

Escola de Programació - Python C1

Treball Final: Sistema OdontoCare

Introducció

L'objectiu principal d'aquest projecte és integrar els diferents continguts del curs i aplicar-los al desenvolupament d'una solució backend completa i funcional.

Per a això, l'estudiant haurà d'implementar un sistema que combini els següents components fonamentals:

- **Framework Backend:** Desenvolupament d'una API REST utilitzant **Flask**, organitzada de forma professional mitjançant **Blueprints** per assegurar modularitat i escalabilitat.
- **Persistència de Dades:** Ús d'una base de dades **SQLite**, gestionada a través de **SQLAlchemy** com a ORM per modelar entitats, relacions i operacions CRUD¹.
- **Seguretat:** Implementació d'un mecanisme d'**autenticació basat en tokens**, garantint l'accés segur als diferents recursos del sistema.
- **Client Extern:** Creació d'un **script independent en Python** que consumeixi els serveis de l'API utilitzant la biblioteca **requests**, demostrant la correcta interacció entre client i servidor.
- **Arquitectura Distribuïda i Comunicació entre Serveis:** Crear imatges en docker.

Aquest objectiu busca consolidar les competències del nivell C1, cosa que permet a l'estudiant demostrar la seva capacitat per dissenyar, desenvolupar i integrar un backend complet amb enfocament professional.

Escenari del Projecte

Una xarxa de clíniques dentals ha decidit modernitzar les seves operacions creant una aplicació a mida per gestionar les cites dels pacients i la disponibilitat dels odontòlegs.

Actualment, el sistema es gestiona de forma manual, la qual cosa provoca errors freqüents, duplicitat d'informació i falta de traçabilitat en els processos administratius.

Com a desenvolupador backend assignat al projecte, la teva missió consisteix a dissenyar i construir una **solució integral, robusta i escalable**, que permeti cobrir totes les necessitats del nou sistema de gestió **OdontoCare**.

Per a això, es requereix el desenvolupament d'una **API RESTful professional**, seguint

bones pràctiques d'arquitectura de software, seguretat i persistència de dades.

El sistema ha de permetre l'administració eficient de la informació mitjançant els següents mòduls essencials:

- **Pacients**
- **Doctors**
- **Centres mèdics o clíniques**
- **Cites mèdiques**

Tota la informació gestionada per l'API ha de persistir en una base de dades fiable.

A més, l'accés als recursos ha d'estar controlat mitjançant un mecanisme d'**autenticació basat en tokens (JWT o similar)**, garantint que només els usuaris autoritzats puguin interactuar amb les dades.

El format de comunicació de tots els serveis serà exclusivament **JSON**, per la qual cosa cada endpoint ha de respondre consistentment en aquest format, tant en operacions exitoses com en gestió d'errors.

L'estudiant haurà de definir i organitzar adequadament l'estructura del projecte.

Adicionalment, haurà d'incloure un arxiu **requirements.txt** per a cada projecte, en el qual s'especifiquin totes les llibreries i dependències necessàries, incorporant aquelles que consideri pertinents per al correcte desenvolupament de l'activitat.

Objectius principals del sistema

1. Dissenyar una API RESTful organitzada, modular i mantenible.
2. Implementar operacions CRUD per a pacients, doctors, centres i cites.
3. Garantir la persistència de la informació en una base de dades (SQL o NoSQL).
4. Incorporar un sistema d'autenticació segura per tokens.
5. Assegurar que totes les respostes s'entreguin en format JSON.
6. Aplicar bones pràctiques com validació de dades, gestió d'excepcions, paginació i documentació bàsica de l'API.
7. Implementar una arquitectura distribuïda basada en contenidors Docker.

Arquitectura de la Solución

Per garantir un desenvolupament ordenat, escalable i alineat amb bones pràctiques d'enginyeria de software, l'API s'ha d'implementar utilitzant una **arquitectura modular basada en Blueprints de Flask**.

Això permetrà separar la lògica del sistema per dominis funcionals, facilitant el seu manteniment, comprensió i reutilització.

La solució **no ha de concentrar tot el codi en un sol arxiu**. En el seu lloc, s'exigeix una estructura organitzada que distribueixi la lògica en mòduls clars i coherents.

L'API s'haurà d'estructurar, com a mínim, amb els següents components:

auth_bp — Autenticació i Gestió d'Usuaris

Encarregat de totes les operacions relacionades amb l'accés segur al sistema. Ha d'incloure:

- Registre d'usuaris autoritzats.
- Inici de sessió mitjançant validació de credencials.
- Generació i validació de **tokens d'autenticació (JWT)**.
- Gestió d'errors d'accés.

