Università degli Studi di Camerino

Scuola di Scienze e Tecnologie

Corso di Laurea in Informatica (Classe L-31)

Editor di oggetti 3D per e-commerce configurabile e in realtà aumentata: il backend di "Customyzx"

Laureando
Paolo Andreassi

Matricola 090629

Relatore
Dott. Francesco De Angelis

Correlatore
Dott. Antonio dell'Ava

A.A. 2017/2018

Indice

Elenco dei codici

Elenco delle figure

Elenco delle tabelle

1. Introduzione

La qui presente tesi rappresenta la terza ed ultima parte dello stage++[1], che è stato realizzato in collaborazione con il Prof. Francesco De Angelis, e-xtrategy s.r.l. e con il collega Filippo Corona. Durante lo stage ci è stato chiesto di progettare e realizzare un'applicazione per la visualizzazione e l'editing di modelli 3D da impiegare in ambito e-commerce. Il prodotto che ne è risultato ha preso il nome di "CustoMyzx".

Per poter raggiungere i numerosi obettivi che questo progetto richiedeva, io e Filippo ci siamo divisi il carico di lavoro in maniera il più possibile equa: una metà del progetto si è focalizzata sul front end, e quindi sull'interfaccia mirata all'utente, l'altra invece sul back end, estraneo a quest'ultimo. Nella prima fase della progettazione, i due semi-progetti sono stati nominati rispettivamente "unicam-product-viewer" e "unicam-product-editor" ai fini di una distinzione tra questi. Io mi sono occupato della parte back end, e quindi di "unicam-product-editor", mentre Filippo, in modo complementare, ha sviluppato la parte front end e il relativo sotto-progetto:"unicam-product-viewer".

Quindi il progetto è composto dalle seguenti parti:

1. Il back end
2. Una web app come interfaccia per il back end
3. Il front end
4. Il configuratore 3D e in realtà aumentata sviluppato tramite web app

Tutte le parti di CustoMyzx sono state scritte in Javascript ad eccezione di uno script esterno realizzato in Python.

Questa tesi tratterà nella fattispecie i primi due punti: la realizzazione dell'applicazione web mediante l'utilizzo del MEAN Stack piuttosto che il tradizionale LAMP Stack[2], e l'uso del modello architetturale nelle web app chiamato "Long Polling".

Il codice completo del progetto è disponibile nei due repository

1. www.github.com/e-xtrategy/unicam-product-editor
2. www.github.com/e-xtrategy/unicam-product-viewer

1.1 Progettazione

La fase di progettazione si è sviluppata secondo il metodo extrategy: l'azienda basa lo sviluppo di software sulle metodologie agili, più adatte ad ambienti dinamici come lo

[1] Lo stage++ è un progetto unico che comprende l'insieme di tre moduli: stage, project work e tesi.

[2] Acronimo (Linux, Apache, MySQL, PHP) che indica una piattaforma software per lo sviluppo di applicazioni web

è il web, come ad esempio la formazione di team di sviluppo piccoli, cross-funzionali e auto-organizzati, lo sviluppo iterativo e incrementale, la pianificazione adattiva, e il coinvolgimento diretto e continuo del cliente nel processo di sviluppo.

In particolare, abbiamo utilizzato una via di mezzo tra Scrum e Kanban.

L'utilizzo del framework Scrum[**scrum**] consiste nel passaggio attraverso una serie di iterazioni a intervalli di tempo regolari, dette sprint, che consentono di consegnare il progetto passo dopo passo al cliente e con cadenza costante. Attraverso la giusta frequenza adottata per gli sprint, si è capaci di fare delle stime più corrette, e inoltre si possono ricevere feedback in tempo reale sul lavoro svolto. Ogni sprint si divide in quattro fasi:

1. **Pianificazione**: incontro iniziale in cui si definisce cosa fare con lo sprint corrente.

2. **Scrum giornaliero**: mini incontro giornaliero della durata di una decina di minuti per sincronizzare il team.

3. **Demo**: incontro per illustrare il lavoro svolto dal team.

4. **Retrospettiva**: una revisione di ciò che è stato fatto, esponendo anche i possibili miglioramenti attuabili.

Il Kanban[**kanban**] serve per tenere d'occhio l'avanzamento dei lavori durante gli sprint; la kanban board dunque non è altro che una lavagna atta a visualizzare e ottimizzare il flusso di lavoro del team. Un classico kanban può essere dato da una lavagna o da un' intera parete (es. e-xtrategy), ma per il nostro sviluppo software abbiamo utilizzato una board virtuale, in cui abbiamo riscontrato aspetti positivi quali tracciabilità, facilità di collaborazione, accessibilità delocalizzata.

