# Capitolul 3. PROIECTAREA aplicației pentru planificare inteligentă utilizând tehnici de ai

Proiectarea unei aplicații pentru planificare inteligentă presupune, în cadrul acestui proiect, dezvoltarea unui sistem de generare automată a orarelor universitare, bazat pe tehnici de inteligență artificială. Aceasta implică definirea unei structuri solide, capabile să gestioneze atât componentele clasice, precum frontend, backend și baza de date, cât și integrarea unui sistem bazat pe inteligența artificială pentru generarea automată.

Scopul acestui proiect este să combine interfața intuitivă cu logica avansată de planificare, permițând utilizatorului să introducă date complexe într-un mod simplificat, pentru a obține un orar valid, coerent, complet și generat automat, respectând toate constrângerile impuse.

## 3.1 Arhitectura și funcționalitatea proiectului

Aplicația pentru generarea automată a orarelor are o arhitectură de tip client-server, care permite separarea clară între interfața utilizatorului (partea de frontend), logica aplicației (partea de backend), stocarea datelor (baza de date) și modulul de generare a orarului (integrarea OpenAI GPT-4). (Figura 4)

A diagram of a flowchart

AI-generated content may be incorrect.

#### Fig. 4. Diagrama arhitecturală a aplicației

Arhitectura generală a aplicației:

1. Frontend (React.js): Interfața aplicației este dezvoltată în React.js și oferă utilizatorului o experiență interactivă, ușor de utilizat, cu pagini dedicate pentru:

* Introducerea grupelor/ subgrupelor, în funcție de an sau nivelul de studiu;
* Gestionarea sălilor (Săli de curs/ laborator/ proiect/ seminar);
* Introducerea profesorilor, a disciplinei, a tipului de activitate și a disponibilităților acestora;
* Definirea regulilor de planificare sau accesarea acestora din baza de date;
* Generarea, vizualizarea și exportul orarului.

1. Backend (Flask – Python): Componenta de server gestionează cererile venite din interfață, prelucrează datele și comunică cu:

* baza de date MySQL pentru operații CRUD;
* serviciile AI externe (GPT-4 de la OpenAI) sau algoritm local clasic;
* componentele de validare a orarului și export în formate PDF/Excel.

1. Baza de date (MySQL): Este utilizată pentru stocarea datelor aplicației: profesori, discipline, grupe, săli, reguli definite, orare generate. Structura bazei de date este relațională și asigură integritatea datelor prin relații bine definite.
2. Modul AI: Modulul de generare a orarului folosește tehnici avansate de inteligență artificială generativă (prin integrarea OpenAI GPT-4) pentru a crea orare valide, respectând regulile definite de utilizator. Alternativ, aplicația suportă și un algoritm clasic propriu, scris în Python.

Funcționalitățile principale ale aplicației sunt:

1. Introducerea datelor: utilizatorul adaugă informații despre grupe/subgrupe, tipul sălilor, detalii despre profesori (disciplina predată, disponibilitatea acestora), direct din interfață.
2. Definirea regulilor: utilizatorul personalizează regulile pentru generare sau folosește o regulă deja stabilită pe care o poate actualiza sau folosi ca atare.
3. Generarea orarului: utilizatorul selectează nivelul și anul pentru care vrea să genereze și ce tip de generare dorește (generarea cu AI sau cu algoritm clasic).
4. Vizualizarea orarului: orarul este afișat într-un format tabelar.
5. Validarea și exportul: orarul poate fi verificat, validat și exportat în format PDF/Excel pentru utilizarea ulterioară.

## 3.2 Diagrame UML pentru modelarea structurii

Diagramele UML (Unified Modeling Language) sunt reprezentări grafice standardizate utilizate pentru a vizualiza, specifica, construi și documenta componentele unui sistem software. Acestea ajută la înțelegerea arhitecturii aplicației și facilitează comunicarea între membrii echipei de dezvoltare.[6]

### 3.2.1. Diagrama pachetelor

Diagramele de pachete sunt utilizate pentru a reprezenta organizarea logică a componentelor unui sistem software, grupate în module sau pachete. Ele evidențiază dependențele dintre module și oferă o imagine de ansamblu asupra structurii proiectului.[7]

În cadrul aplicației dezvoltate, au fost create mai multe diagrame de pachete pentru:

1. Backend-ul aplicației: este organizat modular, respectând principiile separării responsabilităților. (Figura 5) Structura generală include:

* routes/ – pentru definirea rutelor RESTful;
* database/ – responsabil pentru conectarea la baza de date (ex. db\_connection.py);
* logica/ – conține logica de generare și validare a orarului (ex. orar\_generator.py, algoritm\_clasic.py, validare.py);
* templates/ – pentru șabloanele HTML utilizate la redare server-side (dacă este cazul);
* tests/ – conține testele unitare și funcționale;
* fișierul principal app.py, care inițializează aplicația Flask și înregistrează Blueprintrile.

