Ene Matei – Emanuel
342A3

INTEROPERABILITATEA SISTEMELOR INFORMAȚIONALE – PROIECT

Domeniul ales: evidența consultațiilor la o clinică medicală

Ene Matei – Emanuel 342A3

1. Analiza domeniului

Definire: Acest domeniu implică gestionarea informațiilor despre pacienți, medici, consultațiile efectuate. Diagnosticele puse și tratamentele prescrise într-o clinică.

Sistemul va gestiona informații despre consultațiile medicale dintr-o clinica. Entitățile principale sunt: Pacient, Medic, Consultație, Diagnostic, Tratament.

Principalele constrângeri:

- Fiecare pacient are un ID unic (ex: CNP sau ID intern), nume, prenume, data nașterii.
- Fiecare medic are un ID unic, nume, prenume, specializare.
- O consultație este efectuată de un singur medic pentru un singur pacient la o anumită dată și oră.
- O consultație poate avea asociat un diagnostic (cod și descriere) și un tratament (descriere, medicamente prescrise).
- Data consultației trebuie să fie validă.
- Codul de diagnostic poate urma un standard (ex: ICD-1).
- Specializarea medicului poate fi aleasă dintr-o listă predefinită (ex: Cardiologie, Dermatologie, Pediatrie).
- Un pacient poate avea multiple consultații.
- Un medic poate efectua multiple consultații.

Proiectarea și Crearea Documentului XML (consultatii.xml):

- Structura ierarhică: Un element rădăcină <Clinica>, care conține liste de <Pacienti>, <Medici> și <Consultatii>.
- Elementul <Consultatie> va conține informații proprii (data, ora) și referințe (ID Pacient, ID Medic), plus sub-elemente pentru <Diagnostic> și <Tratament>.
- **Niveluri:** Asigură cel puțin 3 niveluri (ex: <Clinica> -> <Consultatii> -> <Consultatie> -> <Diagnostic>).

- **Atribut:** Include cel puțin un atribut (ex: <Pacient id="P001">, <Medic id="M001">, <Consultatie id_consultatie="C001">).

Proiectarea și Crearea Documentului JSON (consultatii.json):

- Structura similară cu XML, folosind obiecte și array-uri.
- **Niveluri**: Asigură cel puțin 4 niveluri (ex: root -> consultatii -> [index] -> tratament -> medicamente -> [index]).

Proiectarea Schemei XML (consultatii.xsd):

- Definește structura consultatii.xml (elemente, atribute, tipuri de date).
- Include cel puțin o restricție (xs:restriction):
    - Ex: Restricționează valorile pentru <Specializare> la o listă predefinită (<xs:enumeration value="Cardiologie"/>, etc.).
    - Ex: Definește un pattern pentru id_pacient (ex: P urmat de cifre).
    - Ex: Restricționează DataNasterii sau Data la tipul xs:date.
- Include cel puțin o extensie (xs:extension):
    - Ar putea fi mai dificil aici. O variantă ar fi să ai un tip complex Persoana cu nume/prenume și să extinzi acest tip pentru Pacient (adăugând data nașterii) și Medic (adăugând specializarea). Sau poate un tip TratamentSimplu (doar indicații) extins la TratamentComplex (cu medicamente).
- Definește cardinalitatea (ex: minOccurs="0", maxOccurs="unbounded").

Proiectarea Schemei JSON (consultatii.schema.json):

- Definește structura consultatii.json folosind specificația JSON Schema.
- Definește tipurile de date (string, number, object, array, boolean).
- Specifică câmpurile obligatorii (required).
- Poate include pattern-uri (pattern), formate (format: date-time), enumerații (enum).

Crearea Foii de Stil XSL (consultatii.xsl):

- Scop: Transformarea consultatii.xml într-un tabel HTML pentru afișare lizibilă (conform cerinței 2).
- Va conține template-uri (xsl:template) care se potrivesc elementelor din XML (match="/", match="Consultatie", etc.).

- Va genera tag-uri HTML (<table>, <tr>, <th>, <td>).
- Va extrage valori din XML folosind xsl:value-of select="...".

2. Implementarea Aplicatiei cu Interfata Vizuala

Aplicația, dezvoltată în Python folosind biblioteca Tkinter pentru interfața grafică, servește ca un instrument pentru gestionarea datelor despre consultațiile medicale stocate în formate XML și JSON. Ea permite încărcarea, vizualizarea, validarea, transformarea (pentru XML via XSLT) și modificarea (CRUD - Create, Read, Update, Delete) acestor date într-un mod interactiv.

