

Ingegneria del Software

UNITN - Dipartimento di Ingegneria e Scienza dell'Informazione
Anno Accademico 2022-2023
Battocchio N. - Bocchi G. - Miazzo L.



**UNIVERSITY
OF TRENTO**

CarMeeTiN

— Sviluppo Applicazione



Versione del documento

Versione	Data	Modifiche
0.1	9/12/2022	Creazione documento e aggiunta degli user flows
0.2	13/12/2022	Aggiunta project structure e prime APIs
0.3	15/12/2022	Aggiunta documentazione APIs
1.0	20/12/2022	Testing eseguito e aggiunta delle ultime sezioni
1.1	27/12/2022	Correzioni minori

INDICE

SCOPO DEL DOCUMENTO	6
<hr/>	
1. USER FLOWS	7
1.1 Utente Anonimo	8
1.2 Utente Autenticato	9
<hr/>	
2. APPLICATION IMPLEMENTATION AND DOCUMENTATION	10
2.1 Project Structure	11
2.1.1 Project directory: __test__	12
2.1.2 Project directory: components	12
2.1.3 Project directory: data	12
2.1.4 Project directory: lib	12
2.1.5 Project directory: pages	12
2.1.6 Project directory: public	13
2.1.7 Project directory: theme	13
2.1.8 Project file: .env.local	13
2.1.10 Project file: .gitignore	13
2.1.11 Project file: jest.config.js	14
2.1.12 Project file: package.json	14
2.1.13 Project file: README.md	14
2.2 Project Dependencies	15
2.3 Project Data and Database	16
2.3.1 Project Data	16
- root-ca.crt	16
- cars.json	16
- emailTemplates/pwdRecovery.hbs	16
- emailTemplates/verifyEmail.hbs	16
2.3.2 Project Database	17
- UserModel	17
- RecoverPwdModel	18
- EventModel	18
- CarModel	19
2.4 Project APIs	20
2.4.1 Resources Extraction from the Class Diagram	20
- Utente	20
- Evento	21
- Auto	21

2.4.2 Resource Models	24
- Utente	25
- Evento	26
- Auto	26
2.5 Sviluppo API	28
2.5.1 Signup (<i>Registrazione</i>)	28
2.5.2 Login (<i>Accesso al sistema</i>)	29
2.5.3 Logout (<i>Uscita dal sistema</i>)	30
2.5.4 requestRecoverPassword (<i>Richiesta recupero password</i>)	31
2.5.5 recoverPassword (<i>Recupero password</i>)	32
2.5.6 updatePassword (<i>Modifica password</i>)	33
2.5.7 verifyEmail (<i>Verifica validità indirizzo email</i>)	34
2.5.8 car (<i>Gestione veicolo</i>)	35
2.5.9 user (<i>Gestione sessione utente</i>)	37
2.5.10 updateUser (<i>Modifica informazioni utente</i>)	38
2.5.11 event (<i>Gestione eventi</i>)	39
2.5.12 subscribeEvent (<i>Iscrizione ad un evento</i>)	41
2.5.13 getUsername (<i>ottieni nome utente</i>)	42
2.5.14 getCars (<i>ottieni lista veicoli</i>)	43
2.5.15 myAgenda (<i>ottieni lista eventi</i>)	44
3. API DOCUMENTATION	45
4. FRONTEND IMPLEMENTATION	47
4.1 Pagina Home	47
4.2 Pagina myGarage	48
4.3 Pagina myEvents	49
4.4 Pagina myAgenda	50
4.5 Pagina Registrazione	51
4.6 Pagina Login	52
4.7 Pagina Richiesta Reset Password	53
4.8 Pagina Reset Password	54
4.9 Pagina Aggiornamento Password	55
4.10 Pagina Aggiornamento dati dell'utente	56
4.11 Pagina Aggiunta di un'auto	57
4.12 Pagina Creazione di un evento	58
4.13 Pagina Evento	59
4.14 Pagina Ricerca Utente/Evento	59
4.15 Pagina Informativa privacy	59

4.16 Pagina Modifica di un evento	60
4.17 Pagina Modifica di un'auto	60
5. GITHUB REPOSITORY AND DEPLOYMENT INFO	61
6. TESTING	62
6.1 Test API	62
<i>6.1.1 test /api/car/getCars</i>	62
<i>6.1.2 test /api/car</i>	63
<i>6.1.3 test /api/event</i>	63
<i>6.1.4 test /api/user/getUsername</i>	63
<i>6.1.5 test /api/user/login</i>	64
<i>6.1.6 test /api/user/signup</i>	64
6.2 Code coverage, test suites e tests	65

SCOPO DEL DOCUMENTO

Il presente documento riporta tutte le informazioni necessarie per lo sviluppo di una parte dell'applicazione CarMeeTiN.

In particolare, presenta tutti gli artefatti necessari per realizzare i servizi di gestione degli eventi e di gestione del garage degli utenti dell'applicazione CarMeeTiN.

Partendo dalla descrizione degli user flow legati al ruolo dell'utente dell'applicazione, il documento prosegue con la presentazione delle API necessarie (tramite l'API Model e il Modello delle risorse) per poter creare, modificare e visualizzare gli eventi.

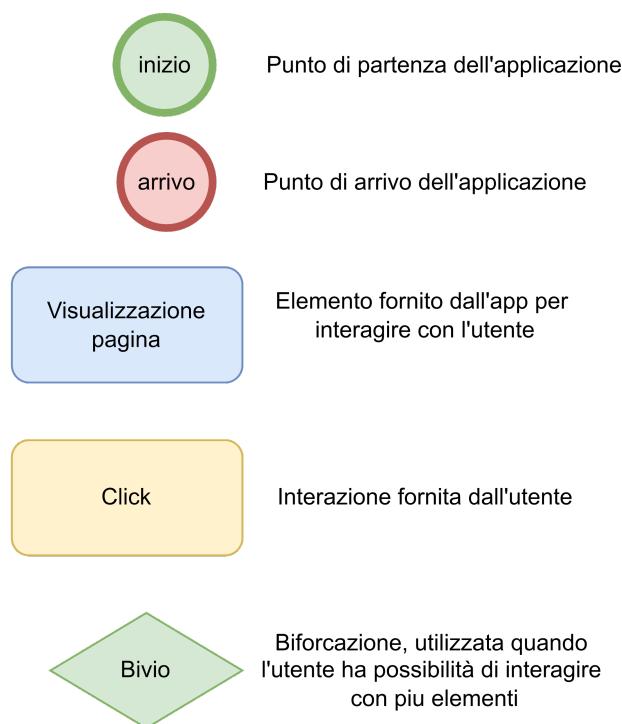
Le APIs fornite per interagire con l'applicazione saranno accompagnate da una descrizione in linguaggio naturale per aiutare la comprensione del loro utilizzo.

Nel documento è stato riportata la documentazione, i test effettuati e una descrizione delle funzionalità fornite per ogni API realizzata.

1. USER FLOWS

In questa sezione del documento di sviluppo vengono riportati gli *User Flows*. Lo scopo degli *User Flows* è quello di specificare come l'utente si potrà muovere e come potrà interagire con il sito web. Sono stati individuati due tipi diversi di *User flow*, in particolare uno specifico per **l'Utente Anonimo**, e l'altro per **l'Utente Autenticato**.

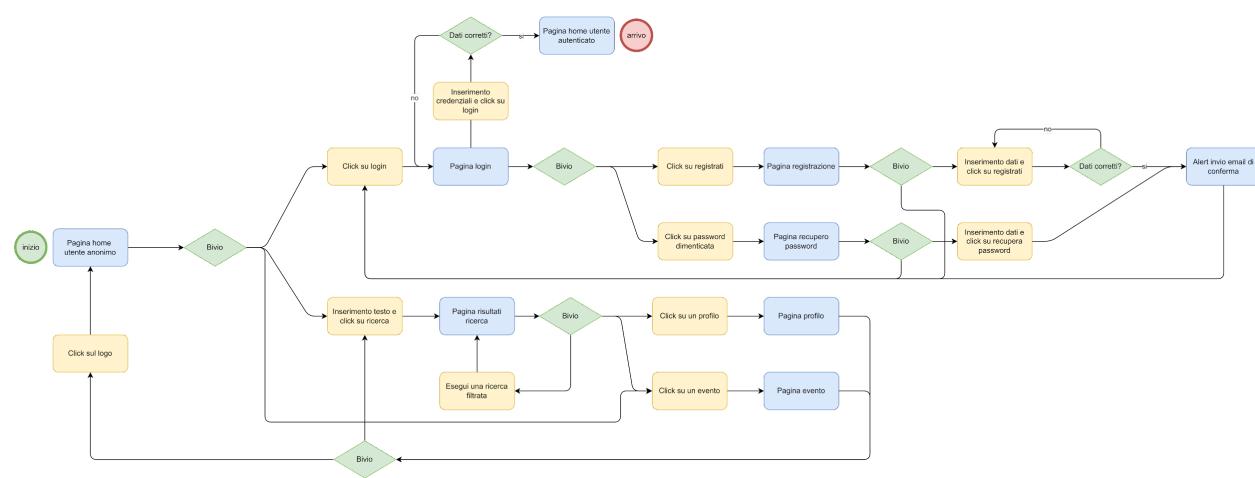
Di seguito è esposta una legenda per indicare i simboli utilizzati:



Per la visione di questa immagine ad alta risoluzione si veda l'[allegato AA](#).

1.1 Utente Anonimo

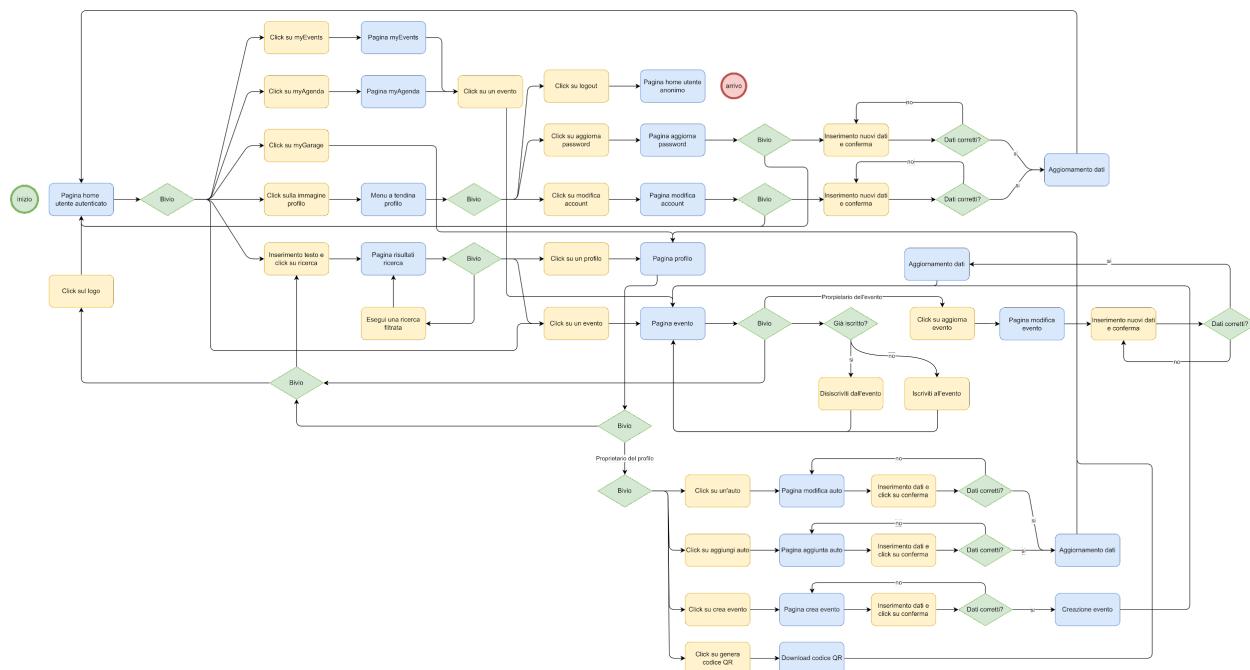
Questo *User Flow* è specifico per l'**Utente Anonimo**. Si ricorda che in ogni momento della sua navigazione, l'utente ha la possibilità di accedere alla pagina Home, accettare i cookies, effettuare le ricerche, e modificare il tema della web app, tramite una *Navbar* che è presente in ogni pagina del sito web. Per semplicità queste interazioni sono state rimosse per rendere l'*User Flow* più leggibile. Le interazioni dell'**Utente Anonimo** sono ovviamente più limitate rispetto a quelle dell'**Utente Autenticato**, con a disposizione solamente la visualizzazione di **profili utenti** ed **eventi**, ed eventualmente la **registrazione**, il **recupero password**, e il **login**.