A diagram of a diagram

AI-generated content may be incorrect.

#### Fig. 5. Diagrama pachetelor- Backend

1. Frontend-ul aplicației este structurat în directorul principal src/, respectând o organizare modulară clară și scalabilă. (Figura 6) Structura include:

* Pages/ – conține componentele principale ale aplicației precum Home.jsx, Grupe.jsx, Profesori.jsx, SetareReguli.jsx etc., care corespund fiecărei pagini vizibile în interfață.
* FunctiiLogice/ – include logica specifică fiecărui domeniu (Auth/, Dashboard/, Orar/, Utils/), separând clar codul funcțional de cel de prezentare.
* Tests/ – pentru testele unitare sau de integrare aplicabile componentelor frontend.
* Fișierele principale App.jsx, main.jsx, index.css, App.css care definesc stilizarea globală, rutarea și inițializarea aplicației.

A diagram of a company

AI-generated content may be incorrect.

#### Fig.6 Diagrama pachetelor- Frontend

1. **Modulele de rutare**: În cadrul backend-ului, rutele API sunt organizate modular în fișiere dedicate, fiecare gestionând o anumită componentă a aplicației (Figura 7):

* auth\_routes.py - autentificare și înregistrare;
* orar\_routes.py - manipularea și vizualizarea orarelor;
* generator\_routes.py - generarea efectivă a orarului pe baza regulilor și a datelor transmise, cu ajutorul GPT-4 sau algoritmului clasic;
* grupe\_routes.py, profesori\_routes.py, reguli\_routes.py, sali\_routes.py - pentru gestionarea grupelor, profesorilor, regulilor și sălilor.

A diagram of a route

AI-generated content may be incorrect.

#### Fig. 7 Diagrama pachetelor- Routes

1. **Structura paginilor React (Pages)**: Directorul Pages conține toate componentele de interfață care corespund celor mai importante funcționalități ale aplicației (Figura 8):

* Dashboard.jsx: pagina principală de administrare pentru utilizatorul autentificat;
* GeneratedTimetable.jsx: vizualizarea orarului generat, cu opțiuni de filtrare, export și salvare;
* Grupe.jsx: gestiunea manuală a grupelor și subgrupelor, în funcție de nivel și an;
* Profesori.jsx: introducerea și modificarea informațiilor despre profesori și discipline;
* Sali.jsx: introducerea sălilor disponibile, pe categorii (curs, seminar, laborator, proiect);
* SetareReguli.jsx: formularul pentru configurarea regulilor AI pentru generarea orarului;
* Home.jsx: prima pagină accesibilă înainte de login;
* Login.jsx și Register.jsx: autentificarea și înregistrarea utilizatorilor în aplicație.

A diagram of a computer program

AI-generated content may be incorrect.

#### Fig. 8 Diagrama pachetelor- Pages

### 3.2.2 Diagrama claselor

Diagrama de clase oferă o reprezentare statică a principalelor entități din cadrul sistemului backend. Aceasta reflectă structura internă a aplicației și relațiile dintre componentele logice implicate în procesul de generare a orarului universitar (Figura 9).

În figura de mai jos sunt prezentate clasele esențiale ale backend-ului:

* OrarGenerator – responsabilă pentru procesul de generare a orarului pe baza criteriilor stabilite de utilizator și a datelor preluate din baza de date;
* AlgoritmClasic – o alternativă deterministă pentru alocarea activităților educaționale;
* ValidatorOrar – verifică sincronizarea activităților de tip curs;
* Database – abstractizează accesul la baza de date și metodele de interogare.

Diagrama evidențiază atât atributele esențiale, cât și metodele publice ale fiecărei clase. Relațiile de compoziție și utilizare sunt marcate vizual prin săgeți și mențiuni de tip „folosește” sau „accesează date”.

A diagram of a computer

AI-generated content may be incorrect.

#### Fig. 9 Diagrama clasei -Backend

Diagrama de mai jos modelează structura logică a componentelor frontend responsabile pentru **configurarea regulilor** de generare și **afișarea orarului** rezultat. Aceasta reflectă atât componentele vizuale de tip React JSX (precum SetareReguli, GeneratedTimetable), cât și hook-urile logice personalizate (useSetariReguli, useGeneratedTimetable, useOrarGenerator, useValidareOrar, useExportOrar) (Figura 10).

Fiecare clasă (funcție sau componentă) conține:

* atributele relevante pentru starea locală,
* metodele utilizate în fluxul aplicației.

Relațiile dintre clase sunt exprimate prin acțiuni precum:

* „utilizează” (ex: SetareReguli utilizează useSetariReguli),
* „transmite reguli”,
* „folosește” (ex: useGeneratedTimetable folosește celelalte hook-uri auxiliare).

