensure database files exist"""

if not self.users\_file.exists():

self.create\_default\_users()

def create\_default\_users(self):

"""Create default admin user for new facility"""

security = SecurityManager()

# Generate secure default password

default\_password = secrets.token\_urlsafe(12)

users = {

"admin@facility.com": {

"password\_hash": security.hash\_password(default\_password),

"role": "admin",

"created\_at": datetime.now().isoformat(),

"last\_login": None,

"failed\_attempts": 0,

"locked\_until": None,

"must\_change\_password": True,

"two\_factor\_enabled": False,

"api\_key": secrets.token\_urlsafe(32),

"permissions": ["all"]

}

}

with open(self.users\_file, 'w') as f:

json.dump(users, f, indent=2)

# Save default password securely for initial setup

setup\_file = self.db\_dir / f"{self.facility\_id}\_setup.txt"

setup\_file.write\_text(

f"Initial Admin Credentials:\n"

f"Email: admin@facility.com\n"

f"Password: {default\_password}\n"

f"Please change this password immediately after first login.\n"

f"This file will be deleted after first successful login."

)

logger.info(f"Created default admin user for facility {self.facility\_id}")

def load\_users(self) -> Dict:

"""Load users from database"""

try:

with open(self.users\_file, 'r') as f:

return json.load(f)

except Exception as e:

logger.error(f"Error loading users: {e}")

return {}

def save\_users(self, users: Dict):

"""Save users to database"""

with open(self.users\_file, 'w') as f:

json.dump(users, f, indent=2)

def add\_user(self, email: str, password: str, role: str = "user",

permissions: List[str] = None) -> Tuple[bool, str]:

"""Add new user"""

security = SecurityManager()

users = self.load\_users()

if email in users:

return False, "User already exists"

# Validate password

valid, errors = security.validate\_password\_strength(password)

if not valid:

return False, "\n".join(errors)

users[email] = {

"password\_hash": security.hash\_password(password),

"role": role,

"created\_at": datetime.now().isoformat(),

"last\_login": None,

"failed\_attempts": 0,

"locked\_until": None,

"must\_change\_password": False,

"two\_factor\_enabled": False,

"api\_key": secrets.token\_urlsafe(32),

"permissions": permissions or self.\_get\_default\_permissions(role)

}

self.save\_users(users)

security.create\_audit\_log(email, "USER\_CREATED", f"Role: {role}")

return True, "User created successfully"

def \_get\_default\_permissions(self, role: str) -> List[str]:

"""Get default permissions for role"""

permissions = {

'admin': ['all'],

'manager': ['read', 'write', 'manage\_events', 'manage\_members',

'view\_reports', 'manage\_facilities'],

'staff': ['read', 'write', 'manage\_events', 'view\_reports'],

'user': ['read', 'create\_bookings', 'view\_own\_data']

}

return permissions.get(role, ['read'])

# ============= MODULE LOADER =============

class ModuleLoader:

"""Smart module loader with dependency management"""

def \_\_init\_\_(self):

self.modules = {}

self.module\_metadata = {}

self.load\_queue = []

self.failed\_modules = {}

def safe\_import(self, module\_name: str, retry: bool = True):

"""Safely import a module with retry logic"""

if module\_name in self.modules:

return self.modules[module\_name]

try:

module = \_\_import\_\_(module\_name)

self.modules[module\_name] = module

logger.debug(f"Successfully imported {module\_name}")

return module

except ImportError as e:

self.failed\_modules[module\_name] = str(e)

logger.warning(f"Failed to import {module\_name}: {e}")

return None

except Exception as e:

self.failed\_modules[module\_name] = str(e)

logger.error(f"Unexpected error importing {module\_name}: {e}")

return None

def get\_module\_health(self) -> Dict:

"""Get module loading health status"""

return {

'loaded': len(self.modules),

'failed': len(self.failed\_modules),

'total': len(self.modules) + len(self.failed\_modules),

'health\_score': len(self.modules) / max(1, len(self.modules) + len(self.failed\_modules))

