CASHPOT Gaming Management System - Documentație Completă

1. ARHITECTURA GENERALĂ

1.1 Stack Tehnologic

- **Frontend**: React.js (SPA)
- **Backend**: FastAPI (Python)
- **Database**: MongoDB (NoSQL)
- **Authentication**: JWT (JSON Web Tokens)
- **Password Hashing**: bcrypt
- **File Storage**: Base64 în database
- **Styling**: CSS custom cu variabile pentru teme

1.2 Structura Proiectului

```
deploy-687cc9f68045b4912ab57737/
├── backend/
│   ├── server.py (FastAPI app)
│   └── requirements.txt
├── frontend/
│   ├── src/
│   │   ├── app.js (componenta principală)
│   │   └── app.css (stilizare)
│   ├── package.json
│   └── .env
└── static/ (build files)
```

2. BACKEND - FASTAPI SERVER

2.1 Configurare Inițială

```python
# server.py
from fastapi import FastAPI, Depends, HTTPException
from fastapi.security import HTTPBearer, HTTPAuthorizationCredentials
import motor.motor_asyncio
import bcrypt
import jwt
import os

# Configurare MongoDB
MONGO_URL = os.environ.get('MONGO_URL', 'mongodb://localhost:27017')
DB_NAME = os.environ.get('DB_NAME', 'financial_planner_dev')

client = motor.motor_asyncio.AsyncIOMotorClient(MONGO_URL)
db = client[DB_NAME]

# JWT Config
```

```
JWT_SECRET = "your-secret-key"
JWT_ALGORITHM = "HS256"
```

2.2 Modele de Date (Pydantic)

2.2.1 User Management

```
class UserRole(str):
    ADMIN = "admin"
    MANAGER = "manager"
    OPERATOR = "operator"

class UserPermissions(BaseModel):
    modules: dict = {
        "dashboard": True,
        "companies": False,
        "locations": False,
        "providers": False,
        "cabinets": False,
        "game_mixes": False,
        "slot_machines": False,
        "invoices": False,
        "onjn_reports": False,
        "legal_documents": False,
        "metrology": False,
        "jackpots": False,
        "users": False
    }
    actions: dict = {
        "companies": {"create": False, "read": False, "update": False, "delete":
False},
        # ... pentru toate modulele
    }
    accessible_companies: List[str] = []
    accessible_locations: List[str] = []

class User(BaseModel):
    id: str = Field(default_factory=lambda: str(uuid.uuid4()))
    username: str
    email: str
    password_hash: str
    first_name: str = ""
    last_name: str = ""
    role: str = UserRole.ADMIN
    assigned_locations: List[str] = []
    permissions: UserPermissions = Field(default_factory=UserPermissions)
    created_at: datetime = Field(default_factory=datetime.utcnow)
    is_active: bool = True
```

2.2.2 Gaming Entities

```python
class Company(BaseModel):
    id: str = Field(default_factory=lambda: str(uuid.uuid4()))
    name: str
    registration_number: str
    tax_id: str
    address: str
    phone: str
    email: str
    contact_person: str
    status: str = "active"
    created_at: datetime = Field(default_factory=datetime.utcnow)
    created_by: str

class Location(BaseModel):
    id: str = Field(default_factory=lambda: str(uuid.uuid4()))
    name: str
    address: str
    city: str
    county: str
    country: str = "Romania"
    postal_code: str
    latitude: Optional[float] = None
    longitude: Optional[float] = None
    company_id: str
    manager_id: Optional[str] = None
    status: str = "active"
    created_at: datetime = Field(default_factory=datetime.utcnow)
    created_by: str

class Provider(BaseModel):
    id: str = Field(default_factory=lambda: str(uuid.uuid4()))
    name: str
    company_name: str
    contact_person: str
    email: str
    phone: str
    address: str
    status: str = "active"
    created_at: datetime = Field(default_factory=datetime.utcnow)
    created_by: str

class Cabinet(BaseModel):
    id: str = Field(default_factory=lambda: str(uuid.uuid4()))
    name: str
    model: Optional[str] = None
    provider_id: str
    status: str = "active"
    created_at: datetime = Field(default_factory=datetime.utcnow)
    created_by: str

class GameMix(BaseModel):
```

```
    id: str = Field(default_factory=lambda: str(uuid.uuid4()))
    name: str
    description: str
    provider_id: str
    game_count: int
    games: List[str]
    status: str = "active"
    created_at: datetime = Field(default_factory=datetime.utcnow)
    created_by: str

class SlotMachine(BaseModel):
    id: str = Field(default_factory=lambda: str(uuid.uuid4()))
    cabinet_id: str
    game_mix_id: str
    provider_id: str
    model: str
    serial_number: str
    denomination: float
    max_bet: float
    rtp: float
    gaming_places: int
    commission_date: Optional[datetime] = None
    invoice_number: Optional[str] = None
    status: str = "active"
    location_id: Optional[str] = None
    production_year: Optional[int] = None
    created_at: datetime = Field(default_factory=datetime.utcnow)
    created_by: str
```