Per la visione di questa immagine ad alta risoluzione si veda l'[allegato AB](#).

1.2 Utente Autenticato

Questo *User Flow* è specifico per l'**Utente Autenticato**. Si ricorda, come per l'**Utente Anonimo**, che in ogni momento della sua navigazione, l'utente ha la possibilità di accedere alla pagina Home, accettare i cookies, effettuare le ricerche, e modificare il tema della web app, tramite una *Navbar* che è presente in ogni pagina del sito web. Inoltre, si fa notare che ogni azione che richiede la modifica dei dati presenti sul Database di sistema, richiede precedentemente di aver verificato l'account. Per semplicità queste interazioni sono state rimosse per rendere l'*User Flow* più leggibile. Le interazioni con le funzionalità di accesso al sistema sono ovviamente diverse rispetto a quelle dell'**Utente Anonimo**, con a disposizione solamente la possibilità di effettuare il **Logout**. L'utente ha però a disposizione la possibilità di **modificare** le informazioni del **proprio account**, oppure **modificare la password**; può **aggiungere**, **modificare** oppure **eliminare un'auto**. Inoltre, l'utente autenticato ha la possibilità di **creare**, **modificare**, **annullare** oppure **cancellare un evento**, infine potrà **iscriversi** oppure **disiscriversi** dal suddetto **evento**.



Per la visione di questa immagine ad alta risoluzione si veda l'[allegato AC](#).

2. APPLICATION IMPLEMENTATION AND DOCUMENTATION

Nelle sezioni precedenti abbiamo identificato le varie features che devono essere implementate per la nostra applicazione con un'idea di come il nostro utente finale può utilizzarle nel suo flusso applicativo.

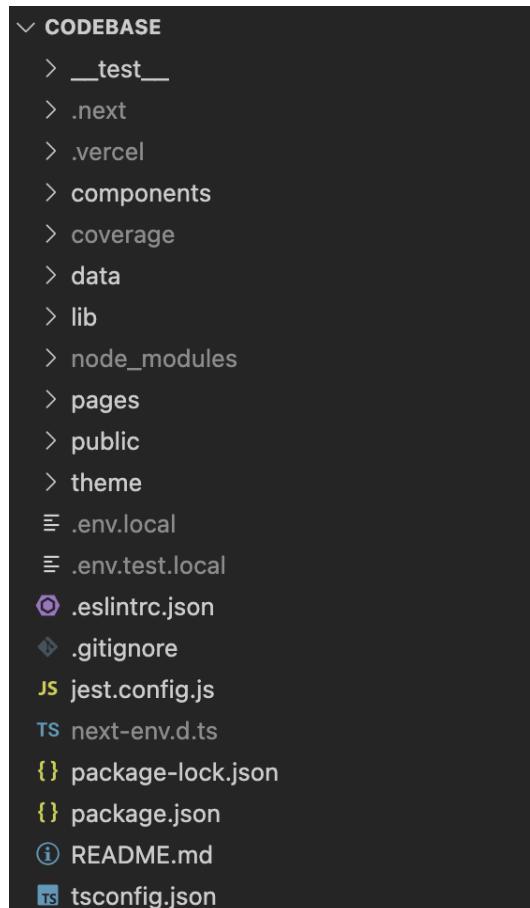
L'applicazione **CarMeeTiN** è stata sviluppata utilizzando [Next.js](#) vers. 12.2.5, un framework Javascript free-open source basato su [React](#) vers. 18.2.0.

2.1 Project Structure

Il software di controllo della versione utilizzato è [Git](#), come repository remota è stato utilizzato [Github](#), condividendo il codice tramite un'[organizzazione](#) condivisa solo ai diretti interessati.

Data la natura del framework Next.js, il progetto è contenuto in unica repository “[CodeBase](#)”.

All'interno di questa possiamo trovare la seguente struttura delle cartelle:



Le cartelle **.next**, **.vercel**, **coverage**, **node_modules**, insieme ai file **package-lock.json**, **next-env.d.ts** e **.eslintrc.json** sono artefatti automaticamente generati dal framework, e pertanto non sono di nostro interesse.

2.1.1 Project directory: test

Questa cartella contiene il codice dei test da eseguire, meglio specificati nella sezione dedicata al [testing](#).

2.1.2 Project directory: components

Questa cartella contiene le componenti di **Front-End** comuni a tutte le pagine, come la **Navbar**, il **Layout** generale delle pagine, il **Footer**, un modulo per i form di inserimento delle credenziali e uno per inserire le informazioni richieste per la procedura che si intende eseguire (es. creazione di un nuovo evento).

2.1.3 Project directory: data

Questa cartella contiene alcune informazioni statiche, come il **certificato** per verificare l'integrità del **Database**. Inoltre qui si possono trovare i **template** delle **Email** di **conferma account** e di **recupero password**.

2.1.4 Project directory: lib

Questa cartella contiene delle librerie interne per facilitare la **connessione al Database**, la **gestione delle credenziali e degli accessi**, gli **schemi di Yup** utilizzati nelle varie pagine, l'**invio delle E-mail** e le operazioni sui **cookie di sessione**.

2.1.5 Project directory: pages

Questa cartella contiene le **routes** delle varie **pagine** e delle **API**. Ogni pagina è conservata in un file Javascript, così come le API. Qui si possono trovare anche le rotte dinamiche, come “[recoverPasswordCode].js”, “[username].js”, ecc..

2.1.6 Project directory: public

Questa cartella contiene gli elementi statici delle pagine, in particolare la **favicon** e le **immagini** contenute nel sito.

2.1.7 Project directory: theme

Questa cartella contiene gli elementi utili ai **temi** e ai **colori** utilizzati dall'applicativo.

2.1.8 Project file: .env.local

Questo file contiene tutte le **variabili d'ambiente**, necessarie al corretto funzionamento dell'applicativo, come l'**username** e **password del Database**, la **chiave segreta** per criptare il **cookie di sessione**, ecc...

2.1.9 Project file: .env.test.local

Questo file contiene tutte le **variabili d'ambiente**, dedicate però all'ambiente di testing. Risulta utile, per evitare che, nel caso di testing con scrittura sul Database, quest'ultimo si riempia di dati temporanei.

2.1.10 Project file: .gitignore

Questo file è utilizzato per il controllo di versione, in particolare per **escludere** alcuni **file** o tipi di file (es .env.local, che se caricato su github esporrebbe le informazioni sensibili del sito), dall'essere caricati sulla repository remota.

2.1.11 Project file: jest.config.js

Questo file è utilizzato per la configurazione della libreria [jest](#) utilizzata nei [testing](#).

2.1.12 Project file: package.json

Questo file è utilizzato per la gestione delle “**dependency**” del framework.

2.1.13 Project file: README.md

Questo file è utilizzato per contenere **informazioni generali** sul progetto.

2.2 Project Dependencies

Per quanto riguarda le dependencies utilizzate, per il **front-end** è stata utilizzata la libreria [Chakra-ui](#) vers. 2.2.6 e [Formik](#). Nel **back-end** sono state utilizzate molteplici librerie e pacchetti, tra queste troviamo le principali [uuid](#) (per la generazione di stringhe crittograficamente randomiche), [Nodemailer](#) (per l'invio delle email automatiche), [hapi/iron](#) (come metodo per criptare il cookie di sessione), [Mongoose](#) (per la connessione al Database), [mapbox-ql](#) e [react-map-gl](#) (per il render della mappa), [Yup](#) (per la validazione delle “RegEx” sui dati inseriti dall’utente), [swr](#) (per i react hooks), e altre dependencies minori.

Per lo **sviluppo** e il **testing** (dependency development) sono state usate [esLint](#) e prettier (per la validazione delle regole di sintassi e della struttura dei file), [Jest](#) e [node-mocks-http](#) (per l’esecuzione dei testing).

2.3 Project Data and Database

2.3.1 Project Data

Per la realizzazione del sito, l'applicativo necessita di consultare alcuni file locali per il suo corretto funzionamento. Vengono elencati qui di seguito, sono consultabili anche nella repo Github del progetto, in [/data](#):

- **root-ca.crt**

Questo certificato consente al protocollo TLS di verificare l'autenticità del Database MongoDB;

- **cars.json**

Questo file contiene la lista delle combinazioni di auto possibili;

- **emailTemplates/pwdRecovery.hbs**

Questo file contiene il template dell'e-mail inviata all'utente quando richiede il reset della password;

- **emailTemplates/verifyEmail.hbs**

Questo file contiene il template dell'e-mail inviata all'utente per verificare la proprietà dell'indirizzo fornito.

2.3.2 Project Database

Il Database scelto per memorizzare i dati a lungo termine è [MongoDB](#) vers. 4.4 gestito in un server in locale. Dati i precedenti documenti, sono state individuate 4 collezioni da inserire nel Database. Anche se MongoDB non richiede uno Schema per le collezioni, si è deciso di operare in modalità “strict”, dunque è necessario fornire all'applicativo questi Schema sotto forma di *Modelli*. I modelli *Mongoose* (libreria utilizzata per connettere Javascript a MongoDB) dedicati ad ogni collezione, sono consultabili sulla repo Github del progetto, in [/lib/models](#) :

- **UserModel**

Questo modello rappresenta un utente registrato all'interno del Database. Di seguito si trova il codice di questo modello, la definizione dei campi e dei requisiti:

```
12 const UserSchema = new Schema(  
13   {  
14     id: { type: String, required: true },  
15     username: { type: String, required: true, unique: true },  
16     email: { type: String, required: true },  
17     image: { type: String, required: false },  
18     hash: { type: String, required: true },  
19     salt: { type: String, required: true },  
20     verified: { type: Boolean, required: true, index: true },  
21     lastPasswordChange: { type: Date, required: true },  
22     verifyEmailSecret: { type: String, required: false },  
23     carIds: { type: Array, required: false },  
24     eventIds: { type: Array, required: false },  
25   },  
26   { timestamps: true }  
27 );  
28  
29 UserSchema.index(  
30   { createdAt: 1 },  
31   {  
32     expireAfterSeconds: UNVERIFIED_TTL,  
33     partialFilterExpression: { verified: false },  
34   }  
35 );
```

Come si può notare, è stato creato un *indice*, accompagnato da una *partialFilterExpression*, che permette a MongoDB di eliminare autonomamente un account non verificato dopo un certo periodo di tempo specificato, in questo caso 12 ore.