}

# ============= MAIN APPLICATION =============

class SportAIEnterpriseApp:

"""SportAI Suite Enterprise Application"""

def \_\_init\_\_(self):

self.security = SecurityManager()

self.config = ConfigurationManager()

self.license = LicenseManager(self.config)

self.db = DatabaseManager(self.config.facility\_id)

self.module\_loader = ModuleLoader()

self.tools = self.build\_tools\_menu()

# Check license on startup

self.license\_valid, self.license\_message = self.license.validate\_license()

def build\_tools\_menu(self) -> Dict[str, Any]:

"""Build the tools menu from available modules"""

tools = {}

# Define tool categories with their corresponding modules

tool\_categories = {

# Core Management

"📊 Central Dashboard": 'central\_dashboard',

"🎯 Event Control Panel": 'event\_control\_panel',

"🏟️ Facility Master Tracker": 'facility\_master\_tracker',

"👥 Membership Dashboard": 'membership\_dashboard',

"💰 Sponsor Dashboard": 'sponsor\_dashboard',

# AI Tools (Professional & Enterprise only)

"🤖 AI Event Forecast": 'ai\_event\_forecast',

"🤖 AI Revenue Maximizer": 'ai\_revenue\_maximizer',

"🤖 AI Strategy Dashboard": 'ai\_strategy\_dashboard',

"🤖 AI Sponsor Finder": 'ai\_sponsor\_opportunity\_finder',

# Facility Management

"🚪 Facility Access Tracker": 'facility\_access\_tracker',

"⚠️ Facility Capacity Alerts": 'facility\_capacity\_alerts',

"📋 Facility Contract Monitor": 'facility\_contract\_monitor',

"🗺️ Facility Layout Map": 'facility\_layout\_map',

# Financial Tools

"🔥 Revenue Heatmap": 'revenue\_heatmap',

"📈 Revenue Projection": 'revenue\_projection\_simulator',

"💰 Dynamic Pricing": 'dynamic\_pricing\_tool',

# Reporting

"📊 Reports Generator": 'weekly\_report\_generator',

"📄 PDF Export Tool": 'pdf\_export\_tool',

# Communications

"📧 Email Manager": 'email\_notifications',

"📱 SMS Alerts": 'sms\_alert\_center',

}

# Filter based on subscription tier

tier = self.config.config['subscription']['tier']

for tool\_name, module\_key in tool\_categories.items():

# Check if tool is available in current tier

if tool\_name.startswith("🤖") and tier == 'starter':

continue # Skip AI tools for starter tier

module = self.module\_loader.safe\_import(module\_key)

if module:

tools[tool\_name] = module

return tools

def validate\_session(self) -> bool:

"""Validate current session"""

if 'user' not in st.session\_state:

return False

user = st.session\_state.user

# Check session timeout

if 'login\_time' in user:

elapsed = time.time() - user['login\_time']

if elapsed > self.security.session\_timeout:

self.security.create\_audit\_log(

user.get('email', 'unknown'),

'SESSION\_TIMEOUT'

)

st.session\_state.user = None

return False

# Validate session token

if 'session\_token' not in user:

return False

return True

def login(self):

"""Handle user login with enhanced security"""

st.sidebar.header('🔐 Secure Login')

# Show license status

if not self.license\_valid:

st.sidebar.error(f"⚠️ {self.license\_message}")

st.sidebar.info("Please contact support to renew your license.")

return

elif "expires in" in self.license\_message.lower():

st.sidebar.warning(self.license\_message)

email = st.sidebar.text\_input('Email', key='login\_email')

password = st.sidebar.text\_input('Password', type='password', key='login\_password')

col1, col2 = st.sidebar.columns(2)

with col1:

if st.button('🔑 Login', use\_container\_width=True):

if self.authenticate\_user(email, password):

st.rerun()

with col2:

if st.button('📝 Register', use\_container\_width=True):

st.session\_state.show\_registration = True

# Show setup instructions for new installations

setup\_file = self.db.db\_dir / f"{self.config.facility\_id}\_setup.txt"

if setup\_file.exists():

with st.sidebar.expander("🔧 Initial Setup"):

st.code(setup\_file.read\_text())

def authenticate\_user(self, email: str, password: str) -> bool:

