UNIVERSITÀ DEGLI STUDI DI CATANIA
DIPARTIMENTO DI INGEGNERIA ETETTRICA ELETTRONICA E NFORMATICA
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

Giuseppe Testa 1000022605
Luigi Fontana 1000037171

Progetto Advanced Programming Languages

RELAZIONE FINALE

Ch.ma prof.ssa Vincenza Carchiolo
Ch.mo prof. Giuseppe Mangioni

Anno Accademico 2022 – 2023

Indice

1. Introduzione

L’applicazione HeyDieei sviluppata da noi nasce con l’obiettivo di permettere agli utenti registrati, tramite applicazione desktop, di poter valutare gli argomenti delle materie dei vari corsi appartenenti al corso di Laurea Magistrale LM-32 in Ingegneria informatica e di valutare i Professori del medesimo corso. L’applicazione prevede inoltre la visualizzazione delle statistiche.

La nostra applicazione prevede una fase di registrazione e dopo questa sarà possibile sfruttare i servizi.

Il software si compone di quattro diverse componenti:

- Il client (FE), scritto interamente in **C#** si basa sulla gestione degli eventi implementata per mezzo di Windows Forms, cioè una user interface basata su .NET Framework.
- Il BE del nostro progetto è stato realizzato interamente in **Go** ed espone delle API REST grazie all'utilizzo del framework "Gin Gonic Web Framework"
- Un modulo per il Web Scraping, scritto in **Python** che legge le informazioni dal sito dieei e le salva sul DB.

Lo strato di persistenza si appoggia su un DB di tipo relazionale, ovvero MySQL.

Per quanto concerne la suddivisione del lavoro svolto, lo studente Testa si è occupato del Front-End scritto in C#, lo studente Fontana si è occupato dello sviluppo dell’algoritmo di Scraping, infine il Back-End scritto in GO è stato curato da entrambi gli studenti.

2. Front-End (C#)

Lo sviluppo del client si basa interamente sulla gestione di eventi; infatti, ad ogni azione dell'utente corrisponderà un evento che grazie agli handler, presenti nei vari forms, sarà gestito e permetterà di eseguire un'azione/operazione specifica.

Il client si compone di diversi forms:

- ***Welcome***: form che permette di effettuare la registrazione e il login all'interno dell'applicazione;
- ***Home***: form che visualizza il menu principale da cui, cliccando i vari tasti, è possibile eseguire le varie funzioni implementate.

All'interno di Home saranno chiamati altri forms definiti nella cartella FunctionalityForms i quali contengono la visualizzazione delle varie funzioni implementare. Questi sono i form creati:

- ***HomePage***: form in cui vengono visualizzate le informazioni dell'utente registrato;
- ***ValutaMaterie***: form permette di esprimere una votazione da 0 a 5 ai vari argomenti della materia che viene scelta attraverso un combobox che contiene tutte le materie. Ogni materia avrà i suoi argomenti da valutare. All'interno di questo form viene chiamato il form *DomandeMaterie*, presente nella cartella QuestionForms, che gestisce la creazione e la visualizzazione degli elementi delle varie domande;
- ***ValutaProfessori:*** permette di sottoscrivere un form per poter valutare i vari Professori basandosi su domande uguali per tutti con un voto che va da 0 a 5. All'interno di questo form viene chiamato il form *DomandeProfessori*, presente nella cartella QuestionForms, che gestisce la visualizzazione e la votazione delle domande relative ai professori.
- ***Statistiche***: permette di visualizzare le classifiche personali e globali. All'interno di questo form vengono chiamati ulteriori due form, StatGenerali e StatsUtenti, che gestiscono la chiamata e la visualizzazione delle statistiche.
- ***CheProf***: permette di scoprire attraverso 12 domande a quale professore si è più affine.