- **RecoverPwdModel**

Questo modello rappresenta un utente che ha richiesto di resettare la password all'interno del Database. Di seguito si trova il codice di questo modello, la definizione dei campi e dei requisiti:

```
12  const RecoverPwdSchema = new Schema(
13  {
14    id: { type: String, required: true },
15    username: { type: String, required: true },
16    secret: { type: String, required: true },
17  },
18  { timestamps: true }
19);
20
21 RecoverPwdSchema.index({ createdAt: 1 }, { expireAfterSeconds: TTL });
```

Come si può notare, è stato creato un *indice*, che permette a MongoDB di eliminare autonomamente la richiesta dell'utente, dopo un certo periodo di tempo specificato, in questo caso 30 minuti.

- **EventModel**

Questo modello rappresenta un evento all'interno del Database. Di seguito si trova il codice di questo modello, la definizione dei campi e dei requisiti:

```
10  const EventSchema = new Schema(
11  {
12    id: { type: String, required: true },
13    ownerId: { type: String, required: true },
14    eventName: { type: String, required: true },
15    image: { type: String, required: false },
16    startDate: { type: Date, required: true },
17    finishDate: { type: Date, required: true },
18    location: { type: Object, required: true },
19    description: { type: String, required: false },
20    maxParticipants: { type: Number, required: true },
21    participants: { type: Number, required: true },
22    participantsIds: { type: Array, required: false },
23    price: { type: Number, required: false },
24    allowedCars: { type: Object, required: false },
25    gpxFile: { type: String, required: false },
26    eventState: { type: String, required: true },
27  },
28  { timestamps: true }
29);
```

- **CarModel**

Questo modello rappresenta un'auto all'interno del Database. Di seguito si trova il codice di questo modello, la definizione dei campi e dei requisiti:

```
10  const CarSchema = new Schema(  
11    {  
12      id: { type: String, required: true },  
13      ownerId: { type: String, required: true },  
14      image: { type: Array, required: false },  
15      description: { type: String, required: false },  
16      carType: { type: Object, required: true },  
17    },  
18    { timestamps: true }  
19  );
```

2.4 Project APIs

In questo paragrafo vengono presentate le API dell'applicazione CarMeeTiN.

Verranno prima descritte e successivamente verrà mostrato il codice con cui vengono implementate.

2.4.1 Resources Extraction from the Class Diagram

Analizzando il diagramma delle classi, sono state individuate tre risorse principali: l'utente, gli eventi e le auto. Sono di seguito riportate le API, divise per categoria e per metodo utilizzato:

- **Utente**

GET APIs:

- **getUsername:** questo metodo permette di ottenere l'username di un utente a partire dal suo userid.
- **user:** questo metodo ritorna la sessione dell'utente a partire dal cookie fornito.
- **logout:** questo metodo permette a un utente autenticato di diventare anonimo, rimuovendo il cookie di sessione nel proprio browser.
- **verifyEmail/{username}, verifyEmail:** questo metodo permette all'utente autenticato di verificare il proprio account, fornendo il codice segreto che gli era stato comunicato tramite messaggio di posta elettronica.

POST APIs:

- **login:** questo metodo permette all'utente anonimo di autenticarsi. Se le informazioni fornite sono corrette, allora il sistema provvederà ad inserire un cookie di sessione nel browser dell'utente.
- **requestRecoverPassword:** questo metodo permette all'utente di richiedere il reset della password, sostituendola con una nuova, nel caso abbia smarrito quella attuale. Il sistema invierà un messaggio di posta elettronica all'indirizzo specificato, con un link per accedere alla pagina di recupero.
- **signup:** questo metodo permette all'utente di creare il proprio account su CarMeeTiN. Se le informazioni fornite sono valide, allora il sistema accetterà i dati richiesti, e invierà un messaggio di posta elettronica per verificare l'indirizzo email.

PUT APIs:

- **recoverPassword:** questo metodo permette all'utente di eseguire il reset della password, sostituendola con una nuova, nel caso abbia smarrito quella attuale.

PATCH APIs:

- **updatePassword:** questo metodo permette all'utente di aggiornare la propria password.
 - **updateUser:** questo metodo permette all'utente di aggiornare i dati del proprio account.
-

● Evento**GET APIs:**

- **event/{eventid, userid}:** questo metodo permette di ottenere gli eventi richiesti.
- **myAgenda/{userid}:** questo metodo permette di ottenere l'agenda di un utente.

POST APIs:

- **event:** questo metodo permette ad un utente di creare il proprio evento, e quindi permettere ad altri utenti di interagire con esso.
- **subscribeEvent:** questo metodo permette ad un utente di iscriversi oppure disiscriversi da un evento.

PATCH APIs:

- **event:** questo metodo permette all'utente proprietario di un evento di aggiornarlo con nuove informazioni.

- **Auto**

GET APIs:

- **car/{carid, userid}**: questo metodo permette di ottenere l'auto richiesta.
- **getCars**: questo metodo permette di ottenere tutte le combinazioni possibili di produttori di auto e i rispettivi modelli.

POST APIs:

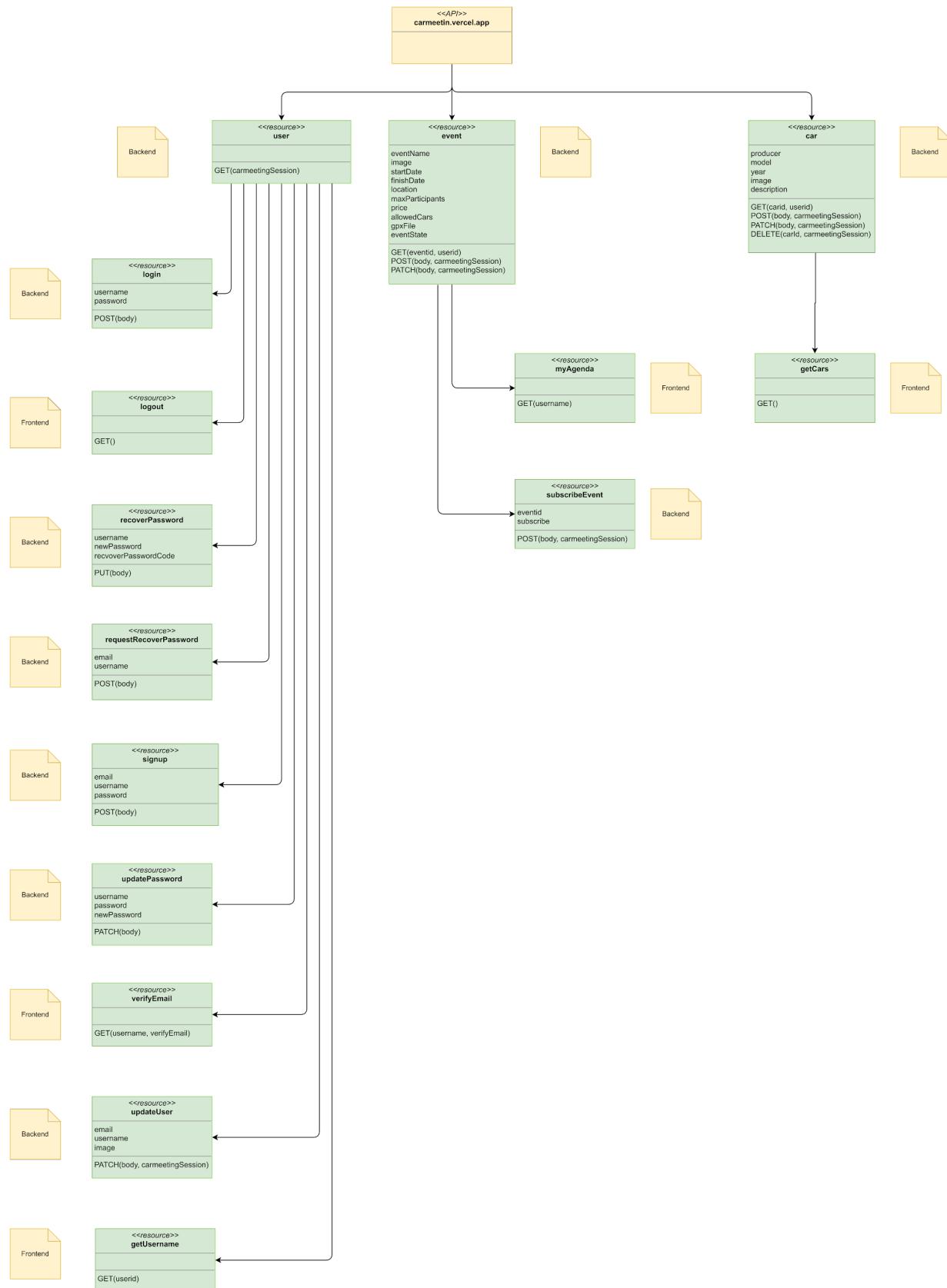
- **car**: questo metodo permette ad un utente di aggiungere un'auto, e quindi permettere ad altri utenti di interagire con essa.

PATCH APIs:

- **car**: questo metodo permette all'utente proprietario di un'auto di aggiornarla con nuove informazioni.

DELETE APIs:

- **car**: questo metodo permette all'utente proprietario di un'auto precedentemente aggiunta di eliminarla.



Per la visione di questa immagine ad alta risoluzione si veda l'[allegato B](#).