"""Authenticate user with security checks"""

users = self.db.load\_users()

if email not in users:

st.sidebar.error('Invalid credentials')

self.security.create\_audit\_log(email, 'LOGIN\_FAILED', 'User not found')

return False

user = users[email]

# Check if account is locked

if user.get('locked\_until'):

locked\_until = datetime.fromisoformat(user['locked\_until'])

if datetime.now() < locked\_until:

remaining = (locked\_until - datetime.now()).seconds // 60

st.sidebar.error(f'Account locked. Try again in {remaining} minutes.')

return False

else:

# Unlock account

user['locked\_until'] = None

user['failed\_attempts'] = 0

# Verify password

if not self.security.verify\_password(user['password\_hash'], password):

# Increment failed attempts

user['failed\_attempts'] = user.get('failed\_attempts', 0) + 1

if user['failed\_attempts'] >= self.security.max\_login\_attempts:

# Lock account

user['locked\_until'] = (

datetime.now() + timedelta(seconds=self.security.lockout\_duration)

).isoformat()

st.sidebar.error('Too many failed attempts. Account locked for 15 minutes.')

self.security.create\_audit\_log(email, 'ACCOUNT\_LOCKED',

f'After {user["failed\_attempts"]} attempts')

else:

remaining = self.security.max\_login\_attempts - user['failed\_attempts']

st.sidebar.error(f'Invalid credentials. {remaining} attempts remaining.')

users[email] = user

self.db.save\_users(users)

self.security.create\_audit\_log(email, 'LOGIN\_FAILED', 'Invalid password')

return False

# Successful login

user['failed\_attempts'] = 0

user['last\_login'] = datetime.now().isoformat()

users[email] = user

self.db.save\_users(users)

# Create session

st.session\_state.user = {

'email': email,

'role': user['role'],

'permissions': user.get('permissions', []),

'login\_time': time.time(),

'session\_token': self.security.generate\_session\_token(),

'session\_id': str(uuid.uuid4()),

'must\_change\_password': user.get('must\_change\_password', False),

'api\_key': user.get('api\_key')

}

# Delete setup file after first admin login

if user['role'] == 'admin':

setup\_file = self.db.db\_dir / f"{self.config.facility\_id}\_setup.txt"

if setup\_file.exists():

setup\_file.unlink()

self.security.create\_audit\_log(email, 'LOGIN\_SUCCESS', f'Role: {user["role"]}')

st.sidebar.success('✅ Login successful!')

return True

def register\_user(self):

"""Handle new user registration"""

st.header('📝 User Registration')

with st.form('registration\_form'):

col1, col2 = st.columns(2)

with col1:

email = st.text\_input('Email Address\*')

password = st.text\_input('Password\*', type='password')

password\_confirm = st.text\_input('Confirm Password\*', type='password')

with col2:

first\_name = st.text\_input('First Name\*')

last\_name = st.text\_input('Last Name\*')

organization = st.text\_input('Organization')

# Password requirements

st.info("""

\*\*Password Requirements:\*\*

- At least 8 characters

- One uppercase letter

- One lowercase letter

- One digit

- One special character (!@#$%^&\*...)