Proiectarea interfeței vizuale si implementarea acesteia:

- **Tehnologie:** S-a utilizat biblioteca standard **Tkinter** din Python, împreună cu modulul ttk pentru widget-uri cu aspect modern.
- **Structură:**
    - Fereastra principală este împărțită vertical folosind un ttk.PanedWindow, permițând redimensionarea zonelor.
    - **Zona Superioară (Controale):** Conține butoane (ttk.Button) aranjate într-un grid pentru operațiuni principale:
        - Încărcare fișiere (Încarcă XML, Încarcă JSON).
        - Validare (Validează XML (XSD), Validează JSON (Schema)).
        - Transformare (Afișează Tabel (XSLT)).
        - Salvare modificări (Salvează Modificări).
    - **Zona Centrală (Afișare Date):** Dominată de un widget ttk.Treeview. Acesta este configurat să afișeze datele într-o structură arborescentă (pentru ierarhie) și cu coloane (pentru atribute/valori specifice). Include bare de derulare (scrollbars) pentru navigare ușoară.
    - **Zona Inferioară (Operații CRUD & Status):**
        - O secțiune ttk.LabelFrame grupează controalele pentru operațiile pe consultații:
            - Căutare (ttk.Entry pentru ID, ttk.Button "Caută").
            - Adăugare (ttk.Button "Adaugă Consultație Nouă").
            - Ștergere (ttk.Button "Șterge Consultația Selectată").

        - O etichetă ttk.Label în partea de jos acționează ca o **bară de status**, afișând mesaje despre operațiunea curentă sau eventuale erori.

- **Comportament:** Butoanele sunt activate/dezactivate (state=tk.NORMAL/tk.DISABLED) în funcție de context (ex: butoanele XML sunt active doar dacă un fișier XML este încărcat).

Încărcarea documentelor XML si JSON cu ajutorul interfeței:

- **Trigger:** Apăsarea butoanelor Încarcă XML sau Încarcă JSON.
- **Proces:**
    1. Se deschide o fereastră de dialog standard (tkinter.filedialog.askopenfilename) care permite utilizatorului să navigheze și să selecteze fișierul dorit (.xml sau .json).
    2. Dacă utilizatorul selectează un fișier, calea către acesta este reținută. Dacă anulează, operațiunea se oprește.
    3. **Gestionarea Erorilor:** Se include tratarea excepțiilor pentru FileNotFoundError (dacă fișierul nu mai există între timp) și pentru erori specifice de parsare (vezi mai jos).

Parsarea Documentelor XML si JSON (parcurgerea elementelor/atributelor):

- **Scop**: Transformarea conținutului fișierului text (XML sau JSON) într-o structură de date manipulabilă în memorie (în Python).
- **Parsarea XML (load_xml function):**
    1. Biblioteca: Se utilizează lxml.etree, care este eficientă și necesară pentru validarea XSD și transformarea XSLT ulterioară.
    2. Metoda: Funcția etree.parse(filepath) citește fișierul XML și construiește un arbore de elemente (ElementTree) în memorie (self.xml_tree). Acesta reprezintă structura ierarhică a documentului.
    3. Parcurgerea: După parsare, funcția populate_tree_from_xml este apelată. Aceasta parcurge arborele pentru a extrage datele:
        - Se obține elementul rădăcină (.getroot()).
        - Se folosesc expresii XPath (ex: root_element.xpath('//Pacient'), root_element.xpath('//Consultatie')) pentru a localiza și itera prin colecții de elemente specifice (toți pacienții, toate consultațiile etc.). XPath permite navigarea eficientă indiferent de locația exactă în ierarhie.

        - Pentru fiecare element găsit (ex: un <Consultatie>):
            - Se extrag valorile atributelor folosind metoda .get('nume_atribut') (ex: consult.get('id_consultatie')).
            - Se extrage conținutul textual al elementelor copil folosind .findtext('nume_copil') sau expresii XPath mai specifice (ex: consult.xpath('./Diagnostic/CodICD10/text()')).
    4. Erori: Dacă fișierul XML nu este bine formatat, etree.parse va ridica o excepție etree.XMLSyntaxError, care este prinsă și afișată utilizatorului printr-un messagebox.