2.2.3 Business Entities

```
class Invoice(BaseModel):
    id: str = Field(default_factory=lambda: str(uuid.uuid4()))
    invoice_number: str
    company_id: str
    location_id: str
    buyer_id: Optional[str] = None
    seller_id: Optional[str] = None
    serial_numbers: str
    issue_date: datetime
    due_date: datetime
    amount: float
    currency: str = "EUR"
    status: str = "pending"
    description: str
    created_at: datetime = Field(default_factory=datetime.utcnow)
    created_by: str

class Metrology(BaseModel):
    id: str = Field(default_factory=lambda: str(uuid.uuid4()))
    serial_number: str
```

```
    certificate_number: str
    certificate_type: str
    issue_date: datetime
    expiry_date: datetime
    issuing_authority: str
    calibration_interval: int
    next_calibration_date: datetime
    status: str = "active"
    description: str
    created_at: datetime = Field(default_factory=datetime.utcnow)
    created_by: str

class Jackpot(BaseModel):
    id: str = Field(default_factory=lambda: str(uuid.uuid4()))
    serial_number: str
    jackpot_type: str
    jackpot_name: str
    current_amount: float
    max_amount: float
    reset_amount: float
    increment_rate: float
    last_reset_date: datetime
    next_reset_date: Optional[datetime] = None
    status: str = "active"
    description: str
    created_at: datetime = Field(default_factory=datetime.utcnow)
    created_by: str
```

2.3 Funcții Utilitare

2.3.1 Password Management

```
def hash_password(password: str) -> str:
    return bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt()).decode('utf-8')

def verify_password(password: str, hashed: str) -> bool:
    try:
        if not isinstance(password, str):
            password = str(password)
        if not isinstance(hashed, str):
            hashed = str(hashed)
        return bcrypt.checkpw(password.encode('utf-8'), hashed.encode('utf-8'))
    except Exception as e:
        print(f"❌ Error in verify_password: {e}")
        return False
```

2.3.2 JWT Management

```
def create_access_token(data: dict, expires_delta: Optional[timedelta] = None):
    to_encode = data.copy()
```

```
    if expires_delta:
        expire = datetime.utcnow() + expires_delta
    else:
        expire = datetime.utcnow() + timedelta(hours=24)
    to_encode.update({"exp": expire})
    return jwt.encode(to_encode, JWT_SECRET, algorithm=JWT_ALGORITHM)

async def get_current_user(credentials: HTTPAuthorizationCredentials =
Depends(security)):
    try:
        payload = jwt.decode(credentials.credentials, JWT_SECRET, algorithms=
[JWT_ALGORITHM])
        user_id: str = payload.get("sub")
        if user_id is None:
            raise HTTPException(status_code=401, detail="Invalid authentication
credentials")

        user = await db.users.find_one({"id": user_id})
        if user is None:
            raise HTTPException(status_code=401, detail="User not found")
        return User(**user)
    except jwt.PyJWTError:
        raise HTTPException(status_code=401, detail="Invalid authentication
credentials")
```