""")

terms = st.checkbox('I agree to the Terms of Service and Privacy Policy')

submitted = st.form\_submit\_button('Create Account')

if submitted:

if not all([email, password, password\_confirm, first\_name, last\_name]):

st.error('Please fill in all required fields')

elif password != password\_confirm:

st.error('Passwords do not match')

elif not terms:

st.error('Please accept the terms and conditions')

else:

# Validate email format

if not re.match(r'^[\w\.-]+@[\w\.-]+\.\w+$', email):

st.error('Invalid email format')

else:

success, message = self.db.add\_user(email, password, 'user')

if success:

st.success(message)

st.info('Please login with your new credentials')

st.session\_state.show\_registration = False

time.sleep(2)

st.rerun()

else:

st.error(message)

def render\_admin\_panel(self):

"""Render admin control panel"""

st.header('🛠️ Admin Control Panel')

tabs = st.tabs(['Users', 'Configuration', 'License', 'System Health', 'Audit Logs'])

with tabs[0]: # Users

self.render\_user\_management()

with tabs[1]: # Configuration

self.render\_configuration()

with tabs[2]: # License

self.render\_license\_info()

with tabs[3]: # System Health

self.render\_system\_health()

with tabs[4]: # Audit Logs

self.render\_audit\_logs()

def render\_user\_management(self):

"""Render user management interface"""

st.subheader('👥 User Management')

users = self.db.load\_users()

# Add new user

with st.expander('➕ Add New User'):

with st.form('add\_user\_form'):

col1, col2 = st.columns(2)

with col1:

new\_email = st.text\_input('Email')

new\_password = st.text\_input('Password', type='password')

with col2:

new\_role = st.selectbox('Role', ['user', 'staff', 'manager', 'admin'])

if st.form\_submit\_button('Add User'):

success, message = self.db.add\_user(new\_email, new\_password, new\_role)

if success:

st.success(message)

st.rerun()

else:

st.error(message)

# Display users

st.subheader('Current Users')

user\_data = []

for email, info in users.items():

user\_data.append({

'Email': email,

'Role': info['role'],

'Created': info.get('created\_at', 'N/A')[:10],

'Last Login': info.get('last\_login', 'Never')[:10] if info.get('last\_login') else 'Never',

'Status': '🔒 Locked' if info.get('locked\_until') else '✅ Active',

'2FA': '✅' if info.get('two\_factor\_enabled') else '❌'

})

if user\_data:

st.dataframe(user\_data, use\_container\_width=True)

def render\_configuration(self):

"""Render configuration interface"""

st.subheader('⚙️ Facility Configuration')

config = self.config.config

# Facility Info

with st.expander('🏢 Facility Information'):

facility\_name = st.text\_input('Facility Name',

value=config['facility']['name'])

facility\_type = st.selectbox('Facility Type',

['multi-sport', 'single-sport', 'recreation-center', 'stadium'],

index=['multi-sport', 'single-sport', 'recreation-center', 'stadium'].index(

config['facility'].get('type', 'multi-sport')))

if st.button('Save Facility Info'):

self.config.update\_config('facility', 'name', facility\_name)

self.config.update\_config('facility', 'type', facility\_type)

st.success('Facility information updated')

# Features

with st.expander('🎯 Features & Modules'):

col1, col2 = st.columns(2)

with col1:

ai\_enabled = st.checkbox('AI Modules',

value=config['features']['ai\_modules'])

analytics = st.checkbox('Advanced Analytics',

value=config['features']['advanced\_analytics'])

with col2:

api\_access = st.checkbox('API Access',

value=config['features']['api\_access'])

white\_label = st.checkbox('White Label',

value=config['features']['white\_label'])

if st.button('Save Features'):

self.config.update\_config('features', 'ai\_modules', ai\_enabled)

self.config.update\_config('features', 'advanced\_analytics', analytics)

self.config.update\_config('features', 'api\_access', api\_access)

self.config.update\_config('features', 'white\_label', white\_label)

st.success('Features updated')

# Branding

with st.expander('🎨 Branding'):

primary\_color = st.color\_picker('Primary Color',

value=config['branding']['primary\_color'])

secondary\_color = st.color\_picker('Secondary Color',

value=config['branding']['secondary\_color'])

if st.button('Save Branding'):

self.config.update\_config('branding', 'primary\_color', primary\_color)

self.config.update\_config('branding', 'secondary\_color', secondary\_color)

st.success('Branding updated')

st.rerun()

def render\_license\_info(self):