- **Parsarea JSON (load_json function):**
    1. Biblioteca: Se utilizează modulul standard json din Python.
    2. Metoda: Fișierul este deschis în mod text (open(filepath, 'r')), iar funcția json.load(f) citește conținutul și îl convertește într-o structură de date nativă Python (de obicei un dicționar - dict - și liste - list) - self.json_data.
    3. Parcurgerea: După parsare, funcția populate_tree_from_json este apelată. Aceasta parcurge structura de dicționare și liste:
        - Se accesează cheile principale (ex: self.json_data['clinica']).
        - Se iterează prin listele care conțin obiectele (ex: for pacient in clinica_data.get('pacienti', [])). Folosirea .get() cu o valoare default ([]) previne erorile dacă o cheie lipsește.
        - Pentru fiecare obiect (dicționar) din listă (ex: un dicționar reprezentând o consultație):
            - Se accesează valorile asociate cheilor folosind obiect.get('nume_cheie', default_value) (ex: consult.get('id_consultatie'), consult.get('diagnostic', {}).get('codICD10')). Se folosește .get() pentru a accesa în siguranță și cheile din obiectele imbricate.
    4. Erori: Dacă fișierul JSON are o sintaxă invalidă, json.load va ridica o excepție json.JSONDecodeError, care este prinsă și afișată.

Ene Matei – Emanuel
342A3

Afișarea datelor stocate in documente (Treeview):

- **Componenta**: Widget-ul ttk.Treeview (self.tree).
- **Proces**: Funcțiile populate_tree_from_xml și populate_tree_from_json, după ce au extras datele prin parsare (conform descrierii de mai sus), populează vizual Treeview-ul:
    1. Curățare: Mai întâi, orice elemente existente în Treeview sunt șterse (self.clear_tree_items).
    2. Inserare Noduri: Se folosește metoda self.tree.insert():
        - Se creează noduri părinte pentru categoriile principale ("Pacienți", "Medici", "Consultații").
        - Pentru fiecare element parcurs (pacient, medic, consultație), se inserează un nod copil sub categoria corespunzătoare.
        - Parametri importanți ai insert:
            - parent: ID-ul nodului părinte.
            - index: tk.END (adaugă la sfârșit).
            - text: Textul afișat în prima coloană (ex: "Consult. C001").
            - values: O tuplă cu valorile de afișat în celelalte coloane definite (Nume/Data, Prenume/Ora etc.).
            - iid (Item ID): Pentru rândurile de consultații, se folosește ID-ul unic al consultației (ex: "C001", "C002") ca identificator intern (iid) în Treeview. Acest lucru este esențial pentru a putea referenția și manipula ulterior un rând specific (la căutare sau ștergere).
- **Rezultat**: Utilizatorul vede datele structurate într-un tabel ierarhic, ușor de navigat.

Utilizarea XSL pentru afișarea datelor XML intr-un format de tabel (display_xslt):

- **Scop**: Transformarea datelor din fișierul XML încărcat într-un format HTML (un tabel), folosind o foaie de stil XSLT.
- **Trigger**: Butonul Afișează Tabel (XSLT) (activ doar după încărcarea XML).
- **Proces**:
    1. Se identifică fișierul XSL (.xsl). Aplicația caută implicit consultatii.xsl sau întreabă utilizatorul.

2. Se parsează fișierul XSL folosind lxml.etree.parse.
3. Se creează un obiect transformator XSLT: transformer = etree.XSLT(xslt_doc).
4. Se aplică transformarea asupra arborelui XML din memorie: html_result_tree = transformer(self.xml_tree).
5. Rezultatul transformării (care este tot un arbore lxml) este convertit într-un string HTML formatat: etree.tostring(html_result_tree, ..., method="html").
6. String-ul HTML este salvat într-un fișier local (consultatii_output.html).
7. Fișierul HTML generat este deschis automat în browser-ul web implicit al utilizatorului folosind modulul webbrowser.

- **Rezultat**: Utilizatorul vede datele consultațiilor (conform definiției din XSL) într-un tabel HTML lizibil în browser.

Posibilitatea introducerii, ștergerii si căutării datelor din fișier (operații CRUD – in memorie):

- **Important**: Aceste operații modifică structura de date din memorie (self.xml_tree sau self.json_data). Modificările devin permanente doar după apăsarea butonului Salvează Modificări.
- **Căutare** (search_consultation):
  1. Utilizatorul introduce un ID (ex: "C001") în câmpul de căutare și apasă "Caută".
  2. Aplicația verifică dacă un item cu iid-ul respectiv există în self.tree folosind self.tree.exists(search_id).
  3. Dacă există, rândul corespunzător este selectat (selection_set), focalizat (focus) și adus în vizualizare (see).
  4. Dacă nu există, se afișează un mesaj informativ.
- **Introducere/Adăugare** (add_consultation_dialog, add_consultation_data):
  1. Butonul "Adaugă Consultație Nouă" deschide o fereastră de dialog (Toplevel) cu câmpuri de introducere (ttk.Entry) pentru datele unei noi consultații (ID Pacient, ID Medic, Data, Ora, Simptome etc.). ID-ul noii consultații este generat automat (ex: Cxxx).
  2. După completarea și validarea simplă a datelor, funcția add_consultation_data este apelată.