2.4 API Endpoints

2.4.1 Authentication

```
@api_router.post("/auth/login", response_model=dict)
async def login(user_data: UserLogin):
    user = await db.users.find_one({"username": user_data.username})
    if not user:
        raise HTTPException(status_code=401, detail="Invalid username or password")

    password_valid = verify_password(user_data.password, user["password_hash"])
    if not password_valid:
        raise HTTPException(status_code=401, detail="Invalid username or password")

    if not user.get("is_active", True):
        raise HTTPException(status_code=401, detail="Account is inactive")

    access_token = create_access_token(data={"sub": user["id"]})
    return {
        "access_token": access_token,
        "token_type": "bearer",
        "user": {
            "id": user["id"],
            "username": user["username"],
            "first_name": user.get("first_name", ""),
            "last_name": user.get("last_name", ""),
```

```
            "role": user.get("role", "admin")
        }
    }
```

2.4.2 CRUD Endpoints (exemplu pentru Companies)

```
@api_router.post("/companies", response_model=Company)
async def create_company(company_data: CompanyCreate, current_user: User =
Depends(get_current_user)):
    if not await check_user_permission(current_user, "companies", "create"):
        raise HTTPException(status_code=403, detail="Insufficient permissions")

    company = Company(**company_data.dict(), created_by=current_user.id)
    await db.companies.insert_one(company.dict())
    return company

@api_router.get("/companies", response_model=List[Company])
async def get_companies(current_user: User = Depends(get_current_user)):
    if not await check_user_permission(current_user, "companies", "read"):
        raise HTTPException(status_code=403, detail="Insufficient permissions")

    query = await filter_by_user_access(current_user, {}, "companies")
    cursor = db.companies.find(query)
    companies = await cursor.to_list(length=None)
    return [Company(**convert_objectid_to_str(company)) for company in companies]
```

3. FRONTEND - REACT APPLICATION

3.1 Configurare Inițială

```
// app.js
const BACKEND_URL = process.env.REACT_APP_BACKEND_URL || 'http://localhost:8002';
const API = `${BACKEND_URL}/api`;

// State Management
const [user, setUser] = useState(null);
const [token, setToken] = useState(localStorage.getItem('token'));
const [currentView, setCurrentView] = useState('dashboard');
const [companies, setCompanies] = useState([]);
const [locations, setLocations] = useState([]);
const [providers, setProviders] = useState([]);
const [cabinets, setCabinets] = useState([]);
const [gameMixes, setGameMixes] = useState([]);
const [slotMachines, setSlotMachines] = useState([]);
const [invoices, setInvoices] = useState([]);
const [metrology, setMetrology] = useState([]);
const [jackpots, setJackpots] = useState([]);
const [users, setUsers] = useState([]);
```

3.2 Authentication Flow

```
// Login Function
async function handleLogin(e) {
    e.preventDefault();
    try {
        const response = await fetch(`${API}/auth/login`, {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ username, password })
        });

        if (response.ok) {
            const data = await response.json();
            localStorage.setItem('token', data.access_token);
            setToken(data.access_token);
            setUser(data.user);
            setLoginError('');
            fetchDashboardData();
        } else {
            const error = await response.json();
            setLoginError(error.detail || 'Login failed');
        }
    } catch (error) {
        setLoginError('Network error');
    }
}

// Logout Function
function handleLogout() {
    localStorage.removeItem('token');
    setToken(null);
    setUser(null);
    setCurrentView('login');
}
```