"""Render license information"""

st.subheader('📜 License Information')

valid, message = self.license.validate\_license()

if valid:

st.success(f'✅ {message}')

else:

st.error(f'❌ {message}')

config = self.config.config['subscription']

col1, col2, col3 = st.columns(3)

with col1:

st.metric('Subscription Tier', config['tier'].title())

with col2:

st.metric('Licensed Seats', config['seats'])

with col3:

valid\_until = datetime.fromisoformat(config['valid\_until'])

days\_remaining = (valid\_until - datetime.now()).days

st.metric('Days Remaining', days\_remaining)

# Feature access

st.subheader('Available Features')

features = {

'Basic Management': self.license.check\_feature\_access('basic\_management'),

'AI Modules': self.license.check\_feature\_access('ai\_modules'),

'Advanced Analytics': self.license.check\_feature\_access('advanced\_analytics'),

'API Access': self.license.check\_feature\_access('api\_access'),

'Multi-Facility': self.license.check\_feature\_access('multi\_facility'),

'White Label': self.license.check\_feature\_access('white\_label')

}

cols = st.columns(3)

for i, (feature, available) in enumerate(features.items()):

with cols[i % 3]:

if available:

st.success(f'✅ {feature}')

else:

st.info(f'🔒 {feature}')

# Upgrade options

if config['tier'] != 'enterprise':

st.info('📈 Upgrade your subscription to unlock more features')

if st.button('View Upgrade Options'):

st.session\_state.show\_pricing = True

def render\_system\_health(self):

"""Render system health dashboard"""

st.subheader('💊 System Health')

# Module health

module\_health = self.module\_loader.get\_module\_health()

col1, col2, col3, col4 = st.columns(4)

with col1:

st.metric('Modules Loaded', module\_health['loaded'])

with col2:

st.metric('Modules Failed', module\_health['failed'])

with col3:

health\_percent = module\_health['health\_score'] \* 100

st.metric('Health Score', f"{health\_percent:.1f}%")

with col4:

st.metric('Total Modules', module\_health['total'])

# System metrics

st.subheader('System Metrics')

# Calculate metrics

db\_size = sum(f.stat().st\_size for f in Path('database').glob('\*\*/\*') if f.is\_file()) / (1024 \* 1024)

log\_size = sum(f.stat().st\_size for f in Path('logs').glob('\*\*/\*') if f.is\_file()) / (1024 \* 1024) if Path('logs').exists() else 0

col1, col2, col3 = st.columns(3)

with col1:

st.metric('Database Size', f"{db\_size:.2f} MB")

with col2:

st.metric('Log Size', f"{log\_size:.2f} MB")

with col3:

st.metric('Active Sessions', len(st.session\_state.get('active\_sessions', [])))

# Failed modules detail

if self.module\_loader.failed\_modules:

with st.expander('❌ Failed Modules'):

for module, error in self.module\_loader.failed\_modules.items():

st.error(f"\*\*{module}\*\*: {error}")

def render\_audit\_logs(self):

"""Render audit logs"""

st.subheader('📝 Audit Logs')

audit\_dir = Path("audit\_logs")

if not audit\_dir.exists():

st.info('No audit logs available')

return

# Load recent logs

log\_files = sorted(audit\_dir.glob('\*.jsonl'), reverse=True)

if not log\_files:

st.info('No audit logs available')

return

# Select log file

selected\_file = st.selectbox('Select Log File',

[f.name for f in log\_files])

if selected\_file:

log\_file = audit\_dir / selected\_file

# Read logs

logs = []

with open(log\_file, 'r') as f:

for line in f:

logs.append(json.loads(line))

# Filter options

col1, col2, col3 = st.columns(3)

with col1:

action\_filter = st.selectbox('Filter by Action',

['All'] + list(set(log['action'] for log in logs)))

with col2:

user\_filter = st.selectbox('Filter by User',

['All'] + list(set(log['user'] for log in logs)))

with col3:

date\_filter = st.date\_input('Filter by Date')