Le funzioni di una kanban sono comunque quelle di standardizzare il flusso di lavoro e di assicurare una buona visibilità e visualizzazione del lavoro del team, per poter individuare e risolvere tempestivamente eventuali criticità. Questa metodologia inoltre è basata sulla trasparenza e sulla comunicazione, pertanto la kanban board dovrebbe essere vista come una visione oggettiva e veritiera agli occhi del team. Di norma si prevedono tre step: To Do, In Progress e Done, che noi abbiamo rinominato rispettivamente in Backlog, Sprint e Approved.

Con questa impostazione, Antonio Dell'Ava, il nostro tutor in e-xtrategy, ci ha guidato verso lo strumento più utile per tenere traccia dello sviluppo del software, implementare le user-stories e coordinare gli sviluppatori assegnati al progetto, in questo caso io e Filippo: Trello ci ha permesso di gestire il nostro piccolo team senza applicare alla lettera Kanban, ma tramite una semplice todo list.

Trello[**trello**] è un gestore di progetti basato sul web e realizzato da Fog Creek Software nel 2011. Qui il progetto è rappresentato da schede, a loro volta contenenti liste corrispondenti ad elenchi di attività. Le liste sono formate da card, che rappresentano le singole attività. Queste card si possono spostare tra le diverse liste con il metodo drag and drop, per seguire il normale flusso di sviluppo passando dalla lista "da fare" ad esempio alla lista "fatto" una volta che è stata realizzata questa parte del progetto.

Nel nostro caso ogni card rappresenta una user-story, ossia una funzionalità utile al raggiungimento di un obiettivo di business, una descrizione volutamente generica di una caratteristica che il software deve avere, e che può essere stata esplicitamente richiesta dal cliente, essere necessità o consiglio dello sviluppatore, o essere nata dalla discussione

tra i due soggetti. Ogni card può vedersi assegnata ad uno o più sviluppatori, e queste si possono organizzare, insieme alle loro schede, in raggruppamenti personalizzati. Si possono inserire inoltre commenti, allegati, date di scadenza, voti e liste di controllo in ogni card. Nel complesso, la suddivisione di un progetto in schede/liste, che a loro volta sono suddivise in card, formano un insieme di dati su misura organizzati in modo gerarchico, che rende più facile ed efficace la gestione dei progetti, ma di conseguenza anche le attività di tutta l'organizzazione.

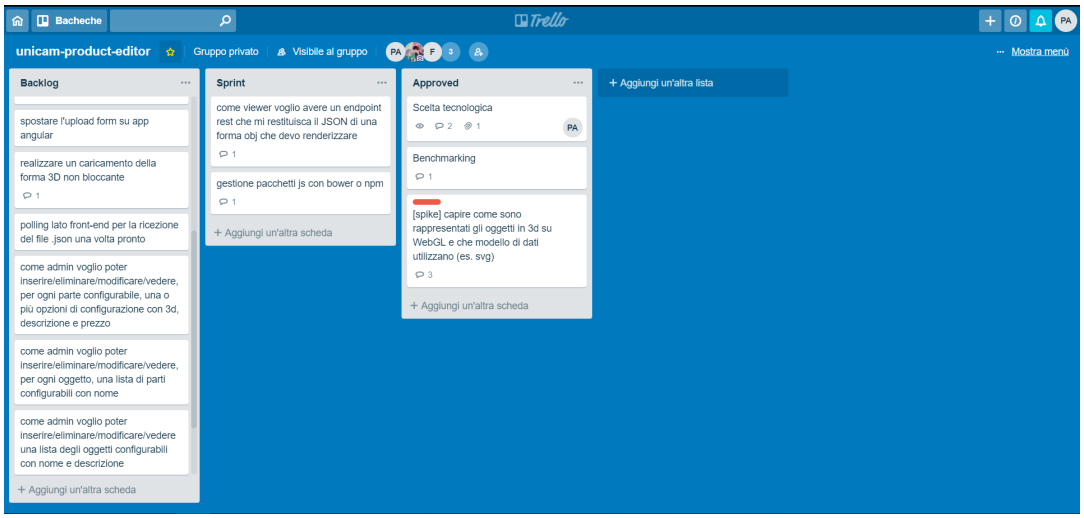

Figura 1.1: Organizzazione di unicam-product-editor su Trello

1.2 Analisi

Si vuole realizzare una applicazione web che permetta di gestire gli oggetti 3D tramite upload, modifica e visualizzazione in formato .JSON. Il backend deve ricevere in input una forma 3D sotto forma di file in formato .obj, convertirlo in .JSON e fornire i servizi REST. Un utente deve poter inserire l'oggetto 3D tramite l'uploader, modificare i metadati dell'oggetto convertito e visualizzare il file convertito attraverso un apposito endpoint una volta effettuata l'autenticazione. Il processo di upload e conversione in particolare non deve essere bloccante, in quanto l'utente deve poter continuare ad utilizzare il servizio anche durante l'upload di una forma 3D molto grande.