3.3 Data Fetching

```
// Fetch Dashboard Data
async function fetchDashboardData() {
    if (!token) return;

    try {
        const headers = { 'Authorization': `Bearer ${token}` };

        const [companiesRes, locationsRes, providersRes, cabinetsRes,
               gameMixesRes, slotMachinesRes, invoicesRes, metrologyRes,
               jackpotsRes, usersRes] = await Promise.all([
            fetch(`${API}/companies`, { headers }),
            fetch(`${API}/locations`, { headers }),
```

```
            fetch(`${API}/providers`, { headers }),
            fetch(`${API}/cabinets`, { headers }),
            fetch(`${API}/game-mixes`, { headers }),
            fetch(`${API}/slot-machines`, { headers }),
            fetch(`${API}/invoices`, { headers }),
            fetch(`${API}/metrology`, { headers }),
            fetch(`${API}/jackpots`, { headers }),
            fetch(`${API}/users`, { headers })
        ]);

        if (companiesRes.ok) setCompanies(await companiesRes.json());
        if (locationsRes.ok) setLocations(await locationsRes.json());
        if (providersRes.ok) setProviders(await providersRes.json());
        if (cabinetsRes.ok) setCabinets(await cabinetsRes.json());
        if (gameMixesRes.ok) setGameMixes(await gameMixesRes.json());
        if (slotMachinesRes.ok) setSlotMachines(await slotMachinesRes.json());
        if (invoicesRes.ok) setInvoices(await invoicesRes.json());
        if (metrologyRes.ok) setMetrology(await metrologyRes.json());
        if (jackpotsRes.ok) setJackpots(await jackpotsRes.json());
        if (usersRes.ok) setUsers(await usersRes.json());
    } catch (error) {
        console.error('Error fetching data:', error);
    }
}
```

3.4 UI Components

3.4.1 Navigation

```
// Navigation Component
function Navigation() {
    const views = [
        { id: 'dashboard', label: 'Dashboard', icon: '📊' },
        { id: 'companies', label: 'Companies', icon: '🏢' },
        { id: 'locations', label: 'Locations', icon: '📍' },
        { id: 'providers', label: 'Providers', icon: '🏭' },
        { id: 'cabinets', label: 'Cabinets', icon: '🎰' },
        { id: 'game_mixes', label: 'Game Mixes', icon: '🎮' },
        { id: 'slots', label: 'Slot Machines', icon: '🎯' },
        { id: 'invoices', label: 'Invoices', icon: '📄' },
        { id: 'onjn_reports', label: 'ONJN Reports', icon: '📋' },
        { id: 'legal_documents', label: 'Legal Documents', icon: '⚖️' },
        { id: 'metrology', label: 'Metrology', icon: '🔬' },
        { id: 'jackpots', label: 'Jackpots', icon: '💰' },
        { id: 'users', label: 'Users', icon: '👥' }
    ];

    return (
        <nav className="sidebar">
            {views.map(view => (
                <button
```

```
                    key={view.id}
                    className={`nav-item ${currentView === view.id ? 'active' :
''}`}
                    onClick={() => setCurrentView(view.id)}
                >
                    <span className="icon">{view.icon}</span>
                    <span className="label">{view.label}</span>
                </button>
            ))}
        </nav>
    );
}
```