I form che vengono chiamati da altri form, verranno visualizzati in una label che attraverso una specifica funzione che prima di tutto controlla che non siano presenti componenti al suo interno attivi (in caso siano presenti viene invocato il Dispose() che consente di liberare le risorse occupate) e successivamente permette la visualizzazione del form all'interno di quella specifica label.

```
private void OpenStats(object child)
{
    if (statsPanel.Controls.Count > 0)
    {
        foreach (Control contr in statsPanel.Controls)
            contr.Dispose();

        statsPanel.Controls.Clear();
    }

    if (child != null)
    {
        Form form = child as Form; //Child come tipo Form
        form.TopLevel = false;
        form.FormBorderStyle = FormBorderStyle.None;
        form.Dock = DockStyle.Fill;
        statsPanel.Controls.Add(form);
        statsPanel.Tag = form;
        form.Show();
        statsPanel.Show();

    }
}
```

Figura 1 - Esempio di funzione che apre form

Gestione collegamento con Go

Per poter gestire le chiamate alle varie Rest API create nel Back-End scritto in Go, è stata creata la classe BEClient.cs che contiene tutte le funzioni che effettuato le richieste. Per fare le richieste http è stata usata la classe HttpClient che fornisce le varie funzioni per poter leggere i dati ricevuti dal back end.

```
1 riferimento
public UserInfo GetUserInfobyUsername(string username)
{
    try
    {
        string json;
        var response = _httpClient.GetAsync("/user/" + username).Result;

        json = response.Content.ReadAsStringAsync().Result;
        Console.WriteLine(json);

        if (string.IsNullOrEmpty(json))
        {
            throw new Exception("Generic error calling API");
        }
        UserInfo userInfo = JsonConvert.DeserializeObject<UserInfo>(json);
        Console.WriteLine(userInfo);

        return userInfo;
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex);
        return null;
    }
}
```

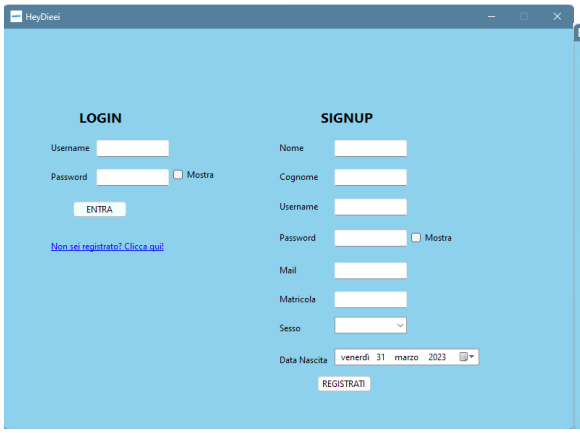
Figura 2 - Esempio di richiesta Http

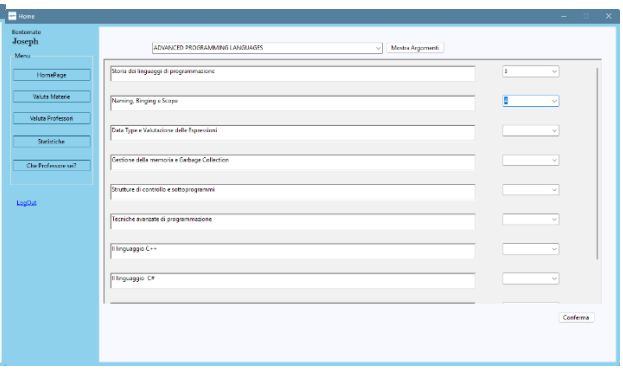
Per poter gestire i vari dati provenienti da Go, sono state create delle classi con dentro i vari campi delle strutture dati. In questo mondo quando riceviamo il Json da parte di Go lo possiamo gestire in maniera corretta e anche nelle altre classi sarà possibile gestire i vari dati ricevuti o da inviare.