2.4.2 Resource Models

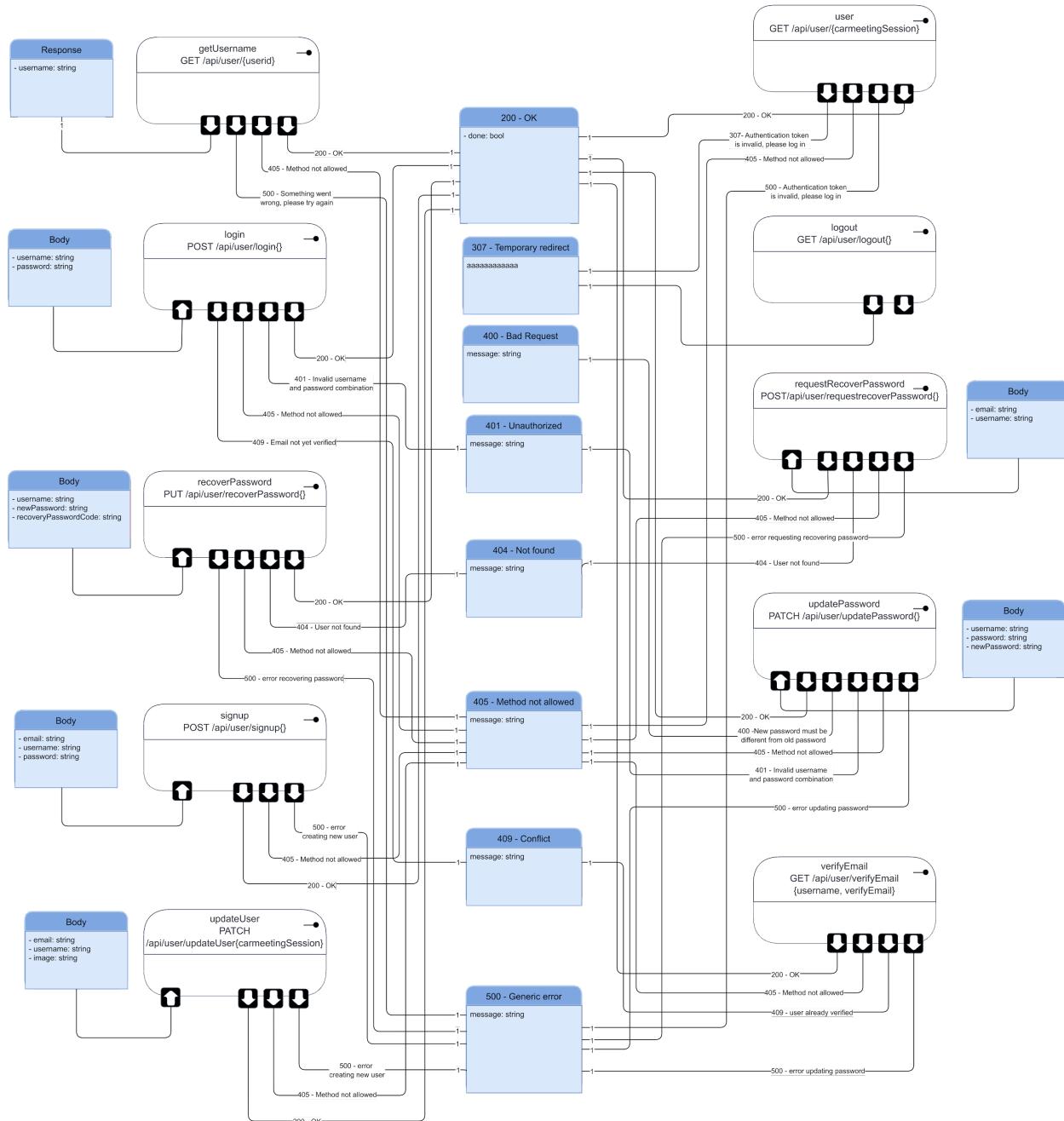
Il Resource Model viene utilizzato per astrarre e rappresentare graficamente le API necessarie per il funzionamento del sito web.

Tale modello esprime, per ogni API, il nome, il metodo con cui vi è possibile accedervi e il path URL per raggiungerla.

Ogni API ha il compito di elaborare dati, detti request body, ovvero le informazioni e i parametri che vengono passate a essa per il suo funzionamento, e vengono rappresentati graficamente da una freccia entrante nell'API. Invece il response body, ovvero la risposta dell'API, viene rappresentata graficamente da una freccia uscente dall'API.

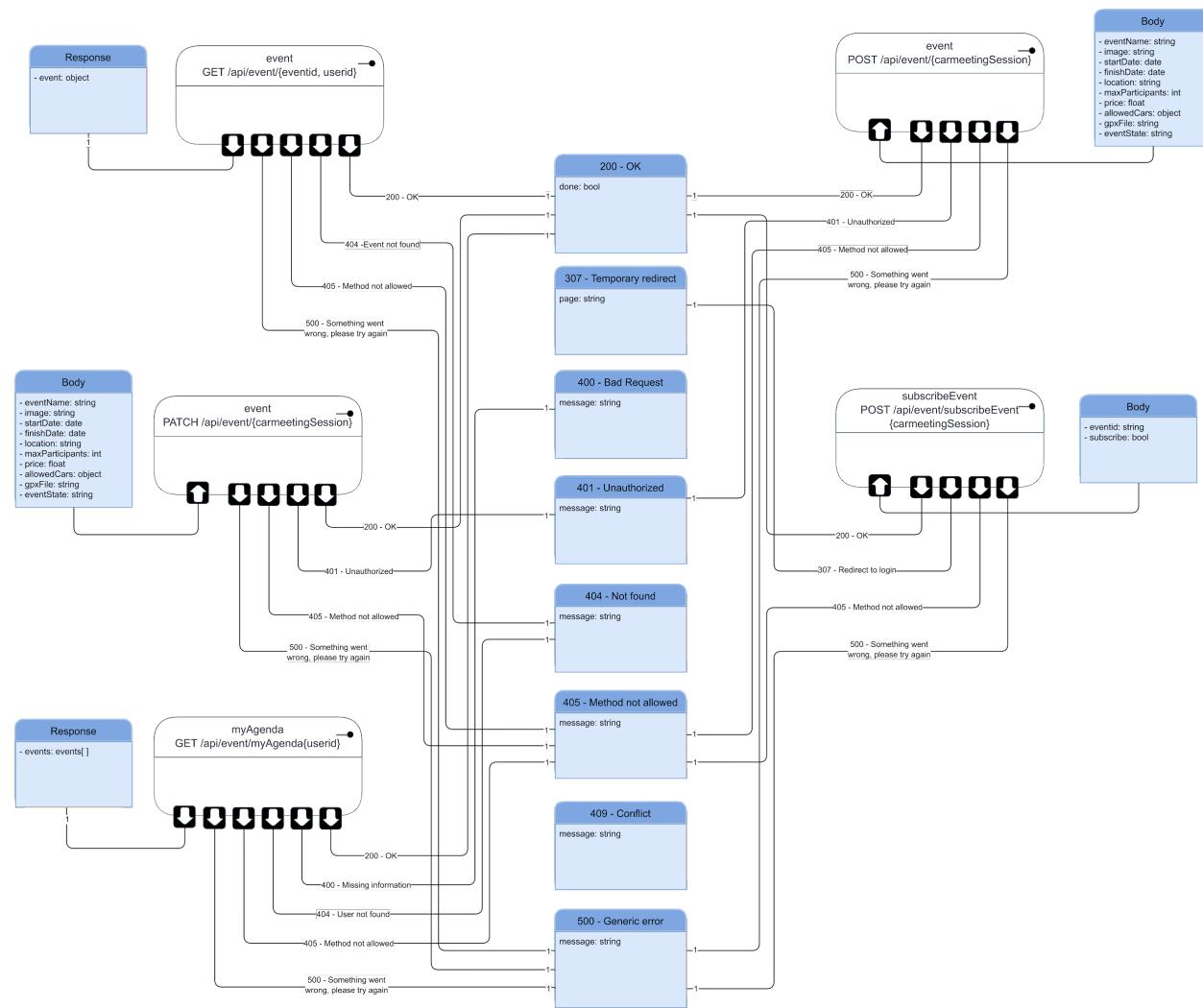
Per il sito CarMeeTiN sono state realizzate 20 API, di cui 10 per la classe Utente, 5 per la classe Evento e 5 per la classe Auto.

Utente



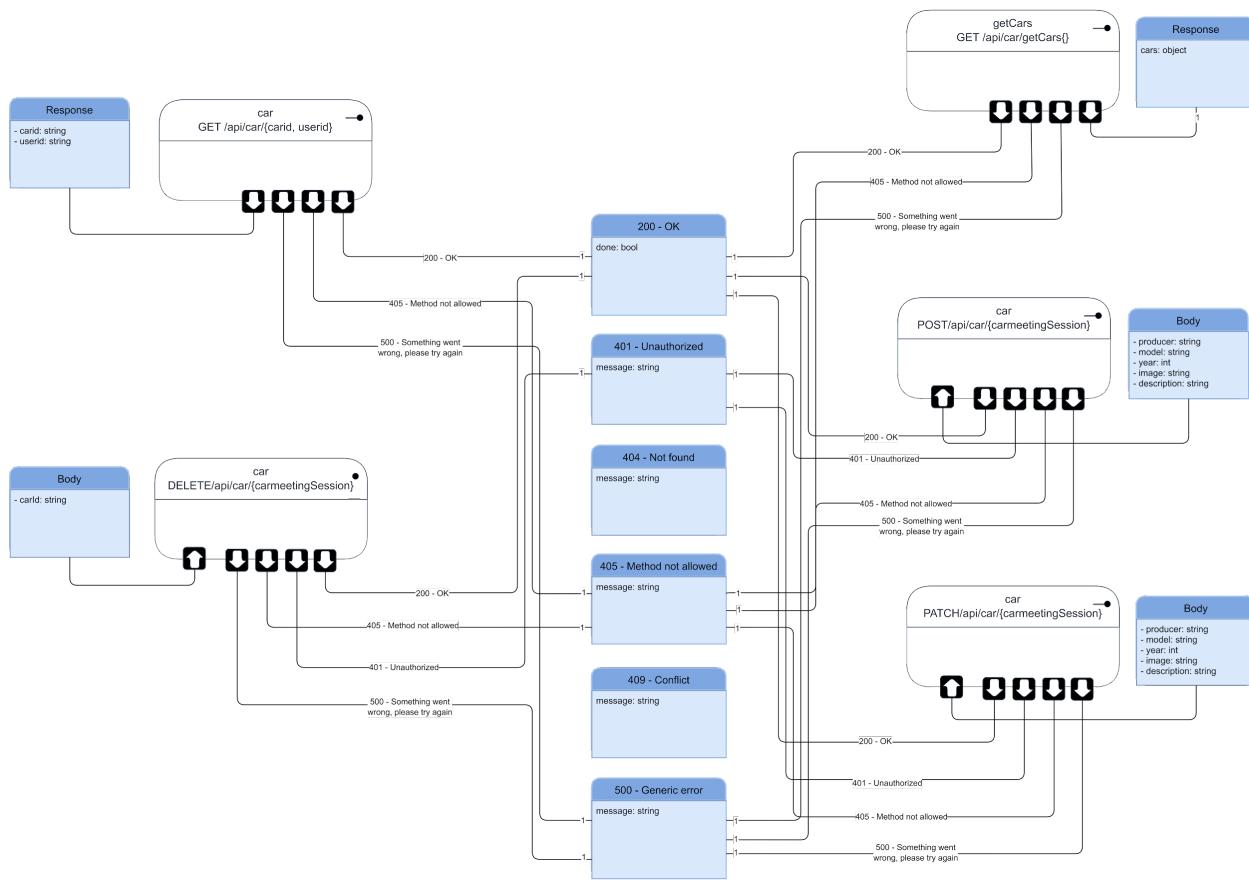
Per la visione di questa immagine ad alta risoluzione si veda l'[allegato CA](#).

Evento



Per la visione di questa immagine ad alta risoluzione si veda l'[allegato CB](#).

Auto



Per la visione di questa immagine ad alta risoluzione si veda l'[allegato CC](#).

2.5 Sviluppo API

Qui di seguito il codice delle principali API utilizzate nel progetto. Il codice completo di tutte le API è consultabile su GitHub.