3.4.2 Data Table Component

```
// Data Table Component
function DataTable({
    title,
    data,
    columns,
    entityType,
    onAdd,
    onEdit,
    onDelete,
    onView
}) {
    const [searchTerm, setSearchTerm] = useState('');
    const [sortField, setSortField] = useState('');
    const [sortDirection, setSortDirection] = useState('asc');

    // Filter data based on search term
    const filteredData = data.filter(item => {
        return Object.values(item).some(value =>
            String(value).toLowerCase().includes(searchTerm.toLowerCase())
        );
    });

    // Sort data
    const sortedData = [...filteredData].sort((a, b) => {
        if (!sortField) return 0;

        const aVal = a[sortField];
        const bVal = b[sortField];

        if (aVal < bVal) return sortDirection === 'asc' ? -1 : 1;
        if (aVal > bVal) return sortDirection === 'asc' ? 1 : -1;
        return 0;
    });

    return (
        <div className="data-table">
```

```
            <div className="table-header">
                <h2>{title} ({sortedData.length})</h2>
                <div className="table-actions">
                    <input
                        type="text"
                        placeholder="Search..."
                        value={searchTerm}
                        onChange={(e) => setSearchTerm(e.target.value)}
                        className="search-input"
                    />
                    {onAdd && (
                        <button className="btn-primary" onClick={onAdd}>
                            <span className="icon">+</span>
                            Add {entityType === 'slots' ? 'Slot' : entityType ===
'metrology' ? 'Metrologie' : title.slice(0, -1)}
                        </button>
                    )}
                </div>
            </div>

            <table>
                <thead>
                    <tr>
                        {columns.map(column => (
                            <th
                                key={column.key}
                                onClick={() => {
                                    if (sortField === column.key) {
                                        setSortDirection(sortDirection === 'asc' ?
'desc' : 'asc');
                                    } else {
                                        setSortField(column.key);
                                        setSortDirection('asc');
                                    }
                                }}
                                className={sortField === column.key ?
`sort-${sortDirection}` : ''}
                            >
                                {column.label}
                            </th>
                        ))}
                        <th>Actions</th>
                    </tr>
                </thead>
                <tbody>
                    {sortedData.map(item => (
                        <tr key={item.id}>
                            {columns.map(column => (
                                <td key={column.key}>
                                    {column.render ? column.render(item) :
item[column.key]}
                                </td>
```

```
                            ))}
                            <td className="actions">
                                {onView && (
                                    <button onClick={() => onView(item)}
className="btn-icon">👁</button>
                                )}
                                {onEdit && (
                                    <button onClick={() => onEdit(item)}
className="btn-icon">✏️</button>
                                )}
                                {onDelete && (
                                    <button onClick={() => onDelete(item)}
className="btn-icon">🗑️</button>
                                )}
                            </td>
                        </tr>
                    ))}
                </tbody>
            </table>
        </div>
    );
}
```

3.5 Specialized Views

3.5.1 Slot Machines View

```
// Slot Machines Table Configuration
const slotColumns = [
    {
        key: 'serial_number',
        label: 'Serial Number',
        render: (item) => (
            <div>
                <div>{item.serial_number}</div>
                <div className="sub-text">{item.location_name || 'No location'}
</div>
            </div>
        )
    },
    {
        key: 'provider_name',
        label: 'Provider',
        render: (item) => (
            <div>
                <div>{item.provider_name}</div>
                <div className="sub-text">{item.cabinet_name}</div>
            </div>
        )
    },
    {
```

```
        key: 'game_mix_name',
        label: 'Game Mix',
        render: (item) => (
            <div>
                <div>{item.game_mix_name}</div>
                <div className="sub-text">{item.model}</div>
            </div>
        )
    },
    {
        key: 'company_name',
        label: 'Property',
        render: (item) => {
            const company = companies.find(c => c.id === item.company_id);
            const location = locations.find(l => l.id === item.location_id);
            if (item.ownership_type === 'property') {
                return `${company?.name || 'Unknown'} - ${item.invoice_number || 'No
invoice'}`;
            } else {
                return `${item.provider_name} - ${item.lease_contract_number || 'No
contract'}`;
            }
        }
    },
    {
        key: 'technical_specs',
        label: 'Technical Specs',
        render: (item) => (
            <div>
                <div>Denom: {item.denomination} | Max Bet: {item.max_bet}</div>
                <div>RTP: {item.rtp}% | Places: {item.gaming_places}</div>
            </div>
        )
    }
];
```