Aquest mòdul garanteix que totes les accions dins del sistema siguin realitzades només per usuaris autenticats.

admin_bp — Administració i Gestió de Centres, Pacients i Doctors

Mòdul orientat a tasques administratives, encarregat de configurar els elements base del sistema. Ha d'incloure:

- Creació d'entitats principals: centres mèdics, pacients i doctors.
- Càrrega de dades, tant massiva com individual, utilitzant arxius en format JSON quan sigui requerit.
- Opcions de consulta per a tots els tipus de registres, permetent:
 - Cerca individual per ID.
 - Visualització opcional d'una llista completa de registres.

Aquest mòdul està dissenyat per a usuaris amb rols administratius o de gestió.

cites_bp — Gestió Operativa de Cites

Responsable del nucli funcional d'OdontoCare: la planificació, administració i control de cites mèdiques. Ha d'incloure:

- Creació, actualització, consulta i eliminació de cites.
- Validació de disponibilitat de doctors i centres.
- Regles operatives per evitar conflictes a l'agenda.
- Respostes en format JSON amb missatges clars i estructurats.

Aquest mòdul serà el més utilitzat durant l'operació diària del sistema.

Model de Dades (SQLAlchemy)

Has de definir almenys les següents taules/models. Pot ser necessària la creació d'estructures addicionals al model; això queda a llibertat de l'estudiant.

Usuario

1. id_user (PK)
2. username
3. password
4. rol (admin, metge, secretari/a, pacient)

Pacient

Dades mínimes del pacient:

- id_pacient (PK)
- id_user (FK opcional)
- nom
- telèfon
- estat(ACTIVO/INACTIVO)

Doctor

- id_doctor (PK)
- id_user (FK opcional)
- nom
- especialitat

Centro Mèdic

- id_centre (PK)
- nom
- adreça

Cita Mèdica

Relaciona pacient, doctor i centre:

- id_cita (PK)
- data
- motiu
- estat
- id_pacient (FK)
- id_doctor (FK)
- id_centre (FK)
- id_usuari_registra (FK)

Funcionalitats Detallades

Autenticació i Seguretat

El sistema no pot confiar en un simple camp JSON. Ha d'incloure:

- **POST /auth/login**
- Verificació d'usuari i contrasenya.
- Retorn d'un token vàlid.
- Enviament obligatori del token al header:

Authorization: Bearer <token>

- Tots els endpoints protegits han de validar aquest token.

Càrrega Inicial de Dades — Múltiples Endpoints

El sistema ha de permetre carregar dades des d'un arxiu local **CSV**, però:

- El servidor NO rep l'arxiu CSV.
- El client processa l'arxiu i envia registre per registre.

Endpoints:

- **crear usuari** POST /admin/usuari
 - Rol requerit: Admin. Registra un usuari (i crea un usuari amb rol "admin" o "secretaria").
- **crear doctors** POST /admin/doctors
 - Rol requerit: Admin. Registra un doctor (i crea un usuari amb rol "metge").
- **crear pacient** POST /admin/pacients
 - Rol requerit: Admin. Registra un pacient (i crea un usuari amb rol "pacient").
- **crear centre** POST /admin/centres
 - Rol requerit: Admin. Registra un centre mèdic.

Gestió de Cites (Core)

Les regles de negoci han de consultar la base de dades.

Pot ser necessari afegir altres mètodes addicionals per complir l'activitat; l'estudiant haurà d'incloure aquells endpoints addicionals.

Agendar Cita — POST /cites

- Rols: Client i Admin
- Validacions obligatòries:
 - a. El **doctor** existeix.
 - b. El **centre mèdic** existeix.
 - c. El **pacient** existeix i està actiu.
 - d. No es pot agendar una cita si el doctor ja en té una altra en la mateixa data i hora (**evitar doble reserva**).

Llistar Cites — GET /cites

- Doctor: només veu les seves pròpies cites.
- Secretaria: pot consultar cites filtrant per data.
- Admin: pot filtrar per doctor, centre, data, estat o pacient.
- S'usen query params per aplicar els filtres.

Cancel·lar Cita — PUT /cites/<id>

- Rols permesos: Secretaria i Admin.
- Validacions:
 - La cita existeix.
 - No està cancel·lada.
- Es canvia l'estat a "Cancelada" i es retorna un missatge JSON confirmant l'acció.

Arxiu de Prova (datos.csv)

L'estudiant ha de crear un arxiu **dades.csv** amb informació fictícia que permeti:

- Carregar doctors
 - Carregar pacients
 - Carregar centres
- Aquest arxiu serà llegit pel client i enviat a l'API.

Client Python

L'estudiant ha de lliurar un script: **carga_inicial.py**, que:

- Realitzi login amb un usuari admin (del CSV).
- Processi i enviï els registres de l'arxiu **dades.csv**.
- Creï una cita mèdica.
- Imprimeixi a la consola el JSON amb la cita creada.