La struttura dei documenti all'interno del progetto sarà la seguente:

- Lista
  - Oggetto
  - Omino Lego
    - Parte
    - Testa
    - Busto
      - Opzione
        - Busto standard
        - Busto con papillon
        - Busto con cravatta

Le user stories derivanti da questo processo di analisi verranno identificate dalla sigla "US-XX", in cui XX rappresenta un numero progressivo, e vengono elencate di seguito:

Numero	**Descrizione**
US-01	Scelta tecnologica.
US-02	Benchmarking.
US-03	Capire come sono rappresentati gli oggetti in 3d su WebGL e che modello di dati utilizzano (es. svg).
US-04	Come admin voglio poter inserire/eliminare/modificare/vedere una lista degli oggetti configurabili con nome e descrizione.
US-05	Come admin voglio poter inserire/eliminare/modificare/vedere, per ogni oggetto, una lista di parti configurabili con nome.
US-06	Come admin voglio poter inserire/eliminare/modificare/vedere, per ogni parte configurabile, una o più opzioni di configurazione con 3D, descrizione e prezzo.
US-07	Come viewer voglio avere un endpoint rest che mi restituisca il JSON di una forma obj che devo renderizzare.
US-08	Creare app single-page con Angular.
US-09	Realizzare un caricamento della forma 3D non bloccante (upload, conversione ed endpoint in modo sincrono).
US-10	Polling lato front-end per la ricezione del file .JSON una volta pronto.

Tabella 1.1: User stories

Il diagramma di Gantt riguardante l'assegnazione dei task e lo svolgimento temporale, è il seguente:

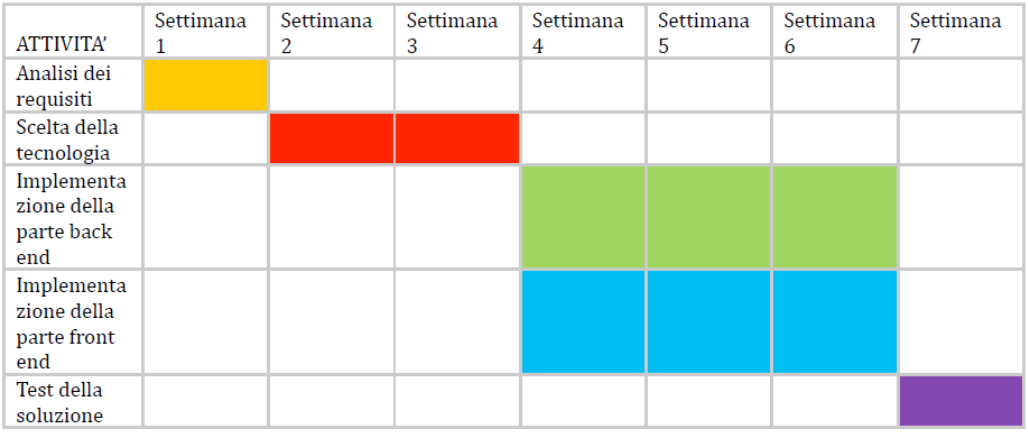

Figura 1.2: Piano di realizzazione del progetto

2. E-commerce

In questo capitolo si illustrano brevemente le tecnologie di visualizzazione applicate nell' e-commerce.

2.1 Visualizzazione dei prodotti nell'e-commerce

Foto, grafica 360 (comunque foto), configuratori basati su fotografie

2.2 3D e realtà aumentata

Il 3D e la realtà aumentata (nell'ecommerce)

3. Scelta Tecnologica

In questo capitolo viene trattata l'architettura utilizzata per unicam-product-editor e il modo in cui vengono gestiti i dati attraverso la web app, per poi passare ad illustrare l'organizzazione del codice.

Per sviluppare unicam-product-editor è stato utilizzato lo stack MEAN che non è altro che l'insieme dei quattro componenti che ne fanno parte:

- M = MongoDB: il popolare database
- E = Express.js: un framework web leggero
- A = Angular.js: un framework per la creazione di single page application
- N = Node.js v4: un interprete JavaScript costruito sul motore JavaScript V8 di Google Chrome orientato agli eventi che lo rende efficiente e leggero

Lo stack MEAN è un' alternativa al più tradizionale stack LAMP (Linux, Apache, MySQL, PHP/Perl/Python/P...), molto popolare per il suo utilizzo nella costruzione di applicazioni web dagli anni '90 in poi. Di seguito andiamo a costruire le RESTful API, che sono la base per qualsiasi tipo di applicazione, come un sito web, un'app Android, iOS o Ubuntu Touch.