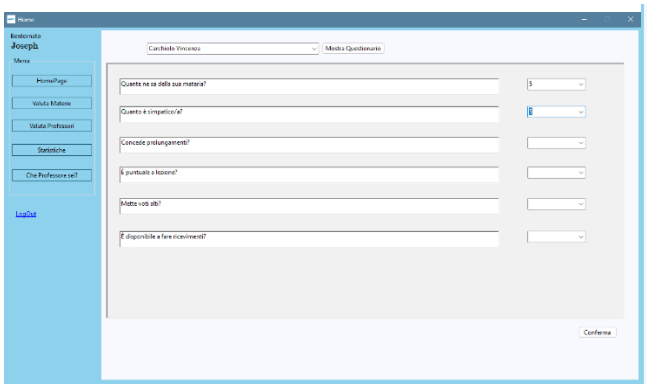
```
[Serializable]
14 riferimenti
public class UserInfo
{
    [JsonProperty("Nome")]
    2 riferimenti
    public string Nome { get; set; }


    [JsonProperty("Cognome")]
    2 riferimenti
    public string Cognome { get; set; }


    [JsonProperty("Username")]
    13 riferimenti
    public string Username { get; set; }


    [JsonProperty("Password")]
    2 riferimenti
    public string Password { get; set; }


    [JsonProperty("Email")]
    2 riferimenti
    public string Email { get; set; }


    [JsonProperty("Matricola")]
    2 riferimenti
    public string Matricola { get; set; }


    [JsonProperty("Sesso")]
    2 riferimenti
    public string Sesso { get; set; }
```

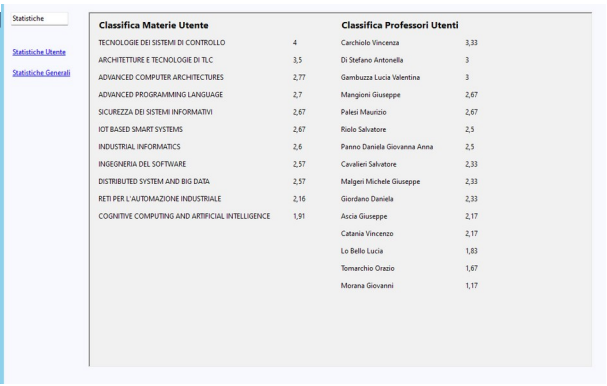
Figura 3 - Esempio di classe

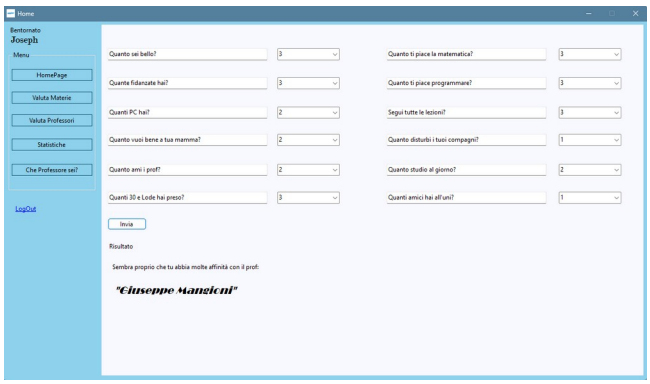
Galleria screen











3. Back-end (Go)

Il BE del nostro progetto è stato realizzato interamente in Go ed espone delle API REST grazie all'utilizzo del framwork "Gin Gonic Web Framework" che consente di creare un web server in grado di esporre le API utilizzando delle route specifiche.

La logica delle varie API esposte dal BE non viene svolta direttamente nel file *main.go*, ma per avere una corretta suddivisione delle responsabilità sono stati creati dei services, cioè dei file in cui sono presenti delle funzioni pubbliche che consentono di gestire le entità presenti sul DB.

Struttura del codice

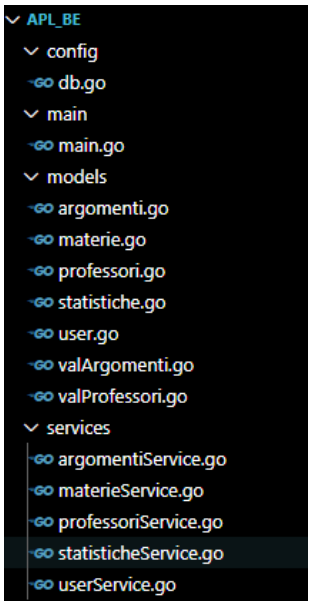


Figura 4 - Struttura del codice in Go

Sono stati creati i 5 diversi tipi di services, ognuno dei quali gestisce una specifica categoria di funzionalità relative a professori, materie, argomenti e statistiche.

