Progetto del corso di Applicazioni Web - A.A. 2024/25

1. Introduzione

Il progetto consiste nello sviluppo di un'**applicazione web full-stack** che segua i **principi del paradigma REST [3]**, esponendo almeno **quattro endpoint**: uno per ciascuno dei metodi HTTP fondamentali (**GET**, **POST**, **PUT**, **DELETE**).

Rappresenta il **naturale completamento del corso**, offrendo la possibilità di consolidare le competenze acquisite nella progettazione del **front-end**, del **back-end** e nella gestione delle **richieste HTTP**.

Il **dominio applicativo è a scelta libera**, purché siano rispettati i **requisiti tecnici** descritti nelle sezioni seguenti.

2. Contesto e principi

Nel 2000, **Roy Fielding** descrisse nella sua tesi di dottorato [1] l'architettura **REST** (REpresentational State Transfer) come un insieme di linee guida per progettare applicazioni efficienti e scalabili su Internet, in particolare tramite il protocollo **HTTP**.

In questa visione, HTTP è un **linguaggio con regole ben definite**: gli **URI** identificano le risorse, i **metodi HTTP** (GET, POST, PUT, DELETE) specificano l'azione da compiere, gli **header** forniscono informazioni aggiuntive e i **codici di stato** (come 200, 201, 404) indicano il risultato della richiesta.

L'idea centrale è lavorare con **entità**, non con operazioni: invece di invocare funzioni come `getBook()`, si interroga l'indirizzo `/books/42`, lasciando al metodo HTTP il compito di determinare l'azione.

Anche il formato della risposta può variare: ad esempio, un libro può essere restituito in **HTML** per la visualizzazione o in **JSON** per l'elaborazione da parte di uno script. È il **client** a indicare il **formato preferito** tramite l'header `Accept`.

In questo progetto è richiesto di realizzare un'applicazione che rispetti i **principi REST**, modellando **una o più entità** (ad esempio *libri*, *allenamenti*, *…*), ciascuna identificata da un **campo `id`** e descritta da **almeno tre attributi** di tipo stringa, numero o booleano. Le entità devono essere rappresentate in **formato JSON** e archiviate **in memoria** oppure **su file**, ad esempio in un file `data.json`.

Questa è una semplificazione a fini didattici: in un'applicazione reale, i dati verrebbero gestiti tramite un **database**. In questo progetto, invece, le modifiche sono **persistenti solo**

fino al riavvio dell'applicazione, a meno che non venga utilizzata la scrittura su file o su un database "embedded" come **SQLite**.

3. Obiettivo

Il progetto richiede di sviluppare un'applicazione web che rispetti i **principi fondamentali del paradigma REST**, sia nella definizione degli endpoint sia nella gestione delle entità. L'applicazione dovrà essere strutturata in modo coerente, con una **logica di routing** che segua la **semantica dei metodi HTTP** e garantisca una corrispondenza chiara tra le **operazioni richieste** e i **percorsi utilizzati**.

Il server dovrà assicurare una **gestione affidabile delle entità** e una **validazione rigorosa dei dati**: sarà necessario controllare la presenza di tutti i campi obbligatori, la correttezza dei tipi, l'aderenza a pattern specifici e il rispetto dei vincoli stabiliti dal modello. Questo contribuirà a garantire la **coerenza e l'affidabilità** delle operazioni ed eviterà possibili **abusi da parte di utenti malintenzionati**.

L'interfaccia utente, sviluppata con **HTML, CSS e JavaScript**, dovrà essere **interattiva** e **reattiva**, in grado di guidare l'utente nelle operazioni previste e di aggiornare dinamicamente in risposta alle interazioni con il server. Anche lato client è richiesto un primo livello di **validazione dei dati**, utile per intercettare errori comuni prima dell'invio delle richieste.

Si raccomanda di fornire un **design semplice e coerente**, garantendo un buon livello di **user experience**. Anche gli **aspetti di presentazione dell'interfaccia** saranno oggetto di valutazione del progetto.

4. Cosa fare

Tutti i dettagli specifici su come svolgere il progetto vengono riportati in questa sezione.