3.5.2 Metrology View

```
// Metrology Table Configuration
const metrologyColumns = [
    {
        key: 'serial_number',
        label: 'Serial Number',
        render: (item) => item.serial_number
    },
    {
        key: 'certificate_number',
        label: 'Certificate Number',
        render: (item) => item.certificate_number
    },
    {
```

```
        key: 'cvt_date',
        label: 'Data CVT',
        render: (item) => (
            <input
                type="date"
                value={item.cvt_date ? item.cvt_date.slice(0,10) : ''}
                onChange={e => handleMetrologyDateChange(item.serial_number,
e.target.value)}
                style={{
                    width: 120,
                    backgroundColor: 'var(--bg-secondary)',
                    color: 'var(--text-primary)',
                    border: '1px solid var(--border-color)',
                    borderRadius: '4px',
                    padding: '4px 8px'
                }}
            />
        )
    },
    {
        key: 'days_left',
        label: 'Days Left',
        render: (item) => {
            let daysLeft = 0;
            if (item.cvt_date) {
                const cvtDate = new Date(item.cvt_date);
                const expiryDate = new Date(cvtDate);
                expiryDate.setFullYear(expiryDate.getFullYear() + 1);
                const now = new Date();
                const diff = Math.ceil((expiryDate - now) / (1000 * 60 * 60 * 24));
                daysLeft = diff > 0 ? diff : 0;
            } else if (item.expiry_date) {
                const expiryDate = new Date(item.expiry_date);
                const now = new Date();
                const diff = Math.ceil((expiryDate - now) / (1000 * 60 * 60 * 24));
                daysLeft = diff > 0 ? diff : 0;
            }

            let color = '#10b981';
            if (daysLeft <= 30) {
                color = '#ef4444';
            } else if (daysLeft <= 90) {
                color = '#f59e0b';
            }

            return (
                <div style={{
                    textAlign: 'center',
                    color: color,
                    fontWeight: 'bold',
                    fontSize: '1.1em',
                    padding: '4px 8px',
```

```
                    borderRadius: '4px',
                    backgroundColor: color === '#ef4444' ? 'rgba(239, 68, 68, 0.1)'
:
                                     color === '#f59e0b' ? 'rgba(245, 158, 11, 0.1)'
:
                                     'rgba(16, 185, 129, 0.1)'
                }}>
                    {daysLeft}
                </div>
            );
        }
    }
];

// Metrology Date Change Handler
function handleMetrologyDateChange(serialNumber, newDate) {
    const token = localStorage.getItem('token');

    const existing = metrology.find(m => m.serial_number === serialNumber);

    if (existing) {
        fetch(`${API}/metrology/${existing.id}`, {
            method: 'PUT',
            headers: { 'Content-Type': 'application/json', Authorization: `Bearer
${token}` },
            body: JSON.stringify({
                ...existing,
                cvt_date: newDate,
                certificate_number: existing.certificate_number || 'CVT-' +
serialNumber,
                certificate_type: existing.certificate_type || 'calibration',
                issue_date: existing.issue_date || newDate,
                expiry_date: existing.expiry_date || new Date(new
Date(newDate).getFullYear() + 1, new Date(newDate).getMonth(), new
Date(newDate).getDate()),
                issuing_authority: existing.issuing_authority || 'ANM',
                calibration_interval: existing.calibration_interval || 12,
                description: existing.description || 'CVT Certificate'
            }),
        }).then(() => fetchDashboardData());
    } else {
        fetch(`${API}/metrology`, {
            method: 'POST',
            headers: { 'Content-Type': 'application/json', Authorization: `Bearer
${token}` },
            body: JSON.stringify({
                serial_number: serialNumber,
                cvt_date: newDate,
                certificate_number: 'CVT-' + serialNumber,
                certificate_type: 'calibration',
                issue_date: newDate,
                expiry_date: new Date(new Date(newDate).getFullYear() + 1, new
```

```
Date(newDate).getMonth(), new Date(newDate).getDate()),
                issuing_authority: 'ANM',
                calibration_interval: 12,
                description: 'CVT Certificate'
            }),
        }).then(() => fetchDashboardData());
    }
}
```