In ognuno dei service sono quindi presenti le funzioni pubbliche che vengono richiamate dagli handler definiti nel *main.go.*

```go
func FindArgomenti(materiaID string)[]models.Argomenti{
    db := config.DatabaseConnection()
    var all_argomenti []models.Argomenti

    rows, err := db.Query("SELECT * FROM ARGOMENTI WHERE materiaID = ?", materiaID)
     if err != nil {
            log.Fatal(err)
        }
    defer rows.Close()

    var arg models.Argomenti
    for rows.Next(){
        if err:= rows.Scan(&arg.ArgomentoID, &arg.Nome, &arg.MateriaID); err != nil{
            log.Fatal(err)
        }
        all_argomenti = append(all_argomenti, arg)
    }

    if err := rows.Err(); err != nil {
        log.Fatal(err)
    }
    return all_argomenti
}

func Valutation_Argomenti(valutation []models.ValArgomenti) (response int) {
    db := config.DatabaseConnection()
    var verify models.ValArgomenti
    var id int
    response = 200
```

Figura 5 - Esempio di service

Per creare classi e quindi lavorare ad oggetti, in Go è necessario usare delle struct con la sintassi seguente:

```go
package models

type Argomenti struct {
    ArgomentoID uint16
    Nome string
    MateriaID uint32
}
```

Figura 6 - Esempio di models

In questo modo abbiamo definito, all'interno della directory "*models/*" diversi oggetti/modelli utilizzati in tutta l'applicazione.

REST API

Il FE quindi non effettuerà delle chiamate dirette verso il DB, ma grazie alle API esposte sarà in grado di effettuare le CRUD sulle funzionalità presenti nell'applicazione. Si è cercato quindi di rispettare le caratteristiche dei metodi HTTP:

- **GET**: utilizzato semplicemente per recuperare dati dal DB.
- **POST**: effettuare il caricamento delle votazioni sul DB.
- **PUT**: inserire nuovi utenti.

```go
func main() {
    r := gin.Default()

    //User API
    r.PUT("/user/signup", createUser)
    r.POST("/user/login", PostUserPass)
    r.GET("user/:username", getUserbyUsername)

    //API Materie
    r.GET("/Materie", getMaterie)
    r.GET("/Argomenti/:materiaID", getArgomenti)
    r.POST("/Val_Argomenti", postValArgomenti)

    //API Professori
    r.GET("/Professori", getProfessori)
    r.POST("/Val_Professori", postValProfessori)

    //API Statistiche
    r.GET("Statistiche/argomenti/:user", getUserStatArgomenti)
    r.GET("Statistiche/professori/:user", getUserStatProfessori)
    r.GET("Statistiche/argomenti/", getStatArgomenti)
    r.GET("Statistiche/professori/", getStatProfessori)

    //API Che Prof Sei
    r.POST("/Cheprof", postCheProf)

    r.Run("localhost:8080") // listen and serve on 0.0.0.0:8080
}
```

Figura 7 - API

Nel nostro caso, infatti, come possiamo vedere, abbiamo registrato diversi path (diverse route) e ad ognuno di essi abbiamo assegnato un handler per la gestione della richiesta HTTP.

4. Web Scraping (Python)

Il modulo per effettuare lo scraping sul sito del Dieei è realizzato interamente in Python e sfrutta la libreria esterna BeautifulSoup che offre svariati metodi per recuperare informazioni dai DOM ricavati mediante richieste HTTP.

Infatti, come è possibile vedere dal file sorgente "scraper.py", una volta istanziato un oggetto di tipo BeautifulSoup passando in ingresso la risposta HTTP (che sarebbe una pagina HTML) possiamo ricavare tutte le informazioni che servono. Conclusa la fase di scraping, i dati vengono conservati in un DB relazionale.