2.5.1 Signup (Registrazione)

Questa API permette la registrazione di nuovi utenti all'interno del sistema. Dopo che l'utente ha inserito email, username e password, l'API controlla tramite la libreria Yup che i dati inseriti siano validi, in caso affermativo crea l'account dell'utente sul database, invia l'email di conferma e ritorna l'HTTP response 200, mentre in caso contrario ritorna l'HTTP response 500.

```
const BASE_URI = process.env.BASE_URI;

if (!BASE_URI) throw new Error("BASE_URI local variable not set");

async function signupHandler(req, res) {
  const schema = Yup.object().shape({
    email: regSchema.email,
    username: regSchema.username,
    password: regSchema.password,
  });

  if (req.method !== "POST") return res.status(405).send("Method not allowed");

  try {
    await schema.validate(req.body);

    const { username, verifyEmail } = await createUser(req.body);

    const encodedUsername = encodeURIComponent(username);
    const encodedVerifyEmail = encodeURIComponent(verifyEmail);

    const verificationLink = `${BASE_URI}api/user/verifyEmail?username=${encodedUsername}&verifyEmail=${encodedVerifyEmail}`;

    if (process.env.NODE_ENV !== "development") {
      await sendVerifyMail(req.body.email, { username, verificationLink });
    } else console.log("Email verification link: " + verificationLink);

    return res.status(200).send({ done: true });
  } catch (error) {
    if (process.env.NODE_ENV === "development") console.error(error);
    return res.status(500).send(error.message);
  }
}

export default signupHandler;
```

2.5.2 Login (Accesso al sistema)

Questa API utilizza il metodo POST permette all'utente di accedere al suo account. Dopo che l'utente ha inserito email e password, l'API controlla tramite la libreria Yup che i dati inseriti siano validi, in caso affermativo permette all'utente di entrare nel proprio account e ritorna l'HTTP response 200. Nel caso in cui la password non corrisponda allo username inserito verrà ritornata l'HTTP response 401, mentre se l'email associata allo username inserito non è stata verificata verrà ritornata l'HTTP response 409.

```
async function loginHandler(req, res) {
  const schema = Yup.object().shape({
    username: loginSchema.username,
    password: loginSchema.password,
  });

  if (req.method !== "POST") return res.status(405).send("Method not allowed");

  try {
    await schema.validate(req.body);

    const user = await findUser(req.body);

    if (!user || !validatePassword(user, req.body.password)) {
      return res.status(401).send("Invalid username and password combination");
    }

    if (!user.verified) {
      return res.status(409).send("Email not yet verified");
    }

    const session = { username: user.username, userid: user.id };

    await setLoginSession(res, req.body.remember, session);

    return res.status(200).send({ done: true });
  } catch (error) {
    if (process.env.NODE_ENV === "development") console.error(error);
    return res.status(401).send(error.message);
  }
}

export default loginHandler;
```

2.5.3 Logout (Uscita dal sistema)

Questa API utilizza il metodo GET e permette all'utente di uscire dal suo account. Una volta che l'utente ha effettuato il logout viene automaticamente reindirizzato alla pagina home.

```
async function logoutHandler(_req, res) {
    removeTokenCookie(res);
    return res.redirect(307, "/");
}

export default logoutHandler;
```

2.5.4 requestRecoverPassword (Richiesta recupero password)

Questa API utilizza il metodo POST e permette all'utente di richiedere di recuperare la password del proprio account.. Dopo che l'utente ha inserito username e email del suo account, l'API controlla tramite la libreria Yup che i dati inseriti siano validi. Se l'email e l'username sono effettivamente associati ad un account attivo l'API invierà alla mail dell'utente un link che permetterà di essere reindirizzati alla pagina di recupero password e ritornerà l'HTTP response 200.

```
async function requestRecoverPasswordHandler(req, res) {
  const schema = Yup.object().shape({
    email: regSchema.email,
    username: regSchema.username,
  });

  if (req.method !== "POST") return res.status(405).send("Method not allowed");

  try {
    await schema.validate(req.body);

    const user = await findUser(req.body);

    if (!user || user.email !== req.body.email)
      return res.status(401).send("Username or email invalid");

    const { recoverPasswordCode } = await createRecoverPasswordUser(user);

    const recoverPasswordCodeURI = encodeURIComponent(recoverPasswordCode);
    const recoveryLink = `${process.env.BASE_URI}recover/${recoverPasswordCodeURI}`;

    if (process.env.NODE_ENV !== "development") {
      await sendPwdRecoveryMail(req.body.email, {
        username: req.body.username,
        recoveryLink,
      });
    } else console.log("Recover password link: " + recoveryLink);

    return res.status(200).json({ done: true });
  } catch (error) {
    if (process.env.NODE_ENV === "development") console.error(error);
    return res.status(500).send("Error requesting recover password");
  }
}

export default requestRecoverPasswordHandler;
```

2.5.5 recoverPassword (Recupero password)

Questa API utilizza il metodo PUT e permette all'utente di richiedere di recuperare la password del proprio account.. Dopo che l'utente ha cliccato il link che gli era stato precedentemente inviato per email può inserire il suo username e la nuova password che intende associare al suo account; una volta fatto ciò l'API controlla tramite la libreria Yup che i dati inseriti siano validi e in tal caso modifica la password e ritorna l'HTTP response 200. Nel caso in cui l'utente inserisca uno username non esistente verrà ritornata l'HTTP response 404 mentre nel caso di un errore generico verrà ritornata l'HTTP response 500.

```
async function recoverPasswordHandler(req, res) {
  const schema = Yup.object().shape({
    username: Yup.string().max(24, "Username too long! (max: 24)").required(),
    newPassword: regSchema.password,
    recoverPasswordCode: secretCodeSchema,
  });

  if (req.method !== "PUT") return res.status(405).send("Method not allowed");

  try {
    await schema.validate(req.body);

    const user = await findUser(req.body);
    const recoverPasswordUser = await findRecoverPwdUser(req.body);

    if (!user || !recoverPasswordUser)
      return res.status(404).send("User not found");

    if (user.id !== recoverPasswordUser.id)
      return res.status(500).send("Something very bad happened");

    if (recoverPasswordUser.secret === req.body.recoverPasswordCode) {
      await changeUserPassword(user, req.body.newPassword);
      await deleteRecoverPasswordUser(recoverPasswordUser);
      return res.status(200).json({ done: true });
    }

    return res.status(500).send("Error while changing password");
  } catch (error) {
    if (process.env.NODE_ENV === "development") console.error(error);
    return res.status(500).send("Error recovering password");
  }
}

export default recoverPasswordHandler;
```

2.5.6 updatePassword (Modifica password)

Questa API utilizza il metodo PATCH e permette all'utente di modificare la password del proprio account. Dopo che l'utente ha inserito username, password attuale e nuova password, l'API controlla tramite la libreria Yup che i dati inseriti siano validi, in caso affermativo permette all'utente di modificare la password e ritorna l'HTTP response 200. Nel caso in cui l'utente inserisca una password identica a quella già esistente viene ritornata l'HTTP response 400, mentre nel caso in cui l'username o la password attualmente in uso inseriti non siano validi viene ritornata l'HTTP response 401.

```
async function updatePasswordHandler(req, res) {
  const schema = Yup.object().shape({
    username: loginSchema.username,
    password: loginSchema.password,
    newPassword: regSchema.password,
  });

  if (req.method !== "PATCH") return res.status(405).send("Method not allowed");

  try {
    await schema.validate(req.body);

    const user = await findUser(req.body);

    if (!user || !validatePassword(user, req.body.password)) {
      return res.status(401).send("Invalid username and password combination");
    }

    if (req.body.password === req.body.newPassword) {
      return res
        .status(400)
        .send("New password must be different from old password");
    }

    changeUserPassword(user, req.body.newPassword);

    return res.status(200).json({ done: true });
  } catch (error) {
    if (process.env.NODE_ENV === "development") console.error(error);
    return res.status(500).send("Error updating password");
  }
}

export default updatePasswordHandler;
```

2.5.7 verifyEmail (Verifica validità indirizzo email)

Questa API utilizza il metodo GET e permette all'utente di verificare la validità dell'indirizzo email che ha precedentemente associato al suo account. Appena l'utente clicca sul link che gli è stato inviato in precedenza, verrà reindirizzato alla pagina home se il link di verifica è valido. Nel caso in cui il link non sia valido l'utente verrà notificato del problema e sarà ritornata l'HTTP response 400, mentre se un certo indirizzo email è già stato verificato verrà ritornata l'HTTP response 409.

```
async function verifyEmailHandler(req, res) {
  const schema = Yup.object().shape({
    username: loginSchema.username,
    verifyEmail: secretCodeSchema,
  });

  if (req.method !== "GET") return res.status(405).send("Method not allowed");

  try {
    await schema.validate(req.query);

    const validatedUser = req.query;
    const user = await findUser(validatedUser, false);

    if (!user) throw new Error("User not found");
    if (user.verified) return res.status(409).send("user already verified");

    if (user.verifyEmailSecret === req.query.verifyEmail) {
      verifyEmail(user);
      return res.redirect(307, "/login");
    } else {
      return res.status(400).send("Invalid verification link");
    }
  } catch (error) {
    if (process.env.NODE_ENV === "development") console.error(error);
    return res.status(500).send("Error while verifying email");
  }
}

export default verifyEmailHandler;
```

2.5.8 car (Gestione veicolo)

Questa API si occupa di tutte le operazioni legate ai veicoli. Quando il sistema richiede di visualizzare le informazioni relative ad un determinato veicolo l'API utilizza il metodo GET, in caso di una richiesta di aggiunta veicolo viene utilizzato il metodo POST, in caso di una richiesta di modifica viene utilizzato il metodo PATCH, infine, in caso di una richiesta di cancellazione di un veicolo, verrà utilizzato il metodo DELETE (negli ultimi tre casi vengono anche fatti dei controlli per verificare che l'utente sia autenticato prima di autorizzarlo ad effettuare queste modifiche).

```
async function carHandler(req, res) {
  if (
    req.method !== "GET" &&
    req.method !== "POST" &&
    req.method !== "PATCH" &&
    req.method !== "DELETE"
  )
    return res.status(405).send("Method not allowed");

  try {
    //-----GET
    if (req.method === "GET") {
      var car;
      if (req.query.carid) car = await findCarById(req.query.carid);
      else if (req.query.userid) car = await findCarByUserId(req.query.userid);
      return res.status(200).json(car ? car : null);
    }

    const session = await getLoginSession(req);
    if (!session) return res.status(401).json("Unauthorized");
    const userFetched = await findUser(session);
    if (userFetched && userFetched.lastPasswordChange > session.createdAt) {
      removeTokenCookie(res);
      return res.redirect(307, "/login");
    }

    //-----DELETE
    else if (req.method === "DELETE") {
      const car = await findCarById(req.body.id);
      if (car.ownerId !== userFetched.id)
        return res.status(401).send("Unauthorized");
      await deleteCar(req.body.id);
      return res.status(200).json({ done: true });
    }
  }
}
```

```
// YUP VALIDATION

const schema = Yup.object().shape({
  producer: Yup.string().required(),
  model: Yup.string().required(),
  year: Yup.number().required(),
  description: carSchema.description,
});

await schema.validate(req.body);
const carTypes = require("/data/cars.json");
const nationality = carTypes.cars.find(
  (car) => car.producer === req.body.producer
).nationality;

//-----POST
if (req.method === "POST") {
  await createCar({
    ...req.body,
    nationality: nationality,
    userId: userFetched.id,
  });
  return res.status(200).json({ done: true });
}

//-----PATCH
else if (req.method === "PATCH") {
  const car = await findCarById(req.body);
  if (car.ownerId !== userSession.userid)
    return res.status(401).send("Unauthorized");
  await updateCar({
    ...req.body,
    nationality: nationality,
    userId: userFetched.id,
  });
  return res.status(200).json({ done: true });
}
} catch (error) {
  if (process.env.NODE_ENV === "development") console.error(error);
  return res.status(500).send("Something went wrong, please try again");
}

export default carHandler;
```