4.1 Scelta del dominio e modello dei dati

Ogni gruppo è chiamato a **scegliere un dominio applicativo** (ad esempio la gestione di un catalogo di libri o una lista di allenamenti) e a identificare **un'entità principale** attorno alla quale ruoteranno tutte le funzionalità fondamentali dell'applicazione, come ad esempio `Book`, `Workout`, ecc. Questa entità dovrà essere **accessibile, modificabile e cancellabile** tramite i corrispondenti **endpoint REST**.

L'entità principale deve essere descritta da almeno **quattro attributi significativi**, coerenti con il dominio scelto, e dotata di un **identificatore univoco** (`id`) generato automaticamente dal server.

A questa **entità principale** possono eventualmente affiancarsi una o più **entità secondarie**, collegate da **relazioni semplici** (ad esempio `Review` per un `Book`, `Exercise` per un `Workout`). L'introduzione di entità aggiuntive è **facoltativa**, ma può rappresentare un'estensione utile per approfondire la modellazione e le relazioni tra i dati. La

progettazione corretta dell'entità principale rimane comunque il **requisito minimo indispensabile** per il completamento del progetto.

Le **regole di validazione dei dati** dovranno essere **definite con precisione** e **documentate nella relazione**. È fondamentale che il **modello dei dati** sia progettato con cura **prima** di iniziare l'implementazione dell'applicazione.

4.2 Progettazione del back-end

Il **server** deve essere sviluppato con **Express.js**, seguendo le pratiche viste a lezione. Il codice va organizzato in modo **chiaro e modulare**, separando le responsabilità principali (es. **definizione delle rotte**, **gestione dei dati**, **validazione**).

L'implementazione dovrà garantire un comportamento **coerente con i principi REST**, utilizzando in modo corretto i **metodi HTTP** e restituendo **codici di stato** e **risposte JSON** appropriate.

Il server dovrà implementare i seguenti **comportamenti**:

Metodo	**Percorso**	**Comportamento previsto**
`GET`	`/items`	Restituisce l'intera collezione di risorse.
`GET`	`/items/:id`	Restituisce una risorsa specifica o un errore.
`POST`	`/items`	Crea una nuova risorsa.
`PUT`	`/items/:id`	Aggiorna la risorsa o la crea se non esiste.
`DELETE`	`/items/:id`	Elimina la risorsa.

Il percorso **/items** è indicativo e andrà sostituito con il **nome dell'entità principale**, declinato al plurale (es. `/books`, `/workouts`, ecc.).

La gestione delle risorse può avvenire **in memoria**, mantenendo gli oggetti in un array o in un dizionario, oppure **su file**, serializzando i dati in **data.json** da leggere e aggiornare in modo asincrono durante l'esecuzione. Entrambe le soluzioni sono accettabili, e la seconda consente di mostrare la **scrittura persistente su disco**.

In alternativa, è possibile utilizzare un **database embedded** come **SQLite**, a condizione che l'integrazione sia **semplice**, **coerente con la struttura del progetto** e **ben documentata nella relazione**. L'uso consapevole di un database, anche se non obbligatorio, può rappresentare un **valore aggiunto nella valutazione finale**, in particolare per i gruppi interessati a sperimentare funzionalità più avanzate.

In ogni caso, il **server** dovrà garantire un **comportamento prevedibile** e conforme allo **standard REST**, restituendo sempre il **codice di stato appropriato** e, se necessario, una **risposta in formato JSON** o un **messaggio d'errore chiaro e informativo**.

Dovrà inoltre eseguire una **validazione accurata dei dati ricevuti**, verificando la **presenza dei campi obbligatori**, la **correttezza dei tipi**, l'aderenza a **pattern specifici**, eventuali **intervalli numerici** e ogni altra regola definita nel **modello dell'entità**.

4.3 Progettazione del front-end

L'**interfaccia utente** dell'applicazione dovrà essere sviluppata con **HTML**, **CSS** e **JavaScript**, seguendo i principi illustrati durante il corso. L'obiettivo è costruire un **front-end funzionale, chiaro e interattivo**, che permetta all'utente di svolgere tutte le operazioni fondamentali sull'entità principale: **visualizzazione della lista**, **consultazione del dettaglio**, **creazione**, **modifica** e **cancellazione**.