4. STYLING - CSS VARIABLES

4.1 Theme System

```
/* app.css */
:root {
    /* Light Theme */
    --bg-primary: #ffffff;
    --bg-secondary: #f8fafc;
    --bg-tertiary: #f1f5f9;
    --text-primary: #1e293b;
    --text-secondary: #64748b;
    --border-color: #e2e8f0;
    --accent-color: #3b82f6;
    --success-color: #10b981;
    --warning-color: #f59e0b;
    --error-color: #ef4444;
    --shadow: 0 1px 3px 0 rgba(0, 0, 0, 0.1);
}

[data-theme="dark"] {
    /* Dark Theme */
    --bg-primary: #1e293b;
    --bg-secondary: #334155;
    --bg-tertiary: #475569;
    --text-primary: #f8fafc;
    --text-secondary: #cbd5e1;
    --border-color: #475569;
    --accent-color: #60a5fa;
    --success-color: #34d399;
    --warning-color: #fbbf24;
    --error-color: #f87171;
    --shadow: 0 1px 3px 0 rgba(0, 0, 0, 0.3);
}

/* Avatar Styling */
.avatar {
    width: 68px;
    height: 68px;
    border-radius: 50%;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.3);
```

```
    display: flex;
    align-items: center;
    justify-content: center;
    font-weight: bold;
    font-size: 1.2em;
    color: var(--text-primary);
    background: var(--bg-secondary);
}

.header-avatar {
    background: transparent;
    box-shadow: none;
    border: none;
}
```

5. DATABASE SCHEMA

5.1 Collections Structure

```
// MongoDB Collections

// users
{
    "_id": ObjectId,
    "id": "uuid-string",
    "username": "admin",
    "email": "admin@example.com",
    "password_hash": "bcrypt-hash",
    "first_name": "Admin",
    "last_name": "User",
    "role": "admin",
    "assigned_locations": [],
    "permissions": {
        "modules": {...},
        "actions": {...},
        "accessible_companies": [],
        "accessible_locations": []
    },
    "created_at": ISODate,
    "is_active": true
}

// companies
{
    "_id": ObjectId,
    "id": "uuid-string",
    "name": "Company Name",
    "registration_number": "123456",
    "tax_id": "RO12345678",
    "address": "Address",
```

```
    "phone": "Phone",
    "email": "email@example.com",
    "contact_person": "Contact",
    "status": "active",
    "created_at": ISODate,
    "created_by": "user-id"
}

// slot_machines
{
    "_id": ObjectId,
    "id": "uuid-string",
    "cabinet_id": "cabinet-id",
    "game_mix_id": "game-mix-id",
    "provider_id": "provider-id",
    "model": "Model Name",
    "serial_number": "123456",
    "denomination": 0.01,
    "max_bet": 100.0,
    "rtp": 96.5,
    "gaming_places": 1,
    "commission_date": ISODate,
    "invoice_number": "INV-001",
    "status": "active",
    "location_id": "location-id",
    "production_year": 2023,
    "created_at": ISODate,
    "created_by": "user-id"
}

// metrology
{
    "_id": ObjectId,
    "id": "uuid-string",
    "serial_number": "123456",
    "certificate_number": "CVT-123456",
    "certificate_type": "calibration",
    "issue_date": ISODate,
    "expiry_date": ISODate,
    "issuing_authority": "ANM",
    "calibration_interval": 12,
    "next_calibration_date": ISODate,
    "status": "active",
    "description": "CVT Certificate",
    "created_at": ISODate,
    "created_by": "user-id"
}
```

6. DEPLOYMENT CONFIGURATION

6.1 Environment Variables

```
# backend/.env
MONGO_URL=mongodb://localhost:27017
DB_NAME=financial_planner_dev
JWT_SECRET=your-secret-key
PORT=8002

# frontend/.env
REACT_APP_BACKEND_URL=http://localhost:8002
```

6.2 Dependencies

```
# backend/requirements.txt
fastapi==0.104.1
uvicorn==0.24.0
motor==3.3.1
bcrypt==4.0.1
PyJWT==2.8.0
python-multipart==0.0.6
```

```
// frontend/package.json
{
  "dependencies": {
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-scripts": "5.0.1"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  }
}
```