3.1 Oggetti 3D

Un modello 3D è una rappresentazione matematica di un oggetto tridimensionale e consiste sostanzialmente in un file di dati strutturati, contenenti le proprietà delle primitive geometriche che costituiscono l'oggetto rappresentato (un corpo fisico). Un oggetto 3D quindi non è altro che un insieme di dati composto da punti nello spazio tridimensionale (tre dimensioni X, Y e Z) e altre informazioni (spesso metadati, o informazioni aggiuntive). I principali formati utilizzati per i modelli 3D:

- OBJ
- FBX
- Collada
- BLEND
- DFX
- STL
- PLY

- 3DS (obsoleto)
- VRML (obsoleto)
- X3D

La scelta per cui abbiamo optato per la realizzazione del nostro progetto è ricaduta sul .obj, o più correttamente Wavefront .obj, un formato di file sviluppato dalla Wavefront Technologies per il software Advanced Visualizer. Il motivo di questa scelta risiede nel fatto che il .obj è un formato aperto che è stato adottato da tantissimi applicativi per la grafica 3D per l'interscambio di dati con altri programmi; ciò lo rende perfetto per il nostro progetto, in quanto questo dovrà poi essere trasformato in un corrispondente file .JSON per poter essere interpretato dal viewer.

Questo è, per intero, il file .obj per la creazione di un cubo

```
# cube.obj
#

g cube

v 0.0 0.0 0.0
v 0.0 0.0 1.0
v 0.0 1.0 0.0
v 0.0 1.0 1.0
v 1.0 0.0 0.0
v 1.0 0.0 1.0
v 1.0 1.0 0.0
v 1.0 1.0 1.0

vn 0.0 0.0 1.0
vn 0.0 0.0 -1.0
vn 0.0 1.0 0.0
vn 0.0 -1.0 0.0
vn 1.0 0.0 0.0
vn -1.0 0.0 0.0

f 1//2 7//2 5//2
f 1//2 3//2 7//2
f 1//6 4//6 3//6
f 1//6 2//6 4//6
f 3//3 8//3 7//3
f 3//3 4//3 8//3
f 5//5 7//5 8//5
f 5//5 8//5 6//5
f 1//4 5//4 6//4
f 1//4 6//4 2//4
f 2//1 6//1 8//1
f 2//1 8//1 4//1
```

Codice 3.1: Cube.obj

in cui sono specificate le posizioni e le grandezze nello spazio tridimensionale dei vertici (v), dei vertici normali (vn), e delle facce (f).

L'oggetto 3D che ne deriva può essere facilmente visualizzato tramite ad esempio il programma Paint 3D:

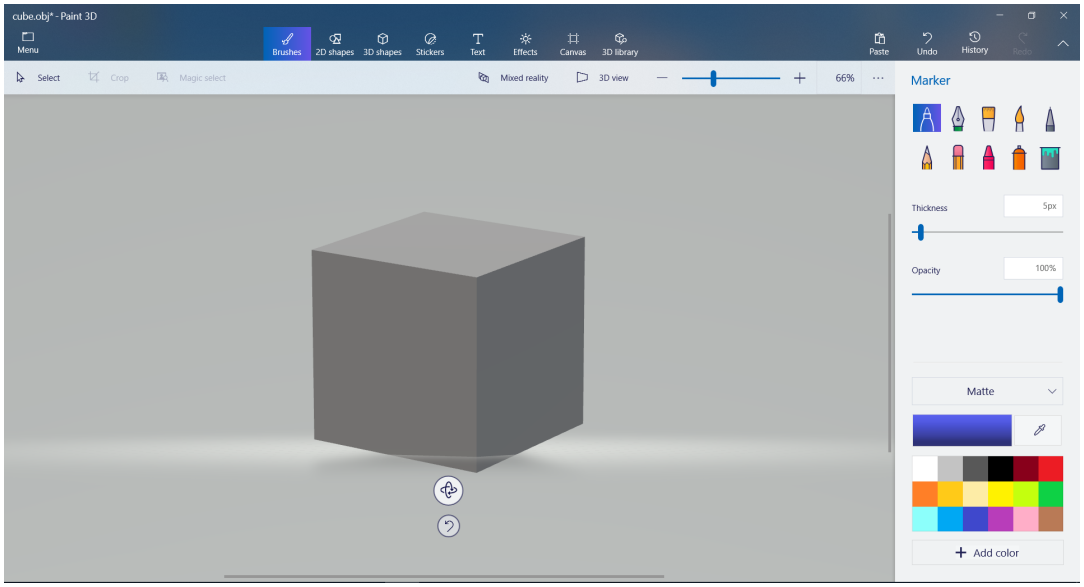

Figura 3.1: Visualizzazione grafica di cube.obj

3.2 JSON

JSON è un formato di dati molto leggere basato su un sottoinsieme della sintassi JavaScript, cioè *array literal* e *object literal*, proposto da Douglas Crockford come formato per i servizi web in sostituzione al verboso XML.

Dato l'utilizzo della sintassi Javascript, le definizioni JSON possono essere incluse all'interno dei file JavaScript.