Le interazioni dovr generare una **richiesta HTTP** verso il server Express, utilizzando la **Fetch API**. Le **risposte del server** dovranno essere gestite con attenzione, **aggiornando dinamicamente il DOM** per riflettere lo stato corrente dell'applicazione.

L'interfaccia deve fornire **feedback visivi chiari** in tutte le situazioni: **conferme** per le operazioni completate con successo e **messaggi d'errore comprensibili** in caso di problemi (come dati mancanti, richieste malformate o risorse non trovate). La **validazione dei dati** deve avvenire anche **lato client**, utilizzando le funzionalità native dei **form HTML5** ed eventualmente integrando **controlli personalizzati in JavaScript**.

Il **layout** della pagina deve essere **intuitivo e ben organizzato**, in modo da rendere chiara la struttura dell'applicazione e facilitare l'accesso alle risorse. È apprezzato l'uso di **tecniche di responsive design**, come **Flexbox**, per offrire una buona esperienza anche su **dispositivi con schermi ridotti**.

L'integrazione di **librerie esterne** per il front-end (come **Bootstrap** o **Tailwind** per lo stile, oppure **htmx** o **Alpine.js** per la gestione dell'interfaccia e delle interazioni) è **facoltativa**, ma **valutata positivamente**. L'uso **consapevole** di strumenti non trattati a lezione, se **ben integrati** nel progetto, può **migliorare l'aspetto visivo**, **semplificare il codice** o **ampliare le funzionalità**, rappresentando un **valore aggiunto** nella valutazione finale.

È comunque **pienamente accettabile** un front-end sviluppato **senza dipendenze esterne**, a condizione che assicuri un **comportamento corretto**, una **buona integrazione** con l'API REST e un'interfaccia utente che **guida l'utente** nelle operazioni e **riflette lo stato applicativo**.

4.4 Autenticazione "light" (obbligatoria per i gruppi)

Per i **gruppi di tre persone** è richiesto di integrare una **simulazione di autenticazione**, realizzata utilizzando **solo strumenti già trattati a lezione**. L'obiettivo è mostrare un **meccanismo base di accesso riservato**, in cui l'utente **effettua il login**, riceve un **identificativo (un "token")**, e lo utilizza per eseguire **operazioni protette**. Non è necessario implementare sistemi di sicurezza avanzati: è sufficiente riprodurre in modo chiaro e funzionale le **fasi fondamentali** del flusso di autenticazione. Il comportamento previsto è descritto nella **tabella seguente**:

Specifica	Descrizione
Entità `User`	L'applicazione mantiene una lista di utenti in memoria (o in `data.json`), ciascuno con i campi `id`, `username`, `password` (anche in chiaro, a fini didattici).
Endpoint `POST /login`	L'endpoint riceve le credenziali, verifica la correttezza e, se valide, restituisce un **token fittizio** (ad esempio lo stesso `userId` o una stringa generata al momento).
Header di autenticazione	Le richieste che **creano**, **aggiornano** o **cancellano** dati (POST, PUT, DELETE) includono l'header `X-Auth-Token: <token>`. Ogni handler verifica il token e risponde con un codice di stato appropriato in caso di token mancante o errato.
Permessi sulle risorse	Ogni risorsa dell'entità principale contiene un campo `ownerId`. Il server consente modifiche **solo se il token corrisponde al proprietario**. Le richieste `GET` possono restare pubbliche.
Client e `localStorage`	Il form di login salva il token nel `localStorage` del browser, un **archivio chiave-valore persistente** [2] che mantiene i dati anche dopo l'operazione di aggiornamento della pagina, mediante `localStorage.setItem('authToken', token)`. Alle richieste successive, il token si recupera con `localStorage.getItem('authToken')` e si inserisce nell'header. L'interfaccia indica se l'utente è autenticato e abilita/disabilita i pulsanti di modifica.

> **Nota**: questa soluzione è volutamente elementare e **non adatta alla produzione**; serve solo a capire il meccanismo base tra credenziali, token e controllo dei permessi.