2.5.9 user (Gestione sessione utente)

Questa API utilizza il metodo GET e si occupa di ottenere l'account associato al cookie sessione dell'utente. Se l'operazione va a buon fine viene ritornata l'HTTP response 200 mentre, nel caso in cui il cookie non fosse valido, l'utente viene reindirizzato alla pagina di login e viene ritornata l'HTTP response 307.

```
async function userHandler(req, res) {
  try {
    if (req.method === "GET") {
      const session = await getLoginSession(req);

      if (!session) return res.status(200).json(null);

      const userFetched = await findUser(session);

      if (userFetched && userFetched.lastPasswordChange > session.createdAt) {
        removeTokenCookie(res);
        return res.redirect(307, "/login");
      }

      const userSession = (session && userFetched) ?? null;
      session.email = userFetched.email;

      return res.status(200).json(userSession ? session : null);
    } else return res.status(405).send("Method not allowed");
  } catch (error) {
    if (process.env.NODE_ENV === "development") console.error(error);
    return res
      .status(500)
      .send("Authentication token is invalid, please log in");
  }
}

export default userHandler;
```

2.5.10 updateUser (Modifica informazioni utente)

Questa API utilizza il metodo PATCH e permette all'utente di modificare l'email, lo username o l'immagine di profilo del proprio account. Dopo che l'utente ha inserito i dati richiesti, l'API controlla tramite la libreria Yup che i dati inseriti siano validi, in caso affermativo permette all'utente di effettuare le modifiche e ritorna l'HTTP response 200. Nel caso in cui il cookie non fosse valido, l'utente viene reindirizzato alla pagina di login e viene ritornata l'HTTP response 307, mentre nel caso di un errore generico viene ritornata l'HTTP response 401.

```
async function verifyEmailHandler(req, res) {
  const schema = Yup.object().shape({
    username: loginSchema.username,
    email: loginSchema.email,
  });

  if (req.method !== "PATCH") return res.status(405).send("Method not allowed");

  try {
    await schema.validate(req.body);

    const session = await getLoginSession(req);
    if (!session) return res.status(200).json({ done: true });
    const userFetched = await findUser(session);
    if (userFetched && userFetched.lastPasswordChange > session.createdAt) {
      removeTokenCookie(res);
      return res.redirect(307, "/login");
    }

    const { username, verifyEmail } = await updateUser(req.body);

    if (!verifyEmail) return res.status(200).json({ done: true });

    const verificationLink = `${BASE_URI}api/user/verifyEmail?username=${encodedUsername}&verifyEmail=${encodedVerifyEmail}`;

    if (process.env.NODE_ENV !== "development") {
      await sendVerifyMail(req.body.email, { username, verificationLink });
    } else console.log("Email verification link: " + verificationLink);
    removeTokenCookie(res);
    return res.redirect(307, "/login");
  } catch (error) {
    if (process.env.NODE_ENV === "development") console.error(error);
    return res.status(500).send("Error updating user");
  }
}

export default verifyEmailHandler;
```

2.5.11 event (Gestione eventi)

Questa API si occupa di tutte le operazioni legate agli eventi. Quando il sistema richiede di visualizzare le informazioni relative ad un determinato evento l'API utilizza il metodo GET, in caso di una richiesta di creazione di un nuovo evento viene utilizzato il metodo POST, infine, in caso di una richiesta di modifica di un evento viene utilizzato il metodo PATCH (negli ultimi due casi vengono anche fatti dei controlli per verificare che l'utente sia autenticato prima di autorizzarlo ad effettuare queste modifiche).

```
async function eventHandler(req, res) {
  if (
    req.method !== "GET" &&
    req.method !== "POST" &&
    req.method !== "PATCH" &&
    req.method !== "DELETE"
  )
    return res.status(405).send("Method not allowed");

  try {
    //-----GET

    if (req.method === "GET") {
      let event;
      if (!req.query.eventid && !req.query.userid) {
        event = await findProgrammedEvents();
      } else if (req.query.eventid) {
        event = await findEventById(req.query.eventid);
      } else if (req.query.userid) {
        event = await findEventByOwnerId(req.query.userid);
      }
      if (event) return res.status(200).json(event);
      return res.status(404).send("Event not found");
    }

    //YUP VALIDATION

    const schema = Yup.object().shape({
      eventName: eventSchema.name,
      startDate: eventSchema.date,
      finishDate: eventSchema.date,
      location: eventSchema.location,
      maxParticipants: eventSchema.maxParticipants,
      cost: eventSchema.cost,
      description: eventSchema.description,
    });

    await schema.validate(req.body);
  }
}
```

```
// CHECK SESSION

const session = await getLoginSession(req);
if (!session) return res.status(401).send("Unauthorized");
const userFetched = await findUser(session);
if (userFetched && userFetched.lastPasswordChange > session.createdAt) {
    removeTokenCookie(res);
    return res.redirect(307, "/login");
}

//-----POST
if (req.method === "POST") {
    //TODO: allowed cars filter
    //TODO: gpx file handler
    //TODO: image handler
    await createEvent({
        ...req.body,
        userId: userFetched.id,
        eventState: "PROGRAMMATO",
    });
    return res.status(200).json({ done: true });
}

//-----PATCH
else if (req.method === "PATCH") {
    const event = await findEventById(req.body.eventid);
    if (event.ownerId !== userFetched.id)
        return res.status(401).send("Unauthorized");
    await updateEvent(req.body);
    return res.status(200).json({ done: true });
}
} catch (error) {
    if (process.env.NODE_ENV === "development") console.error(error);
    return res.status(500).send("Something went wrong, please try again");
}

export default eventHandler;
```

2.5.12 subscribeEvent (Iscrizione ad un evento)

Questa API utilizza il metodo POST e permette ad un utente di iscriversi ad un certo evento. Se l'operazione va a buon fine viene ritornata l'HTTP response 200, mentre, se l'utente non è loggato, quest'ultimo viene reindirizzato alla pagina di login e viene ritornata l'HTTP response 307.

```
✓ async function subscribeEventHandeler(req, res) {
  if (req.method !== "POST") return res.status(405).send("Method not allowed");

  try {
    const session = await getLoginSession(req);
    if (!session) return res.status(401).send("Unauthorized");
    const userFetched = await findUser(session);
    if (userFetched && userFetched.lastPasswordChange > session.createdAt) {
      removeTokenCookie(res);
      return res.redirect(307, "/login");
    }

    //TODO: check if user has the allowed car

    if (await subscribeEvent({ ...req.body, userid: userFetched.id })) {
      await subscribeUser({
        userid: userFetched.id,
        eventid: req.body.eventid,
        subscribe: req.body.subscribe,
      });

      return res.status(200).json({ done: true });
    }
    return res.status(500).send();
  } catch (error) {
    if (process.env.NODE_ENV === "development") console.error(error);
    return res.status(500).send("Something went wrong, please try again");
  }
}

export default subscribeEventHandeler;
```

2.5.13 getUsername (ottiene nome utente)

Questa API utilizza il metodo GET e si occupa di ottenere l'username di un account dal suo userid. Nel caso in cui l'operazione vada a buon fine viene ritornata l'HTTP response 200, mentre in caso contrario, viene ritornata l'HTTP response 500.

```
async function userHandler(req, res) {
  try {
    if (req.method === "GET") {
      if (!req.query.userid) return res.status(200).json({ done: true });

      const userFetched = await findUserById(req.query.userid);

      return res
        .status(200)
        .json(
          userFetched
            ? { id: req.query.userid, username: userFetched.username }
            : null
        );
    } else return res.status(405).send("Method not allowed");
  } catch (error) {
    if (process.env.NODE_ENV === "development") console.error(error);
    return res.status(500).send("Something went wrong, please try again");
  }
}

export default userHandler;
```

2.5.14 getCars (ottiene lista veicoli)

Questa API utilizza il metodo GET e si occupa di ottenere la lista dei veicoli che è possibile selezionare nel momento in cui l'utente cerca di aggiungere una nuova auto al garage. Nel caso in cui l'operazione vada a buon fine viene ritornata l'HTTP response 200, mentre in caso contrario, viene ritornata l'HTTP response 500.

```
async function subscribeEventHandeler(req, res) {
  if (req.method !== "GET") return res.status(405).send("Method not allowed");

  try {
    const cars = require("/data/cars.json");

    return res.status(200).json(cars.cars);
  } catch (error) {
    if (process.env.NODE_ENV === "development") console.error(error);
    return res.status(500).send("Something went wrong, please try again");
  }
}

export default subscribeEventHandeler;
```

2.5.15 myAgenda (ottiene lista eventi)

Questa API utilizza il metodo GET e si occupa di ottenere la lista degli eventi a cui un certo utente è iscritto (compresi quelli creati dall'utente stesso). Nel caso in cui l'operazione vada a buon fine viene ritornata l'HTTP response 200 e la lista degli eventi, mentre in caso contrario, viene ritornata l'HTTP response 500.

```
async function subscribeEventHandeler(req, res) {
  if (req.method !== "GET") return res.status(405).send("Method not allowed");
  if (!req.query.username) return res.status(400).send("Missing information");

  try {
    const userFetched = await findUser(req.query);

    if (!userFetched) return res.status(404).send("User not found");

    const userEvents = userFetched.subscribedEventIds;
    let events = [];

    for (let i = 0; i < userEvents.length; i++) {
      const eventFetched = await findEventById(userEvents[i]);
      if (!eventFetched)
        return res.status(500).send("Something went wrong, please try again");
      events.push(eventFetched);
    }

    return res.status(200).json(events);
  } catch (error) {
    if (process.env.NODE_ENV === "development") console.error(error);
    return res.status(500).send("Something went wrong, please try again");
  }
}

export default subscribeEventHandeler;
```

3. API DOCUMENTATION

Le API locali fornite dall'applicazione CarMeeTiN descritte nella sezione precedente sono state documentate utilizzando il pacchetto NPM [next-swagger-doc](#). La documentazione relativa alle API è quindi direttamente disponibile a chiunque disponga del codice sorgente. Per poter generare l'endpoint dedicato alla presentazione delle API è stato usato Swagger UI, attraverso cui è stato possibile creare una pagina web dalle definizioni delle specifiche OpenAPI.

Di seguito è mostrata la pagina web relativa alla documentazione che presenta le API per la gestione dei dati della applicazione.

Nello sviluppo delle API vengono usate le funzioni GET, POST, PATCH, PUT e DELETE.

Nello specifico, la funzione GET viene utilizzata per recuperare i dati da un server, la funzione POST per creare una nuova risorsa sul server utilizzando informazioni fornite dall'utente in input, la funzione PATCH per aggiornare una risorsa sul server, PUT per sostituire un'intera risorsa, mentre la funzione DELETE viene utilizzata per eliminare una risorsa dal server che non è più necessaria.