3.2.1 Gli array literal

Gli *array literal* rappresentano il metodo più semplice per creare un array in JavaScript. Basta infatti elencare una insieme di valori separati da una virgola all'interno delle parentesi quadre, per esempio:

```
var users = ["Paolo", "Antonio", "Filippo"];
```

A livello di programmazione è equivalente al codice:

```
var users = new Array("Paolo", "Antonio", "Filippo");
```

Entrambe le definizioni generano lo stesso risultato ed è possibile accedere ad ogni elemento dell'array utilizzando il relativo indice

```
console.log(users[0]); \\prints Paolo
console.log(users[1]); \\prints Antonio
console.log(users[2]); \\prints Filippo
```

Gli array in JavaScript hanno però due caratteristiche ben precise:

1. In JavaScript l'array non ha un tipo di dati ben definito (non è tipizzato), e quindi ogni elemento può contenere tipi di dati diversi.

2. Nonostante entrambi i metodi sopra citati per la creazione di un array siano validi in JavaScript, in .JSON soltanto gli array literal verranno accettati e interpretati correttamente.

3.2.2 Gli object literal

Un object literal è definito tra parentesi graffe. Al suo interno troviamo un numero qualsiasi di coppie chiave-valore, definite con una stringa, i due punti ed il valore. Ogni coppia deve essere seguita da una virgola, tranne l'ultima. Questo insieme di coppie chiave-valore rappresenta, nel suo insieme, un oggetto. Un esempio:

```
var user = {
"firstname": "Paolo",
"age": 23,
"student": true
};
```

Il codice qui sopra genera un oggetto *user* con le proprietà firstname, age e student. Per accedere alle proprietà dell'oggetto basta usare la notazione con il punto:

```
console.log(user.firstname); \\prints "Paolo"
console.log(user.age); \\prints "23"
console.log(user.student); \\prints "true"
```

Lo stesso oggetto potrebbe essere creato utilizzando il costruttore `Object` di JavaScript

```
var user = new object():
user.firstname = "Paolo";
user.age = 23;
user.student = true;
```

Come già detto, sebbene entrambe le definizioni siano accettate in JavaScript, in JSON è valida solo la seconda notazione, object literal.

3.2.3 La sintassi JSON

La sintassi di JSON non è altro che il miscuglio di object literal ed array literal per memorizzare dati e solo e soltanto rappresentarli. Un esempio:

```
[
{
"firstname": "Paolo",
"age": 23,
"student": true
},
{
"firstname": "Giuseppe",
"age": 40,
"student": false
}
]
```

La prima cosa da notare è che in questo documento non sono presenti variabili, così come punti e virgola. In questo modo quando si trasmettono dei dati tramite HTTP ad un browser, il tutto avviene abbastanza velocemente grazie al numero ridotto di caratteri.

Oltre a questo, ci sono altri ovvi benefici nell'utilizzo di JSON come formato dati per la comunicazione in JavaScript: non bisogna preoccuparsi della valutazione dei dati, e quindi, è garantito un accesso più veloce alle informazioni che questo contiene.

3.3 RESTful API

Il significato dell'acronimo REST è REpresentational State Transfer, ossia una rappresentazione del trasferimento di stato di un determinato dato.

REST utilizza un modello client-server, dove il server è un server HTTP e il client invia richieste HTTP (GET, POST, PUT, DELETE), con un URL che al suo interno contiene i parametri codificati. L'URL descrive l'oggetto e il server genericamente risponde restituendo un'immagine, un documento HTML, un file CSV o qualunque altro tipo di dato. Nel nostro caso il server risponde restituendo del codice e un JSON (JavaScript Object Notation). Una RESTful API quindi è l'Interfaccia di Programmazione di un'Applicazione che usa le richieste HTTP per gestire i dati che vengono scambiati fra client e server.

Dato che in unicam-product-editor viene utilizzato lo stack MEAN, abbiamo scelto di utilizzare documenti JSON che risultano particolarmente adatti per la nostra applicazione, visto che tutti i nostri componenti sono in JavaScript e MongoDB interagisce perfettamente con questo formato. Faremo degli esempi più dettagliati più avanti quando definiremo i nostri Data Model.

Le operazioni principali che si possono eseguire con questo metodo sono le seguenti:

- `GET` per richiedere al server un determinato set di dati;
- `POST` per creare un nuovo documento all'interno del database;
- `PUT` per modificare o sostituire completamente un documento già esistente;
- `DELETE` per cancellare un documento contenuto all'interno del database al quale siamo collegati.

Queste operazioni vengono spesso descritte attraverso l'acronimo CRUD, che sta per CREATE, READ, UPDATE e DELETE. Il server HTTP, da parte sua, usa spesso assieme alle API REST dei codici di risposta; di seguito vengono elencati alcuni fra i più comuni:

- 200 - "OK"
- 201 - "Created" (Utilizzato con POST)
- 400 - "Bad Request" (Ad esempio per l'assenza di parametri)
- 401 - "Unauthorized" (Non ci sono i parametri per l'autenticazione)
- 403 - "Forbidden" (L'utente è autenticato ma non ha i permessi)
- 404 - "Not Found"

Una descrizione completa è contenuta nell'apposito RFC 2616[**RFC2616**].