3. Aceasta adaugă noua consultație în structura de date din memorie:
    - XML: Creează un nou element <Consultatie> cu sub-elementele corespunzătoare și îl anexează sub elementul părinte <Consultatii> folosind etree.SubElement.
    - JSON: Creează un nou dicționar Python reprezentând consultația și îl adaugă la sfârșitul listei consultatii din dicționarul principal.
4. Se reîmprospătează afișajul din Treeview apelând populate_tree_from_xml/json.

- **Ștergere** (delete_consultation):
    1. Utilizatorul selectează unul sau mai multe rânduri de consultații în Treeview.
    2. La apăsarea butonului "Șterge Consultația Selectată", aplicația preia iid-urile rândurilor selectate.
    3. Se cere confirmarea utilizatorului (messagebox.askyesno).
    4. Pentru fiecare ID de șters:
        - XML: Se localizează elementul <Consultatie> corespunzător folosind XPath (//Consultatie[@id_consultatie="ID_DE_STERS"]), se obține elementul părinte și se folosește parent.remove(element_de_sters).
        - JSON: Se filtrează lista consultatii, păstrând doar acele dicționare al căror id_consultatie *nu* este cel de șters.
    5. Se șterge rândul corespunzător din Treeview folosind self.tree.delete(id_de_sters).

Validarea fișierelor (validate_xsd, validate_json_schema):

- **Scop:** Verificarea conformității fișierului încărcat (XML sau JSON) cu o schemă predefinită (XSD sau JSON Schema).
- **Trigger:** Butoanele Validează XML (XSD) sau Validează JSON (Schema).
- **Proces XSD:**
    1. Se încarcă fișierul XSD (.xsd).
    2. Se creează un obiect XMLSchema din fișierul XSD parsat (etree.XMLSchema).

3. Se apelează xmlschema.assertValid(self.xml_tree). Dacă XML-ul nu respectă schema, această metodă ridică o excepție etree.DocumentInvalid.
4. Se afișează un messagebox cu rezultatul (succes sau detaliile erorii).

- **Proces JSON Schema:**
  1. Se încarcă fișierul JSON Schema (.json).
  2. Se apelează funcția validate(instance=self.json_data, schema=schema) din biblioteca jsonschema. Dacă JSON-ul nu respectă schema, se ridică o excepție jsonschema.exceptions.ValidationError.
  3. Se afișează un messagebox cu rezultatul.

3. Crearea Ontologiei specifice domeniului

Definirea Elementelor Ontologiei:

- **Clase (Classes):** Pacient, Medic, Consultatie, Diagnostic, Tratament, Specializare, Clinica (poate fi clasa de bază sau doar un concept).
- **Proprietăți de Obiect (Object Properties): Relații între clase.**
  - areConsultatie (Domeniu: Pacient, Rang: Consultatie)
  - esteConsultatDe (Inversa lui areConsultatie; Domeniu: Consultatie, Rang: Pacient)
  - efectueazaConsultatie (Domeniu: Medic, Rang: Consultatie)
  - esteEfectuataDe (Inversa; Domeniu: Consultatie, Rang: Medic)
  - areSpecializare (Domeniu: Medic, Rang: Specializare)
  - rezultaInDiagnostic (Domeniu: Consultatie, Rang: Diagnostic)
  - prescrieTratament (Domeniu: Consultatie, Rang: Tratament)
- **Proprietăți de Date (Data Properties): Atribute ale claselor.**
  - numePacient (Domeniu: Pacient, Rang: string)
  - prenumePacient (Domeniu: Pacient, Rang: string)
  - dataNasterii (Domeniu: Pacient, Rang: dateTime sau date)
  - idPacient (Domeniu: Pacient, Rang: string)

        - numeMedic (Domeniu: Medic, Rang: string)
        - dataConsultatie (Domeniu: Consultatie, Rang: dateTime)
        - codDiagnostic (Domeniu: Diagnostic, Rang: string)
        - descriereDiagnostic (Domeniu: Diagnostic, Rang: string)
        - indicatiiTratament (Domeniu: Tratament, Rang: string)
    - **Indivizi (Individuals):** pacient "Ion Popescu", medic "Maria Ionescu", o consultație specifică între ei**.**