In generale, queste vengono utilizzate in modo tale da consentire a diverse applicazioni e servizi di interagire tra loro, consentendo quindi la creazione di sistemi più potenti e flessibili permettendo l'utilizzo di dati e di funzionalità fornite da altri sistemi.

L'endpoint da invocare per raggiungere tale documentazione è il seguente:

<http://carmeetin.vercel.app/api-docs>

N.B: si consiglia di visualizzare la pagina in modalità light e non dark, tramite il pulsante apposito in alto a destra, per evitare alcuni bug visivi.

Di seguito riportiamo l'immagine della pagina web relativa alla documentazione:

The screenshot shows the CarMeeTiN API Documentation page. It is organized into three main sections: **Car**, **Event**, and **User**. Each section contains a list of API endpoints with their methods, URLs, and descriptions. The sections are collapsible, indicated by a '^' icon at the top right of each.

Car APIs relevant to car objects

- GET** /api/car/getCars Gets all the combinations of cars makes and models
- GET** /api/car/{carid, userid} Returns the requested car
- POST** /api/car Creates a new car
- PATCH** /api/car Update an existing car
- DELETE** /api/car Delete an existing car

event APIs relevant to event objects

- GET** /api/event/{eventid, userid} Returns the requested event
- POST** /api/event/ Creates a new event
- PATCH** /api/event/ Updates an existing event
- GET** /api/event/myAgenda/{userid} Gets the agenda of the user
- POST** /api/event/subscribeEvent Subscribes or unsubscribes a user to an event

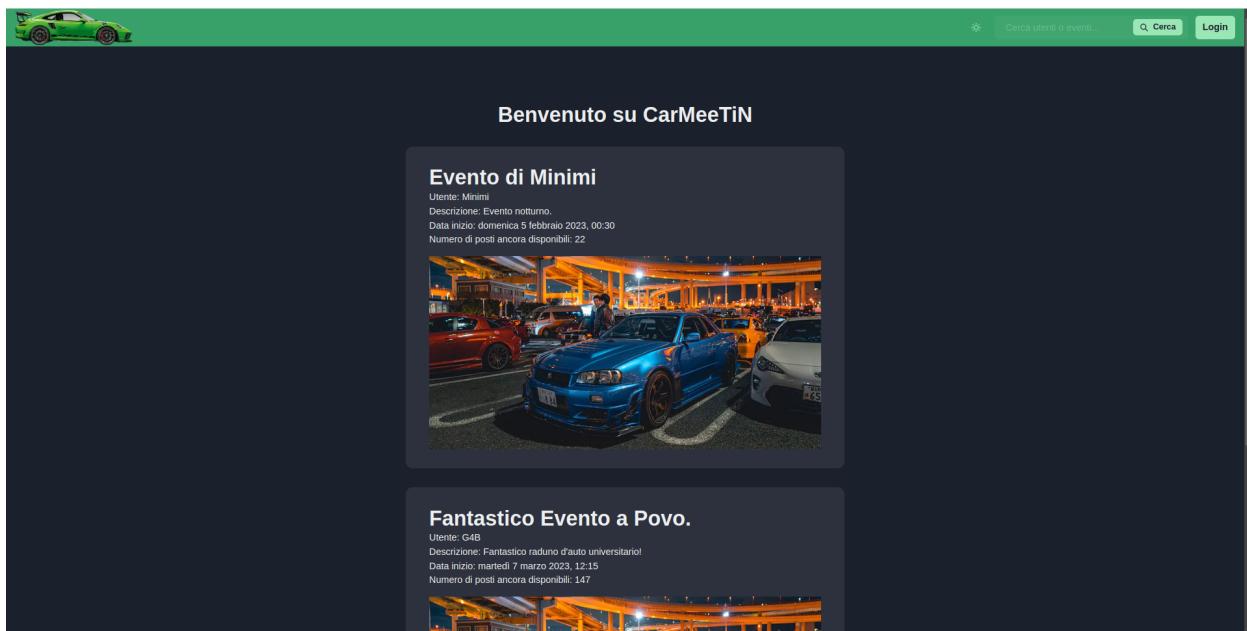
USER APIs relevant to user access

- GET** /api/user/getUsername Gets username from userid
- GET** /api/user Gets user from session cookie
- POST** /api/user/login Lets user log in
- GET** /api/user/logout Logs out user
- PUT** /api/user/recoverPassword Recovers the password of the user
- POST** /api/user/requestRecoverPassword Sends an email to the user with a secret code to recover the password
- POST** /api/user/signup Creates a new user account
- PATCH** /api/user/updatePassword Updates the password of the user
- PATCH** /api/user/updateUser Updates the user account
- GET** /api/user/verifyEmail/{username, verifyEmail} Verify the email of the user

4. FRONTEND IMPLEMENTATION

Questa sezione del documento fornisce le funzionalità di visualizzazione, inserimento e cancellazione dei dati forniti nell'applicazione CarMeeTiN all'utente finale attraverso l'utilizzo delle varie API sviluppate e precedentemente descritte.

4.1 Pagina Home

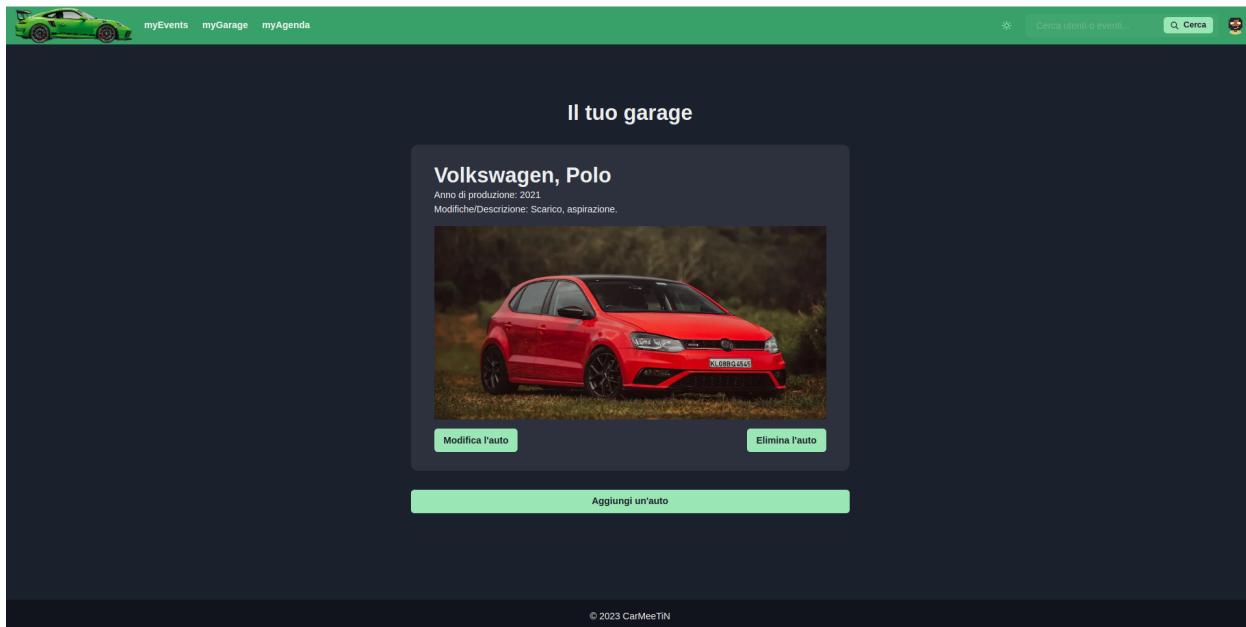


La pagina Home è la prima pagina che viene visualizzata accedendo al sito web, da essa si può visualizzare ed entrare nelle pagine degli eventi programmati.

Se non si è al momento autenticati, cliccando in alto a destra è possibile accedere alla pagina di login ([sezione 4.6](#)), in caso contrario è possibile accedere a:

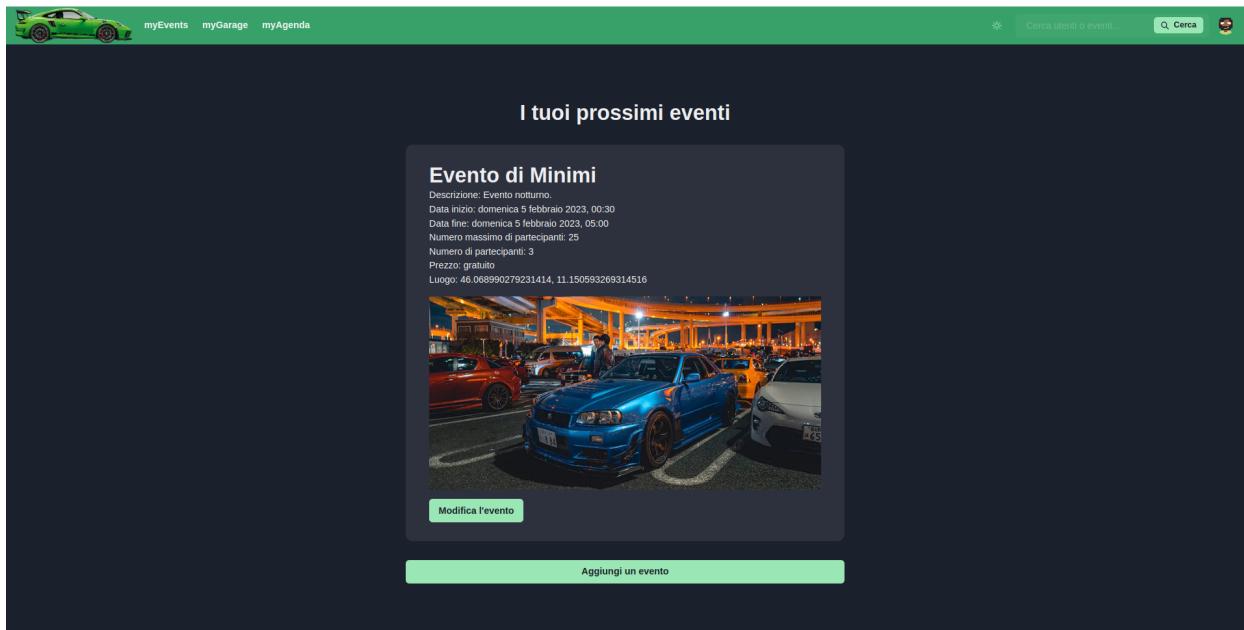
- myEvents ([sezione 4.3](#))
- myGarage ([sezione 4.2](#))
- myAgenda ([sezione 4.4](#))
- Aggiorna Account ([sezione 4.10](#))
- Aggiorna Password ([sezione 4.9](#))
- Logout