# Apply filters

filtered\_logs = logs

if action\_filter != 'All':

filtered\_logs = [l for l in filtered\_logs if l['action'] == action\_filter]

if user\_filter != 'All':

filtered\_logs = [l for l in filtered\_logs if l['user'] == user\_filter]

if date\_filter:

filtered\_logs = [l for l in filtered\_logs

if datetime.fromisoformat(l['timestamp']).date() == date\_filter]

# Display logs

if filtered\_logs:

log\_data = []

for log in filtered\_logs[:100]: # Limit to 100 most recent

log\_data.append({

'Time': datetime.fromisoformat(log['timestamp']).strftime('%Y-%m-%d %H:%M:%S'),

'User': log['user'],

'Action': log['action'],

'Details': log.get('details', ''),

'Session': log.get('session\_id', 'N/A')[:8]

})

st.dataframe(log\_data, use\_container\_width=True)

else:

st.info('No logs match the selected filters')

def render\_pricing\_page(self):

"""Render pricing and upgrade page"""

st.header('💎 Subscription Plans')

plans = {

'Starter': {

'price': '$99/month',

'features': [

'✅ Basic facility management',

'✅ Member management',

'✅ Event scheduling',

'✅ Basic reporting',

'✅ 5 user accounts',

'✅ Email support',

'❌ AI modules',

'❌ Advanced analytics',

'❌ API access'

]

},

'Professional': {

'price': '$299/month',

'features': [

'✅ Everything in Starter',

'✅ AI-powered insights',

'✅ Advanced analytics',

'✅ Revenue optimization',

'✅ API access',

'✅ 25 user accounts',

'✅ Priority support',

'✅ Custom branding',

'❌ Multi-facility support'

]

},

'Enterprise': {

'price': 'Custom',

'features': [

'✅ Everything in Professional',

'✅ Multi-facility management',

'✅ Unlimited users',

'✅ Custom integrations',

'✅ White-label options',

'✅ Dedicated support',

'✅ SLA guarantee',

'✅ On-premise deployment',

'✅ Custom training'

]

}

}

cols = st.columns(3)

for i, (plan\_name, plan\_info) in enumerate(plans.items()):

with cols[i]:

st.markdown(f"### {plan\_name}")

st.markdown(f"\*\*{plan\_info['price']}\*\*")

for feature in plan\_info['features']:

st.markdown(feature)

if plan\_name == 'Enterprise':

if st.button('Contact Sales', key=f'upgrade\_{plan\_name}', use\_container\_width=True):

st.info('Please contact sales@sportai.com for Enterprise pricing')

else:

if st.button(f'Upgrade to {plan\_name}', key=f'upgrade\_{plan\_name}', use\_container\_width=True):

st.info(f'Redirecting to payment page for {plan\_name} plan...')

st.markdown("---")

st.markdown("""

### 🎯 Why Choose SportAI Suite?

- \*\*Industry-Leading AI\*\*: Powered by advanced machine learning for optimal facility management

- \*\*Proven ROI\*\*: Average 35% increase in facility utilization within 6 months

- \*\*Trusted by 500+ Facilities\*\*: From local sports complexes to major stadiums

- \*\*24/7 Support\*\*: Expert help whenever you need it

- \*\*Regular Updates\*\*: New features and improvements every month

- \*\*Data Security\*\*: Enterprise-grade security and compliance

""")

def render\_dashboard(self):

"""Render main dashboard"""

user = st.session\_state.user

# Custom CSS with branding colors

st.markdown(f"""

<style>

.stApp {{

background: linear-gradient(135deg, {self.config.config['branding']['primary\_color']}10 0%, {self.config.config['branding']['secondary\_color']}10 100%);

}}

</style>

""", unsafe\_allow\_html=True)

# Header

st.title(f"🏟️ {self.config.config['branding']['facility\_name']}")

st.markdown(f"Welcome back, \*\*{user['email']}\*\*! | Role: \*\*{user['role'].title()}\*\*")