Per lanciare il modulo di Web Scraping è necessario scaricare tutte le dipendenze utilizzate dagli script in python; le dipendenze sono specificate nel file *requirements.txt* e tramite il comando:

- `pip3 install -r requirements.txt`

Una volta che tutte le librerie esterne sono state scaricante mediante il gestore dei pacchetti di python, pip3, dopo aver creato le tabelle nel db con il comando:

- `python create_tables.py`

è possibile lanciare lo scraper con il comado:

- `python scarper.py`

Eseguendo lo script:

- `python fill_tables.py`

è possibile riempire di dati le tabelle delle valutazioni presenti nel DB.

Estrazione dei dati

Lo scraper è suddiviso in funzionalità dove ciascuna effettua uno scraping sul sito del dipartimento raccogliendo i dati relativi ai professori, alle materie e agli argomenti delle materie per poi memorizzare le informazioni ottenute sul Database.

```
<tr><td class="table-active" colspan="3"><b>1&deg; anno</b></td></tr>
<tr><td><a href="/corsi/lm-32/insegnamenti?seuid=D356C96F-79B6-4361-A129-7EB15CD435CC">1015336 - ADVANCED COMPUTER ARCHITECTURES</a></td><td></td><td><a href="/
<tr><td><a href="/corsi/lm-32/insegnamenti?seuid=675FE57B-8D73-44B0-8182-AF4E87D87061">1001628 - ARCHITETTURE E TECNOLOGIE DEI SISTEMI DI TELECOMUNICAZIONI</a><
<tr><td><a href="/corsi/lm-32/insegnamenti?seuid=8573BAE2-E458-4C07-9DD7-CB28DB57E5F4">1001628 - ARCHITETTURE E TECNOLOGIE DEI SISTEMI DI TELECOMUNICAZIONI</a><
<tr><td><a href="/corsi/lm-32/insegnamenti?seuid=E93D6374-A612-44F6-AA55-D4C3AC42D1E9">1001224 - INGEGNERIA DEL SOFTWARE</a></td><td></td><td><a href="/_uni_mod
<tr><td><a href="/corsi/lm-32/insegnamenti?seuid=C75BE68E-3EA8-4638-AFDE-B40A42736E2C">1002067 - RETI PER L'AUTOMAZIONE INDUSTRIALE</a></td><td></td><td><a href
<tr><td><a href="/corsi/lm-32/insegnamenti?seuid=F6E507E1-9DCA-44B1-B3EA-36C0793C3A13">1002043 - SICUREZZA DEI SISTEMI INFORMATIVI</a></td><td></td><td><a href=
<tr><td><a href="/corsi/lm-32/insegnamenti?seuid=A47086F9-8A38-4C56-991E-374C70BB433C">1001928 - TECNOLOGIA DEI SISTEMI DI CONTROLLO</a></td><td></td><td><a hre
<tr><td class="table-active" colspan="3"><b>2&deg; anno</b></td></tr>
<tr><td><a href="/corsi/lm-32/insegnamenti?seuid=0BB7441C-4EA3-4255-87DB-4638439EC455">1015347 - ADVANCED PROGRAMMING LANGUAGES</a></td><td></td><td><a href="/_
<tr><td>1015347 - ADVANCED PROGRAMMING LANGUAGES</td><td></td><td><a href="/_uni_mod/docenti_open_from_cf.php?cf=TU5HR1BQNzFDMjRDMzUxTw==">MANGIONI GIUSEPPE</a>
<tr><td><a href="/corsi/lm-32/insegnamenti?seuid=B54D365B-8407-4600-A803-FA2CB383C88D">1015335 - COGNITIVE COMPUTING AND ARTIFICIAL INTELLIGENCE</a></td><td></t
<tr><td><a href="/corsi/lm-32/insegnamenti?seuid=B701A37C-08C4-4B2D-8598-A421B250A6F6">1016152 - DISTRIBUTED SYSTEMS AND BIG DATA</a></td><td></td><td><a href="
<tr><td>1016152 - DISTRIBUTED SYSTEMS AND BIG DATA</td><td></td><td><a href="/_uni_mod/docenti_open_from_cf.php?cf=TVJOR05ONzdNMThGMjU4SQ==">MORANA GIOVANNI</a>
<tr><td><a href="/corsi/lm-32/insegnamenti?seuid=4A06EAE0-CD27-4E3B-B676-F3D60168BA9B">1015361 - INDUSTRIAL INFORMATICS</a></td><td></td><td><a href="/_uni_mod/
<tr><td><a href="/corsi/lm-32/insegnamenti?seuid=7CA43046-251F-4E66-B770-479A5507785E">1015340 - INTERNET OF THINGS BASED SMART SYSTEMS</a></td><td></td><td><a
<tr><td><a href="/corsi/lm-32/insegnamenti?seuid=059191CF-DE8C-43BD-B5EB-86D89C1383BD">1015340 - INTERNET OF THINGS BASED SMART SYSTEMS</a></td><td></td><td><a
</table></div>
```