4.5 Ricerca con paginazione (obbligatoria per i gruppi)

Oltre all'autenticazione "light", ai gruppi di tre studenti è richiesto di integrare una funzionalità di **ricerca testuale combinata con paginazione**. Questa estensione serve a dimostrare la capacità di gestire query lato client e lato server in modo flessibile, filtrando e suddividendo i risultati restituiti. Il flusso previsto è il seguente:

Specifica	Descrizione

Endpoint arricchito	L'endpoint `GET /items` (o analogo) **accetta i parametri facoltativi**: • `q` → ricerca testuale (substring su un attributo a scelta, es. *title*) • `limit` → numero massimo di oggetti da restituire (default: **10**) • `offset` → indice di partenza (default: **0**)
Risposta paginata	La risposta **restituisce un oggetto JSON** con almeno due campi: • `results`: array di oggetti filtrati • `total`: conteggio complessivo degli elementi prima dell'applicazione di `limit` e `offset`
Filtro lato server	Il **filtraggio** viene eseguito lato server: si applica prima la ricerca (`q`), poi si limita l'array risultante con `slice(offset, offset + limit)`
Interfaccia di ricerca	L'interfaccia utente comprende **un campo di input** per `q` con relativo pulsante "Cerca", e **due pulsanti** "Precedente" / "Successivo" (oppure link numerati) che modificano i parametri `limit` e `offset` nella fetch
Stato dell'interfaccia	L'interfaccia mostra il valore di `total` e l'intervallo attuale (es. **"11–20 di 83"**). Il pulsante "Precedente" è disabilitato quando `offset = 0`, e "Successivo" quando l'intervallo supera `total`

Nota: la ricerca può essere **case-insensitive** e la paginazione può avvenire **interamente in memoria**. Non è necessario usare database o algoritmi avanzati: bastano poche righe di JavaScript sul server.

4.6 Scrittura della relazione

Il lavoro svolto deve essere documentato in una **relazione in formato PDF**, della lunghezza massima di **sei pagine**. Il documento deve essere redatto in modo **chiaro, ordinato e conciso**, offrendo una panoramica delle **scelte progettuali**, delle **funzionalità implementate** e delle **eventuali estensioni** introdotte. La relazione deve seguire la seguente struttura:

- **Sommario** (max mezza pagina): breve introduzione al progetto con **dominio scelto**, **funzionalità realizzata**, **tecnologie utilizzate**, **obiettivo**, **metodo** e **risultati**.
- **Modello dei dati** (max una pagina): descrizione dell'**entità principale** (e di eventuali entità secondarie), con **attributi**, **tipi** e **regole di validazione**.
- **Implementazione del back-end** (una-due pagine): struttura del server, funzionamento degli **endpoint REST**, gestione di **status code**, **persistenza** e **validazione dei dati**.
- **Implementazione del front-end** (una-due pagine): viste sviluppate, organizzazione dei file, uso della **Fetch API**, aggiornamento dinamico del DOM, eventuali **librerie esterne** adottate.

- **Conclusioni** (max mezza pagina): osservazioni finali, **difficoltà incontrate**, **punti di forza**, proposte di miglioramento, **funzionalità extra** (filtri, ricerca, test…).

Le funzionalità di **autenticazione** e **ricerca con paginazione** sono **obbligatorie per i gruppi**. Gli studenti che lavorano individualmente **possono ometterle senza penalizzazione**, ma devono comunque documentare le funzionalità implementate con la stessa cura.

La relazione deve includere in copertina **nomi**, **cognomi** e **numeri di matricola** di tutti i partecipanti, e va inserita nella cartella `docs/` del repository GitHub al momento della consegna.

5. Come cominciare

Per iniziare, ogni gruppo (o studente singolo) deve **richiedere l'approvazione del progetto via email**, prima di avviare lo sviluppo.