7. SECURITY FEATURES

7.1 Authentication & Authorization

- JWT token-based authentication
- Role-based access control (Admin, Manager, Operator)
- Permission-based module access
- Password hashing with bcrypt
- Token expiration (24 hours)

7.2 Data Validation

- Pydantic models for request/response validation
- Input sanitization
- SQL injection prevention (MongoDB)

- XSS protection

8. BUSINESS LOGIC

8.1 Slot Machine Management

- Serial number tracking
- Provider and cabinet relationships
- Game mix assignments
- Location assignments
- Status management (active/inactive)

8.2 Metrology Tracking

- CVT certificate management
- Expiry date calculation
- Days remaining calculation
- Automatic status updates

8.3 Financial Tracking

- Invoice management
- Payment tracking
- Currency handling
- Financial reporting

9. USER INTERFACE FEATURES

9.1 Responsive Design

- Mobile-friendly interface
- Adaptive layouts
- Touch-friendly controls

9.2 Data Visualization

- Dashboard with key metrics
- Status indicators
- Color-coded alerts
- Progress tracking

9.3 Search and Filter

- Global search functionality
- Column-specific filtering
- Sortable tables
- Advanced filtering options

10. ERROR HANDLING

10.1 Backend Error Handling

```
@app.exception_handler(HTTPException)
async def http_exception_handler(request, exc):
    return JSONResponse(
        status_code=exc.status_code,
```

```
        content={"detail": exc.detail}
    )

@app.exception_handler(Exception)
async def general_exception_handler(request, exc):
    return JSONResponse(
        status_code=500,
        content={"detail": "Internal server error"}
    )
```

10.2 Frontend Error Handling

```
// Error boundary component
class ErrorBoundary extends React.Component {
    constructor(props) {
        super(props);
        this.state = { hasError: false };
    }

    static getDerivedStateFromError(error) {
        return { hasError: true };
    }

    componentDidCatch(error, errorInfo) {
        console.error('Error caught by boundary:', error, errorInfo);
    }

    render() {
        if (this.state.hasError) {
            return <h1>Something went wrong.</h1>;
        }
        return this.props.children;
    }
}
```

11. TESTING STRATEGY

11.1 Backend Testing

- Unit tests for utility functions
- Integration tests for API endpoints
- Database connection tests
- Authentication tests

11.2 Frontend Testing

- Component unit tests
- Integration tests for user flows
- E2E tests for critical paths
- Accessibility testing

12. MONITORING AND LOGGING

12.1 Application Logging

```
import logging

logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
)
logger = logging.getLogger(__name__)

# Usage in endpoints
logger.info(f"User {user.username} logged in")
logger.error(f"Database connection failed: {error}")
```

12.2 Performance Monitoring

- API response time tracking
- Database query optimization
- Memory usage monitoring
- Error rate tracking

13. BACKUP AND RECOVERY

13.1 Database Backup

```
# MongoDB backup script
mongodump --db financial_planner_dev --out /backup/$(date +%Y%m%d)
```

13.2 Data Recovery

- Point-in-time recovery
- Selective data restoration
- Schema migration support

14. SCALABILITY CONSIDERATIONS

14.1 Database Optimization

- Index creation for frequently queried fields
- Connection pooling
- Query optimization
- Data archiving strategies

14.2 Application Scaling

- Horizontal scaling with load balancers
- Caching strategies (Redis)
- CDN for static assets
- Microservices architecture potential

15. COMPLIANCE AND REGULATIONS

15.1 Gaming Regulations

- ONJN compliance
- Audit trail maintenance
- Data retention policies
- Security standards adherence

15.2 Data Protection

- GDPR compliance
- Data encryption
- Access logging
- Privacy controls

Această documentație oferă o privire completă asupra sistemului CASHPOT Gaming Management, incluzând toate aspectele tehnice, arhitecturale și funcționale necesare pentru reconstruirea aplicației într-un alt sistem.