Lo sviluppo di API REST è alla base del nostro sviluppo, in quanto permette alla nostra applicazione di essere multipiattaforma. Per fare alcuni esempi di applicazioni di questo tipo si possono citare Google Docs, Pixlr o lo stesso Trello insieme ad Evernote. Le API REST permettono la facile implementazione dell'applicazione su molte piattaforme che potranno essere sviluppate in un secondo momento, trasformando il progetto iniziale in un progetto indipendente dalla piattaforma di partenza.

I dati da gestire nel nostro progetto si possono riassumere in questo modo:

- Dati degli utenti
- Liste
- Oggetti
- Opzioni di ogni oggetto
- Parti di ogni opzione

Come potremo vedere a breve, diversamente dai tradizionali Database relazionali (in cui la forma principale di struttura dati è data dalle Tabelle), in MongoDB, essendo un Database NoSQL, e quindi non relazionale, non si parla di Tabelle, anche se il concetto di Database ad alto livello è lo stesso.

Nel Database MongoDB possono essere contenute più Collection, che possono forse essere paragonate alle Tabelle nei Database relazionali. A sua volta, ogni Collection conterrà uno o più Document, che a confronto con un Database relazionale corrisponde all'incirca alla riga di una Tabella. Ogni Document (così come ogni Collection) non segue però uno schema specifico come in una Tabella, ma può essere formato da una o più coppie chiave-valore, dove il contenuto può essere a sua volta una variabile semplice oppure qualcosa di più complicato come un array.

Il documento JSON appena preso in esempio rappresenta un utente nel nostro sistema. Ogni campo ha il suo specifico scopo, ma il campo più importante in un Document MongoDB è senza dubbio il campo `_id`: questo rappresenta la chiave primaria di ogni Document. Quando un Document viene salvato senza questo campo, MongoDB ne assegna uno automaticamente, e con un valore univoco.

Ora passiamo ai vari Document che compongono le nostre Collection; la Collection degli Utenti conterrà Documenti di questo tipo:

```
{
      "_id": {
            "$oid": "5894fa05f882a91c7c5cf0e6"
      },
      "displayName": "Paolo Andreassi",
      "email": "pablo15941@gmail.com",
      "password": "$2a$10$kyaJc1WP9KFvSVWzrPagLe.vgY76FjBQjwt76.HhC5u9oo96qoRo.",
      "__v": 0,
      "picture": "https://graph.facebook.com/v2.3/10209758034881202/...",
      "google": "106815394034042298036"
}
```

Codice 3.2: User Collection

Si può notare l'importante presenza del campo `_id`, così come gli altri dati utili al riconoscimento dell'utente. La password inoltre in unicam-product-editor non viene mai mostrata in chiaro all'interno del Database, anche se il campo ad essa corrispondente è presente e popolato. Questo è il risultato di una password impostata dall'utente ma successivamente encriptata con una funzione hash.

Il Document di tipo Lista ha uno scopo puramente descrittivo nei confronti di ciò che contiene:

```
{
"_id": {
"$oid": "589a30d1b2a4c717a8da16b3"
},
"nome": "Omino LEGO",
"__v": 0
}
```

Codice 3.3: List Collection

La Collection degli Oggetti avrà dei Document in cui i campi `nome` e `descrizione` hanno una funzione descrittiva, ma oltre al campo `_id` è presente il campo `_list` che funge da collegamento con la Collection delle Liste: in questo campo infatti è indicato l'identificativo della Lista a cui appartiene uno specifico Oggetto:

```
{
"_id": {
"$oid": "589a38f93ec83413e4f010b2"
},
"nome": "Omino LEGO",
"descrizione": "LEGO figure",
"_list": [
"589a30d1b2a4c717a8da16b3"
],
"__v": 0
}
```

Codice 3.4: Object Collection

Essendo un Database con una struttura nidificata, anche i Document nelle Collection delle Parti e delle Opzioni avranno una struttura simile a quelli delle Collection degli Oggetti: anche in queste infatti è indicato il campo del Documento *parente* a cui fanno riferimento:

```
{
"_id": {
"$oid": "589a3a2857ba982460b83fb7"
},
"nome": "Testa",
"_object": [
"589a38f93ec83413e4f010b2"
],
"__v": 0
}
```

Codice 3.5: Part Collection

La differenza più importante risiede nei Document che compongono la Collection delle Opzioni: in questi vengono indicati anche il campo `prezzo`, ma soprattutto il campo `forma`, che contiene una parte di URL con cui effettuare una chiamata REST API. Il campo forma, che viene mostrato all'utente nell'interfaccia della applicazione web, permette di visualizzare il file JSON relativo ad una forma 3D inserita tramite l'uploader e successivamente convertita nel formato usato da MongoDB e JavaScript:

```
{
"_id": {
"$oid": "589a3c5938319d1c8cb4e787"
},
"nome": "Testa Yoda",
"prezzo": 50,
"forma": "/shape/58593a6319290c09984a439b",
"_part": [
```

```
9   "589a3a2857ba982460b83fb7"
10  ],
11  "__v": 0
12  }
```

Codice 3.6: Option Collection

Infine la Collection delle forme 3D convertite contiene l'insieme dei file JSON che rappresentano le stesse forme.