Arquitectura Distribuïda i Comunicació entre Serveis

A més dels mòduls principals del sistema, l'activitat requereix implementar una **arquitectura distribuïda basada en contenidors Docker**, on cada component clau funcioni com un **microservei independent**. L'estudiant haurà de dissenyar i desplegar el sistema OdontoCare com un conjunt de serveis autònoms, cadascun executant-se en el seu propi contenidor Docker i comunicant-se **únicament mitjançant serveis REST**, sense compartir bases de dades ni accés directe entre ells.

Els mòduls se separen de la següent manera:

- **Servei de Gestió d'Usuaris i Registre Administratiu**
- **Servei de Gestió de Cites**

El servei de cites no ha d'accedir directament a les bases de dades dels altres mòduls. En el seu lloc, ha d'obtenir tota la informació necessària únicament mitjançant els **serveis REST** exposats pels altres microserveis. Això garanteix un intercanvi d'informació adequat, segur i coherent amb l'arquitectura distribuïda definida per a l'activitat.

L'estudiant podrà realitzar els ajustos necessaris en el seu disseny per complir amb els requisits de:

- Intercomunicació entre els diferents serveis.
- Independència de bases de dades.
- Aïllament i responsabilitat única de cada mòdul.

Requisits de Lliurament i Demostració

El lliurament final del projecte no només inclou el codi, sinó també la demostració pràctica i l'evidència del funcionament del sistema de microserveis.

Codi Funcional (fork Git)

El requisit fonamental és el lliurament del codi font complet i funcional.

- Plataforma: El codi ha d'estar allotjat en un repositori Git.
- Contingut: Ha d'incloure tots els components del sistema OdontoCare, seguint l'arquitectura distribuïda definida (serveis d'Usuaris i Cites).

L'estudiant ha de desenvolupar i presentar un conjunt de **scripts** que demostrin de forma pràctica el funcionament dels serveis.

Proves d'Integració (Opcional)

Opcionalment es poden incloure o desenvolupar proves d'integració que validin la comunicació entre els serveis i l'accés extern als endpoints.

Les proves d'integració podran incloure qualsevol dels següents mètodes:

- **Scripts** que realitzin crides directes als **endpoints** del servei (usant llibreries HTTP o comandes com curl).
- Implementació de proves unitàries utilitzant **unittest** o el mòdul **flask.testing**.

Documentació de proves d'Endpoints

Lliurament de documentació o scripts amb la següent informació clarament indicada per a cada prova d'endpoint:

- **L'Endpoint Utilitzat**: La ruta completa del servei REST.
- **L'Arxiu d'Entrada**: El cos de la sol·licitud enviat, obligatòriament en **format JSON**.

Vídeo Explicatiu

Es requereix una demostració visual i concisa de l'aplicatiu.

Requisit	Detall
Durada Màxima	5 minuts
Contingut	Ha d'evidenciar clarament el funcionament complet de l'aplicatiu, incloent-hi la interacció entre els microserveis.
Funcionament	Mostrar el flux de treball, des de la inicialització fins a la creació d'una cita mèdica, destacant la comunicació RESTful.

Avaluació

La puntuació de cada pregunta és la següent:

- Pregunta 1: 100%

Es valorarà la validesa de la solució i la claredat de l'argumentació.

L'exercici té indicat a l'enunciat el seu pes en la valoració final. Els criteris d'avaluació per avaluar aquest exercici són els següents:

Pregunta	No assolit (C-)	Mínimament assolit (C+)	Assolit (B)	Assolit de forma excel·lent (A)
Pregunta 1	La resposta és incorrecta o no va ser desenvolupada	La resposta és parcialment correcta i està mínimament justificada	La resposta és correcta i descriu certs passos per arribar a la solució, i està justificada	La resposta és correcta i indica tots els passos per arribar a la solució. Es referencia correctament els apunts per justificar la resposta

Format

Es suggereix, amb l'objectiu d'estandarditzar el format, que l'activitat segueixi les següents restriccions:

- El desenvolupament del **vídeo explicatiu és obligatori**; en cas contrari, l'activitat serà considerada **incompleta**. El vídeo ha de formar part del projecte allotjat en Git.
- L'exercici es desenvoluparà a la plataforma **Git**, en la qual es podran realitzar **múltiples lliuraments** durant el procés de desenvolupament.
- Un cop l'estudiant consideri que l'exercici està llest per a la seva qualificació, haurà de **notificar al Docent Col·laborador** mitjançant un missatge que inclogui la **ruta del repositori** i el **nom d'usuari de Git**.
- A més, és obligatori **incloure comentaris clars i adequats en el codi**, que facilitin la seva comprensió i manteniment.

Lliurament

El lliurament de l'activitat s'haurà de realitzar mitjançant un fork a GitHub.

💡 Únicament es qualificaran aquells projectes que hagin enviat el correu sol·licitant la revisió per part del professor col·laborador de la teva aula; en cas de no complir amb aquest requisit i un cop finalitzat el curs, **l'estudiant suspendrà de manera automàtica el nivell.**