Figura 8 - Html del sito dieei

```
55
56  def programs_scraping():
57      url = site_url + "/corsi/lm-32/programmi"
58      response = requests.get(url)
59      soup = BeautifulSoup(response.content, "html.parser")
60      reduce = {}
61      programs_href = soup.find_all("a", attrs={'href': re.compile("^/corsi/lm-32/i")})  # script per prendere i link alle
62      for href in programs_href:
63          url = site_url + href.get("href")
64          reduce[url] = href.text.split(" - ")
65
66      first_key = next(iter(reduce))
67      reduce.pop(first_key)
68      return reduce
69
70  def programs_Sql_Load():
71      global programs_info
72      programs_info = programs_scraping()
73      for k,v in programs_info.items():
74          #print(f"{k} -> url:{v[0]} materia: {v[1]}")
75          sql = """INSERT INTO MATERIE (materiaID, nome, url) VALUES (%s,%s,%s);"""
76          val = (v[0], v[1], k)
77          mycursor.execute(sql, val)
78          mydb.commit()
79
80
```

Figura 9 - Funzioni utilizzate per fare scraping

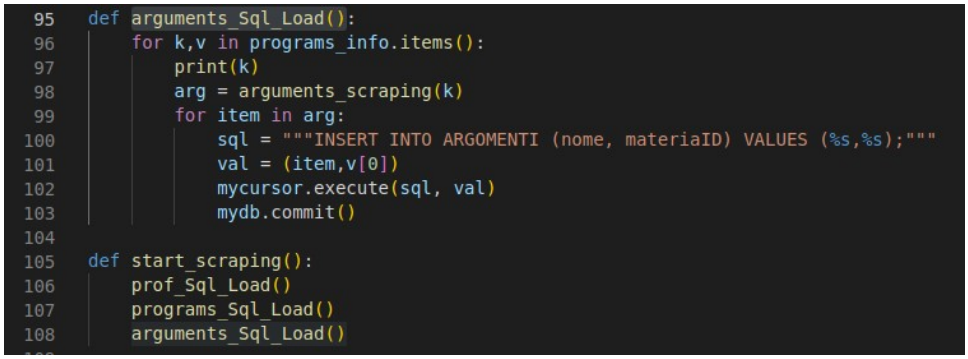
Grazie ad un oggetto di tipo **BeautifulSoup** (fornito dalla libreria *bs4*) istanziato mediante il suo costruttore, che vuole in ingresso una risorsa su cui effettuare il parsing è possibile cercare tra i diversi nodi della pagina HTML per recuperare le informazioni desiderate.

Salvataggio dei dati

Una volta finito lo scraping dei dati, questi ultimi verranno memorizzati sul DB grazie alle funzioni:

- programs_Sql_Load()
- prof_Sql_Load()
- arguments_Sql_Load()

Che interagiscono con il database effettuando delle query di inserimento dati, al fine della persistenza dei dati.



```
95  def arguments_Sql_Load():
96      for k,v in programs_info.items():
97          print(k)
98          arg = arguments_scraping(k)
99          for item in arg:
100             sql = """INSERT INTO ARGOMENTI (nome, materiaID) VALUES (%s,%s);"""
101             val = (item,v[0])
102             mycursor.execute(sql, val)
103             mydb.commit()
104
105 def start_scraping():
106     prof_Sql_Load()
107     programs_Sql_Load()
108     arguments_Sql_Load()
```

Figura 10 - Esempio di funzione che effettua le query