3.4 Documentazione

Una parte fondamentale della progettazione di API REST è la documentazione. Per scrivere la documentazione necessaria è stato usato lo strumento Postman.

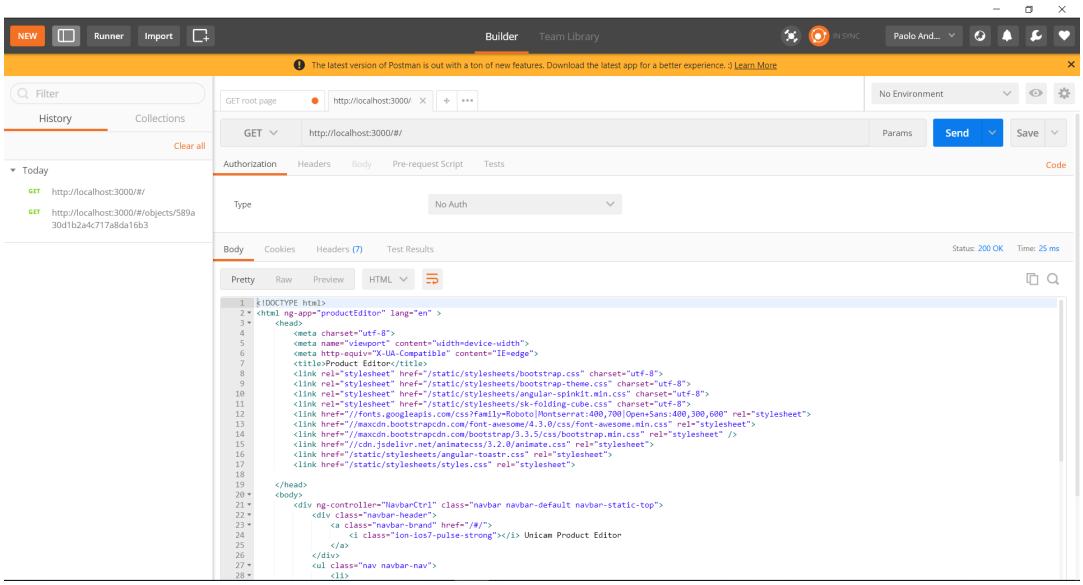

Figura 3.2: L'interfaccia utente di Postman

3.5 NodeJS

Node.js[**node**] v4 è un interprete JavaScript orientato agli eventi, progettato per realizzare applicazioni di rete scalabili. La caratteristica principale è che rimane inattivo finché non arriva una connessione in ingresso, o più generalmente un qualsiasi evento, e viene chiamata la relativa callback. Questo paradigma è in contrasto con quello tradizionale basato sui thread. Il principale vantaggio del paradigma ad eventi rispetto a quello basato sui thread si ha quando c'è un alto traffico. In questo caso, infatti, l'alto numero di thread genera un alto overhead diminuendo drasticamente le prestazioni della macchina host. Inoltre, dato che nessuna funzione di Node eseguirà direttamente operazioni di I/O, il processo non si bloccherà mai.

La versione di Node usata in unicam-product-editor è la v4.6.0. Se da un lato su un server è normale avere solo una versione di Node, potrebbe non esserlo su una macchina di sviluppo. Può capitare, infatti, che uno sviluppatore sia impegnato su più progetti Node che utilizzano versioni differenti. Per risolvere questo problema

possiamo installare NVM. Node Version Manager permette di avere più versioni di Node contemporaneamente nel nostro sistema e fornisce una semplice CLI per poterle gestire. In particolare, basta creare un file di testo chiamato `.nvmrc` in cui salvare il numero di versione di Node che si intende usare per la cartella corrente. Poi, sarà sufficiente digitare il comando `nvm install` per scaricare ed utilizzare la giusta versione.