# Quick stats

col1, col2, col3, col4 = st.columns(4)

with col1:

st.metric("Available Tools", len(self.tools))

with col2:

tier = self.config.config['subscription']['tier']

st.metric("Subscription", tier.title())

with col3:

module\_health = self.module\_loader.get\_module\_health()

health\_percent = module\_health['health\_score'] \* 100

st.metric("System Health", f"{health\_percent:.0f}%")

with col4:

valid\_until = datetime.fromisoformat(self.config.config['subscription']['valid\_until'])

days\_remaining = (valid\_until - datetime.now()).days

st.metric("License Days", days\_remaining)

# Main content tabs

tabs = st.tabs(['📊 Overview', '🛠️ Tools', '📈 Analytics', '⚙️ Settings'])

with tabs[0]: # Overview

st.header('Dashboard Overview')

# Activity metrics

st.subheader('📊 Today\'s Activity')

col1, col2, col3, col4 = st.columns(4)

with col1:

st.metric('Active Members', '1,247', delta='+12')

with col2:

st.metric('Facility Usage', '87%', delta='+5%')

with col3:

st.metric('Revenue Today', '$4,532', delta='+$234')

with col4:

st.metric('Events Scheduled', '18', delta='+3')

# Quick actions

st.subheader('🚀 Quick Actions')

action\_cols = st.columns(4)

with action\_cols[0]:

if st.button('📅 Schedule Event', use\_container\_width=True):

st.session\_state.tool\_selection = '🎯 Event Control Panel'

st.rerun()

with action\_cols[1]:

if st.button('👥 Add Member', use\_container\_width=True):

st.session\_state.tool\_selection = '👥 Membership Dashboard'

st.rerun()

with action\_cols[2]:

if st.button('📊 View Reports', use\_container\_width=True):

st.session\_state.tool\_selection = '📊 Reports Generator'

st.rerun()

with action\_cols[3]:

if st.button('💰 Revenue Analysis', use\_container\_width=True):

st.session\_state.tool\_selection = '🔥 Revenue Heatmap'

st.rerun()

with tabs[1]: # Tools

st.header('Available Tools')

# Search

search\_term = st.text\_input('🔍 Search tools...', placeholder='Type to filter')

# Filter tools

if search\_term:

filtered\_tools = {k: v for k, v in self.tools.items()

if search\_term.lower() in k.lower()}

else:

filtered\_tools = self.tools

# Display tools in grid

if filtered\_tools:

cols = st.columns(3)

for i, (tool\_name, tool\_module) in enumerate(filtered\_tools.items()):

with cols[i % 3]:

if st.button(tool\_name, key=f'tool\_{tool\_name}', use\_container\_width=True):

st.session\_state.tool\_selection = tool\_name

st.rerun()

else:

st.info('No tools found matching your search')

with tabs[2]: # Analytics

st.header('Analytics Dashboard')

# Date range selector

col1, col2 = st.columns(2)

with col1:

start\_date = st.date\_input('Start Date', datetime.now() - timedelta(days=30))

with col2:

end\_date = st.date\_input('End Date', datetime.now())

# Metrics

st.subheader('Key Performance Indicators')

kpi\_cols = st.columns(4)

with kpi\_cols[0]:

st.metric('Total Revenue', '$127,543', delta='+12.3%')

with kpi\_cols[1]:

st.metric('Member Retention', '92%', delta='+2.1%')

with kpi\_cols[2]:

st.metric('Facility Utilization', '78%', delta='+5.7%')

with kpi\_cols[3]:

st.metric('Customer Satisfaction', '4.7/5', delta='+0.2')

# Charts placeholder

st.info('📊 Interactive charts and detailed analytics available in Professional and Enterprise plans')

with tabs[3]: # Settings

if user['role'] == 'admin':

self.render\_admin\_panel()

else:

st.info('Admin access required for settings management')

def run(self):