Această diagramă oferă o privire clară asupra modularizării logice a aplicației în React și a relațiilor dintre componentele funcționale.

A diagram of a company

AI-generated content may be incorrect.

#### Fig. 10 Diagrama claselor- Frontend

### 3.2.3 Diagrama de componente

Diagrama de componente ilustrează arhitectura aplicației dintr-o perspectivă structurală, evidențiind relațiile dintre modulele frontend, backend și baza de date (Figura 11). Aceasta ajută la înțelegerea modului în care este realizată comunicarea între componentele aplicației și traseul parcurs de date, de la interfața utilizator până la persistarea în baza de date.

În figura următoare, se evidențiază:

* Frontend-ul realizat în React, cu componente precum App.jsx, Grupe.jsx, Sali.jsx, SetareReguli.jsx și GeneratedTimetable.jsx – fiecare trimițând cereri către backend prin intermediul rutei dedicate.
* Backend-ul implementat în Flask, modularizat în rute (\*\_routes.py), care intermediază accesul la baza de date.
* Baza de date MySQL, unde fiecare entitate are asociată o tabelă (GrupeTable, SaliTable, ReguliTable, OrarTable), iar rutele backend comunică direct cu acestea pentru operații CRUD.

Săgețile din diagramă indică direcția de comunicare: de la frontend spre backend și mai departe spre baza de date.

A diagram of a computer

AI-generated content may be incorrect.

#### Fig. 11 Diagrama de componente ale aplicației

## 3.3 Diagrame UML pentru modelarea comportamentului

Pentru a surprinde comportamentul sistemului în raport cu utilizatorul și cu procesele interne, au fost utilizate mai multe diagrame UML comportamentale. Acestea evidențiază fluxurile logice, interacțiunile dinamice dintre componente și cazurile de utilizare ale aplicației.

### 3.3.1 Diagrama cazurilor de utilizare

Această diagramă evidențiază principalele acțiuni pe care le poate efectua un utilizator autenticat în sistemul de generare automată a orarului universitar. Este o reprezentare abstractă a funcționalităților oferite de aplicație și a interacțiunii directe cu sistemul:

* Autentificare – validarea identității utilizatorului pentru acces securizat;
* Configurare reguli orar – definirea parametrilor pentru generarea orarului (zile active, intervale, preferințe);
* Generare orar – inițierea procesului de creare automată a orarului pe baza regulilor stabilite;
* Vizualizare orar – afișarea grafică a orarului generat în interfața React;
* Export orar PDF/Excel – descărcarea orarului în formate utile pentru tipărire sau partajare.

A diagram of a network

AI-generated content may be incorrect.

#### Fig. 12 Diagrama cazurilor de utilizare

### 3.3.2 Diagrama de secvență

Această diagramă surprinde ordinea temporală a mesajelor transmise între componentele aplicației în cadrul procesului de generare a orarului, folosind modelul AI. Actorul principal – utilizatorul – declanșează acțiunea, iar fluxul este gestionat secvențial între interfața React, backend-ul Flask, modulul de generare a orarului, serviciul AI (OpenAI API) și baza de date MySQL.

Pașii esențiali reprezentați sunt:

* Utilizatorul solicită generarea orarului;
* Frontend-ul React trimite cererea către backend;
* Backend-ul Flask pregătește promptul și regulile pentru AI;
* OrarGenerator trimite cererea către modelul AI;
* OpenAI API răspunde cu orarul generat;
* Orarul este salvat în baza de date;
* Backend-ul returnează răspunsul către frontend;
* Frontend-ul afișează orarul pentru utilizator.

A diagram of a company

AI-generated content may be incorrect.

#### Fig. 13 Diagrama de secvență

### 3.3.3 Diagrama de activitate

Această diagramă de activitate ilustrează fluxul logic al pașilor principali parcurși în aplicație pentru a genera un orar valid. Este o reprezentare grafică a procesului decizional și operațional, pornind de la configurarea regulilor și până la afișarea sau exportul rezultatului final.

* Pașii descriși în diagramă:
* Start – inițializarea procesului.
* Configurare reguli orar – utilizatorul introduce regulile de generare.
* Transmiterea regulilor la backend – aceste reguli sunt trimise către server.
* Preluarea datelor – sistemul colectează informațiile necesare din baza de date (săli, profesori, grupe etc.).
* Generarea orarului – se apelează algoritmul AI (GPT) sau o alternativă euristică.
* Validarea orarului – verificarea constrângerilor și a consistenței orarului.
* Dacă orarul este valid, se afișează și utilizatorul poate opta pentru export PDF/Excel.
* Dacă orarul este invalid, se trimite un mesaj de eroare.
* Stop – finalul procesului.

A diagram of a company

AI-generated content may be incorrect.

#### Fig. 14 Diagrama de activitate