Node è disponibile per diverse piattaforme, come Linux, Microsoft Windows e Apple OS X. Le applicazioni Node.js vengono costruite utilizzando librerie e moduli disponibili per l'ecosistema ed alcuni di essi sono stati utilizzati in unicam-product-editor. Per iniziare ad utilizzare Node.js, dobbiamo per prima cosa creare un file `package.json`, che descrive la nostra applicazione ed elenca tutte le sue dipendenze. NPM (Node.js Package Manager) installa una copia delle librerie in una sotto-cartella chiamata `node_modules/`, della cartella principale dell'applicazione. Questo comporta diversi benefici, ad esempio isola le diverse versioni delle librerie, evitando così i problemi di compatibilità che si sarebbero presentati se avessimo installato tutto in una cartella standard come `/usr/lib`.

Il comando `npm install` creerà la cartella `node_modules/`, con dentro tutte le librerie richieste e specificate nel file package.json:

```
{
      "name": "unicam-product-editor",
      "version": "1.0.0",
      "private": "true",
      "description": "Editor di prodotti 3D",
      "repository": "https://github.com/e-xtrategy/unicam-product-editor",
      "main": "app.js",
      "scripts": {
            "test": "nodemon --ignore tmp/ app.js",
            "start": "node app.js"
      },
      "author": "Paolo Andreassi",
      "license": "ISC",
      "dependencies": {
      ...
      "async": "^2.1.4",
      "bcryptjs": "^2.4.0",
      "body-parser": "^1.15.2",
      ...
      "cors": "^2.8.1",
      "express": "^4.14.0",
      "express-fileupload": "0.0.5",
      "express-handlebars": "^3.0.0",
      ...
      "mongodb": "^2.2.10",
      "mongoose": "^4.7.8",
      ...
      "python-shell": "^0.4.0",
      ...
      "satellizer": "^0.15.5",
      ...
      },
      "devDependencies": {
            "nodemon": "^1.11.0",
            "should": "~7.1.0",
            "supertest": "^1.1.0"
      },
            "engines": {
            "node": "4.6.0"
      }
}
```

Codice 3.7: package.json

E' interessante notare alcune parti di questo file, oltre al fatto che anche questo è in formato JSON, e oltre alle prime righe che descrivono il progetto in sé per sé. Ci sono due script: quello chiamato *start* avvia l'applicazione normalmente tramite il comando `node` seguito dal nome del file principale del server `app.js`. Lo script chiamato *test* invece è interessante, perché avvia uno strumento (installato tramite l'apposita dipendenza) chiamato nodemon. Questo utilissimo *demone* fa sì che in fase di test ogni qual volta si verifichi un errore che interrompe l'esecuzione del server nodemon attende un salvataggio dei file del progetto, che indica che lo sviluppatore ha modificato il codice per correggere l'errore, e non appena questo evento si verifica il demone riavvia in automatico il server ricominciando ad eseguire il codice.

Un'altra dipendenza interessante riguarda *python-shell*, che nel nostro progetto servirà ad eseguire lo script della libreria Three.js scritto in codice Python (altrimenti non eseguibile da Javascript), che si occupa di convertire il file .obj in JSON. python-shell può ricevere in input interi file scritti in Python per eseguirli in modo asincrono.

Focalizziamoci ora sulla dipendenza *async* per poter entrare nel dettaglio in una peculiarità di Node.js. Qualsiasi operazione che ha a che fare con blocchi di I/O, come leggere un file o interrogare un database, prenderà una callback come ultimo parametro e continuerà l'esecuzione del programma, ma solo quando l'operazione sarà completa il programma richiamerà la callback. Questo accade perché Node.js è progettato per eseguire le istruzioni in modo asincrono.

```
1 function foo() {
2       someAsyncFunction(params, function(err, results) {
3             console.log("one");
4       });
5       console.log("two");
6 }
```

Codice 3.8: operazioni asincrone

Dal codice 3.8 ci si potrebbe aspettare che il risultato sia: `onetwo`.
In realtà potrebbe capitare che l'output sia `twoone`.

Questa caratteristica dei programmi asincroni è detta esecuzione non deterministica. Anche se permette al programma di essere particolarmente performante, si può avere la necessità in alcuni casi di eseguire delle determinate istruzioni prima di altre o in un certo ordine.

Per risolvere il problema della programmazione non deterministica si usa Async. Async è un modulo che fornisce delle funzioni per lavorare con il JavaScript in modo asincrono, che seguono la convenzione di Node.js di mettere una singola callback come ultimo parametro della funzione.

L'esempio seguente illustra come stampare i due numeri nell'ordine corretto utilizzando la libreria async:

```
actionArray = [
      function one(cb) {
            someAsyncFunction(params, function(err, results) {
                  if (err) {
                        cb(new Error("There was an error"));
                  }
                  console.log("one");
                  cb(null);
            });
      },
      function two(cb) {
            console.log("two");
            cb(null);
      }
]
async.series(actionArray);
```

Codice 3.9: operazioni sincrone

Così facendo il risultato sarà quello aspettato, ossia `one` verrà stampato sicuramente prima di `two`.

4. Long Polling

5. Implementazione

6. Conclusioni

A. Installazione e Uso

B. Screenshot