"""Main application runner"""

st.set\_page\_config(

page\_title=f'{self.config.config["branding"]["facility\_name"]} - SportAI Suite',

page\_icon='🏟️',

layout='wide',

initial\_sidebar\_state='expanded'

)

# Initialize session state

if 'show\_registration' not in st.session\_state:

st.session\_state.show\_registration = False

if 'show\_pricing' not in st.session\_state:

st.session\_state.show\_pricing = False

# Check if showing special pages

if st.session\_state.show\_registration:

self.register\_user()

if st.button('← Back to Login'):

st.session\_state.show\_registration = False

st.rerun()

return

if st.session\_state.show\_pricing:

self.render\_pricing\_page()

if st.button('← Back'):

st.session\_state.show\_pricing = False

st.rerun()

return

# Validate session

if not self.validate\_session():

# Show login page

st.title('🏟️ SportAI Suite Enterprise')

st.markdown(f"""

### {self.config.config['branding']['facility\_name']} Management Platform

\*\*Enterprise-grade sports facility management with AI-powered insights\*\*

- 🤖 AI-Powered Optimization

- 📊 Advanced Analytics

- 💰 Revenue Maximization

- 🔐 Enterprise Security

- 📱 Mobile Responsive

- 🌐 Multi-Facility Support

Please log in to continue.

""")

self.login()

return

# User is logged in

user = st.session\_state.user

# Check for password change requirement

if user.get('must\_change\_password'):

st.warning('⚠️ You must change your password')

new\_password = st.text\_input('New Password', type='password')

confirm\_password = st.text\_input('Confirm Password', type='password')

if st.button('Change Password'):

if new\_password == confirm\_password:

users = self.db.load\_users()

users[user['email']]['password\_hash'] = self.security.hash\_password(new\_password)

users[user['email']]['must\_change\_password'] = False

self.db.save\_users(users)

st.session\_state.user['must\_change\_password'] = False

st.success('Password changed successfully')

st.rerun()

else:

st.error('Passwords do not match')

return

# Sidebar

with st.sidebar:

st.success(f"✅ {user['email']}")

st.caption(f"Role: {user['role'].title()}")

# Session info

elapsed = int(time.time() - user['login\_time'])

remaining = self.security.session\_timeout - elapsed

st.caption(f"Session: {remaining//60} min remaining")

if st.button('🚪 Logout', use\_container\_width=True):

self.security.create\_audit\_log(user['email'], 'LOGOUT')

st.session\_state.clear()

st.rerun()

st.markdown("---")

# Tool selection

if 'tool\_selection' in st.session\_state and st.session\_state.tool\_selection:

st.info(f"Tool: {st.session\_state.tool\_selection}")

if st.button('← Back to Dashboard'):

st.session\_state.tool\_selection = None

st.rerun()

# API Key for developers

if user.get('api\_key'):

with st.expander('🔑 API Access'):

st.code(user['api\_key'][:10] + '...')

if st.button('Copy Full Key'):

st.code(user['api\_key'])

# Main content

if 'tool\_selection' in st.session\_state and st.session\_state.tool\_selection:

# Run selected tool

tool\_name = st.session\_state.tool\_selection

if tool\_name in self.tools:

tool\_module = self.tools[tool\_name]

try:

st.header(tool\_name)

if hasattr(tool\_module, 'run'):

tool\_module.run()

else:

st.error(f"Tool {tool\_name} is not properly configured")

except Exception as e:

st.error(f"Error running {tool\_name}: {e}")

logger.error(f"Tool execution error: {e}")

else:

st.error(f"Tool {tool\_name} not found")

else:

# Show main dashboard

self.render\_dashboard()

# ============= ENTRY POINT =============

if \_\_name\_\_ == "\_\_main\_\_":

try:

app = SportAIEnterpriseApp()

app.run()

except Exception as e:

logger.critical(f"Application failed to start: {e}")

st.error(f"Critical error: {e}")

st.stop()