La mail va inviata a michael.soprano@uniud.it, con le seguenti caratteristiche:

- **Oggetto:** `[Progetto WebApp - 2024/2025 - Richiesta] cognome1[_cognome2_cognome3]`
- **Corpo del messaggio:** richiesta sintetica di approvazione del progetto
- Tutti i membri del gruppo devono essere **in copia** alla mail

L'approvazione sarà comunicata via email entro pochi giorni. La **data di risposta da parte del docente** costituisce il riferimento per il calcolo dei **45 giorni disponibili per la consegna**. Il progetto può essere avviato solo dopo l'approvazione ufficiale.

6. Come consegnare

Gli studenti devono organizzarsi in **gruppi di tre persone**. È ammesso anche il **lavoro individuale**, ma **non sono accettati gruppi con un numero diverso di componenti**.

La consegna deve avvenire **entro 45 giorni** dalla **data di accettazione del progetto** da parte del docente, comunicata via email in risposta alla proposta inviata dal gruppo.

Ciascun gruppo (o studente singolo) è tenuto a consegnare i seguenti materiali:

- una **relazione in formato PDF** (massimo **6 pagine**), redatta in modo chiaro e sintetico, contenente **nomi, cognomi e numeri di matricola** di tutti i partecipanti
- il **codice sorgente dell'applicazione**, organizzato secondo la seguente struttura:
  - `backend/` con il codice del server **Express.js**
  - `frontend/` con i file **HTML**, **CSS** e **JavaScript** dell'interfaccia
  - eventuale `data.json` per la simulazione della **persistenza dei dati**
  - `docs/` con la relazione e **materiali aggiuntivi** (screenshot, diagrammi, ecc.)
- un file `README.md` contenente:
  - **istruzioni per l'installazione e l'esecuzione**

    - **descrizione delle route principali**
    - **esempi di richieste e risposte**
    - (facoltativo) una **collezione Postman** per il test delle API

Tutto il materiale va caricato in una **repository pubblica su GitHub**. Chi non possiede un account può registrarsi gratuitamente su github.com. Dopo l'accesso:

1. Creare una nuova **repository pubblica**, assegnando un nome significativo (es. `progetto-webapp-2024-nomecognome`)
2. Caricare tutti i file richiesti, rispettando la struttura delle cartelle indicata
   verificare che la repository sia **visibile pubblicamente**, ad esempio aprendo il link in una finestra in incognito

Per finalizzare la consegna, ogni gruppo (o studente singolo) deve inviare una mail a michael.soprano@uniud.it con le seguenti caratteristiche:

- **Oggetto**: `[Progetto WebApp - 2024/2025 - Consegna] cognome1[_cognome2_cognome3]`
- **Corpo**: il link alla repository GitHub contenente tutto il materiale
- La mail deve essere **messa in copia a tutti i membri del gruppo**

È **responsabilità del gruppo** verificare con attenzione che il **link fornito sia corretto e funzionante**, che la **repository sia accessibile** e che la **consegna sia completa e inviata entro i termini stabiliti**.

Consegne **incomplete**, **inaccessibili** o **oltre la scadenza** non saranno prese in considerazione.

7. Discussione

Dopo la consegna, ogni **gruppo** (o **studente singolo**) sarà convocato per una **breve discussione individuale** sul progetto. L'incontro si terrà **entro 10–15 giorni dalla scadenza**, e la data esatta sarà comunicata **via email** una volta ricevuto tutto il materiale.

Durante la discussione sarà richiesto di **illustrare sinteticamente le funzionalità principali** sviluppate e di **rispondere a domande** relative alle **scelte progettuali**, alla **logica del codice** e al **funzionamento generale dell'applicazione**. L'obiettivo è accertare la **comprensione del lavoro svolto** e il **contributo personale** di ciascun partecipante.

La discussione rappresenta una **parte integrante della valutazione finale**. È quindi essenziale che **tutti i membri del gruppo partecipino attivamente** e siano in grado di **spiegare con chiarezza** il proprio ruolo e le decisioni prese durante lo sviluppo.

Bibliografia

- [1] Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures* (Doctoral dissertation, University of California, Irvine). https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
- [2] Mozilla Developer Network (MDN). *Window.localStorage - Web APIs | MDN*. https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage
- [3] W3Schools. *RESTful Web Services - Introduction*. https://www.w3schools.in/restful-web-services/intro/