4.2 Pagina myGarage



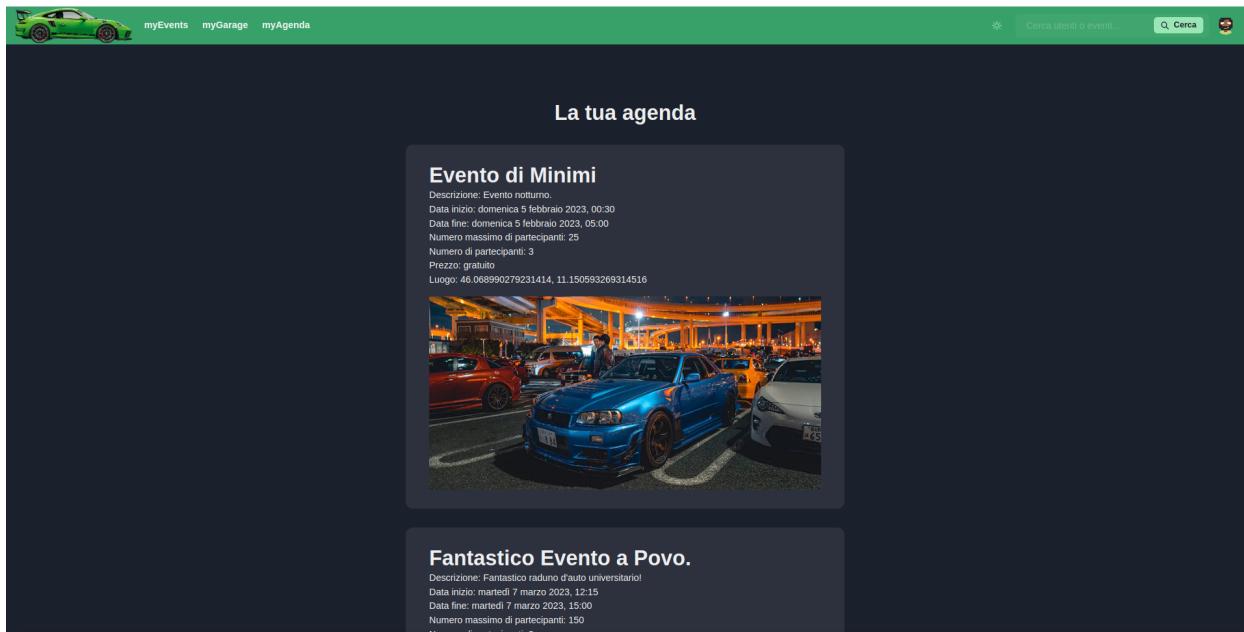
Dalla pagina myGarage è possibile aggiungere ([sezione 4.11](#)), modificare ([sezione 4.17](#)) o eliminare delle auto dal proprio garage o visualizzare quelle precedentemente aggiunte.

4.3 Pagina myEvents



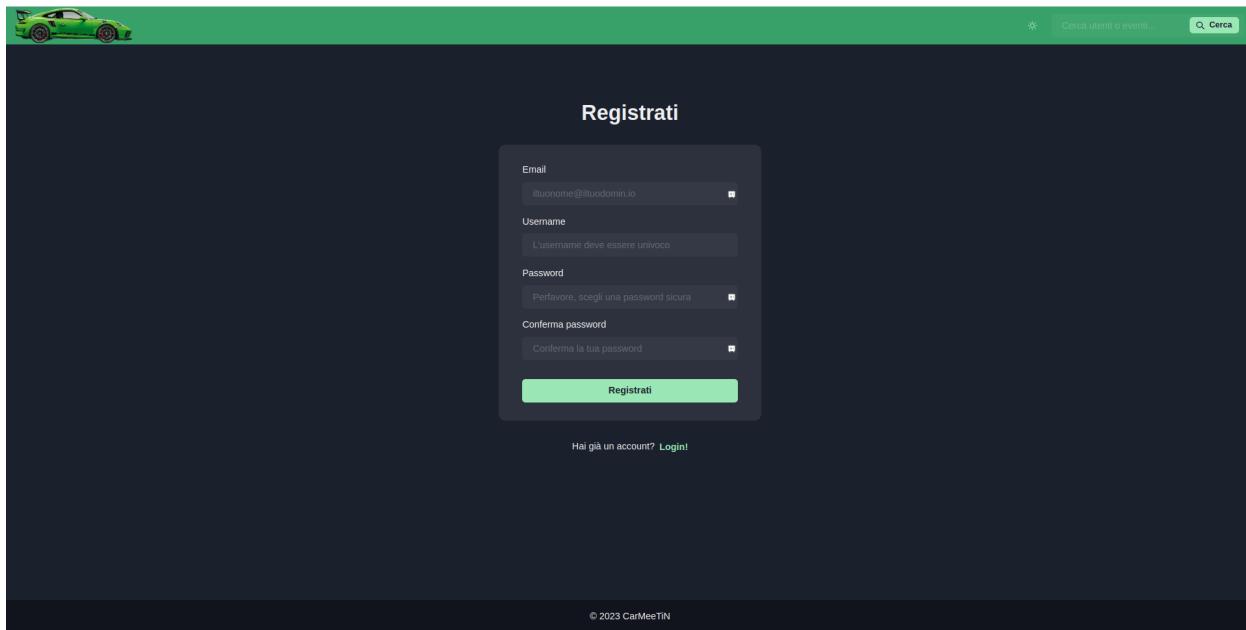
Dalla pagina myEvents è possibile creare ([sezione 4.12](#)) o modificare ([sezione 4.16](#)) degli eventi oppure visualizzare quelli precedentemente creati accedendo alla rispettiva pagina cliccandoci sopra ([sezione 4.13](#)).

4.4 Pagina myAgenda



Dalla pagina myAgenda è possibile visualizzare gli eventi a cui si è iscritti e accedere alle loro rispettive pagine cliccandoci sopra ([sezione 4.13](#)).

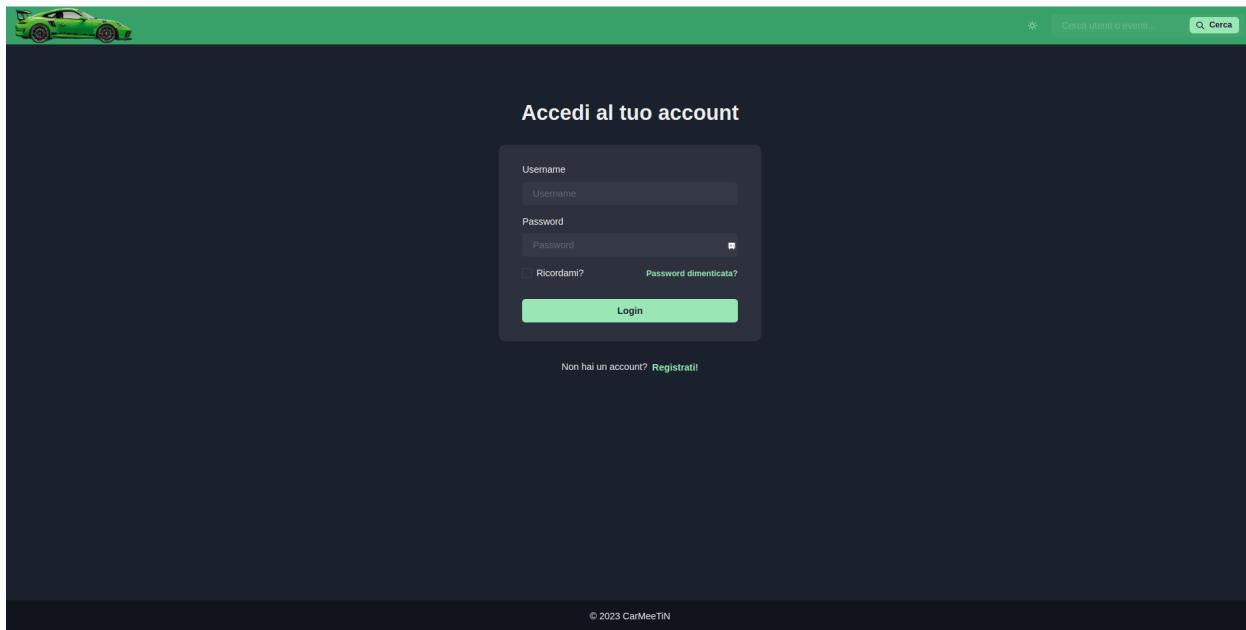
4.5 Pagina Registrazione



Dalla pagina registrazione è possibile registrarsi al sistema inserendo email, username e due volte la password.

È anche disponibile un pulsante “*Login!*” per accedere alla pagina di login ([sezione 4.6](#)) in caso si disponga già di un account.

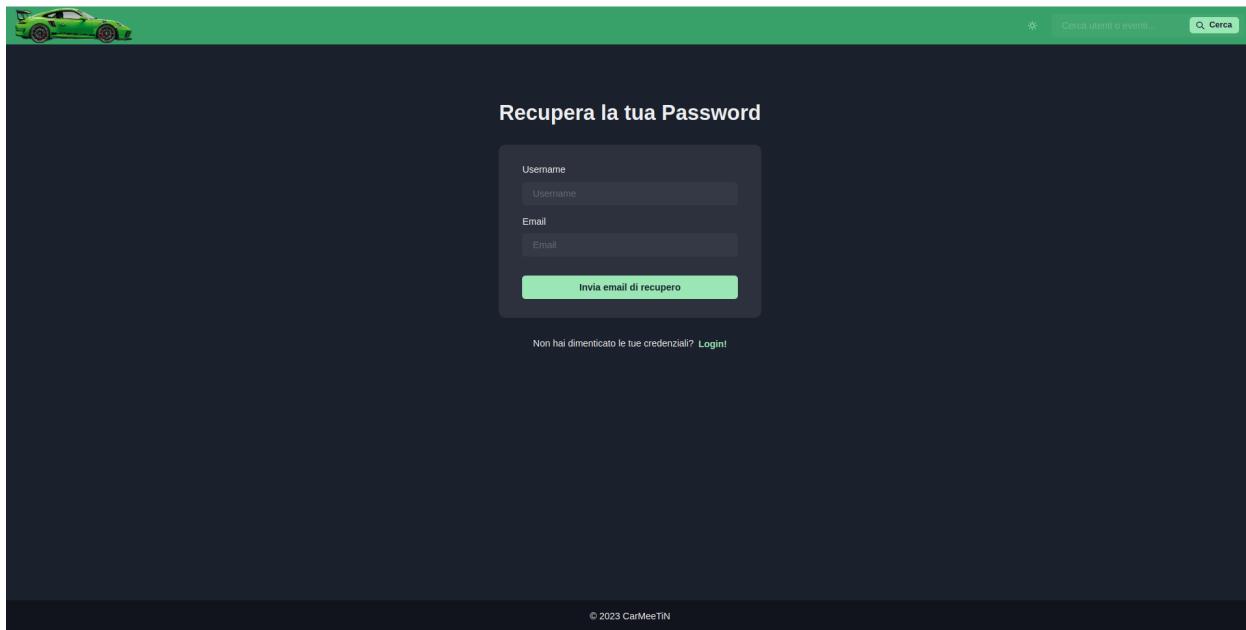
4.6 Pagina Login



Dalla pagina login è possibile autenticarsi al sistema inserendo username e password.

Sono anche disponibili i pulsanti “[Registrati!](#)” per accedere alla pagina di registrazione ([sezione 4.5](#)) in caso non si disponga ancora di un account, e “[Password dimenticata?](#)” per accedere alla pagina di richiesta reset password ([sezione 4.7](#)) in caso si abbia smarrito la password.

4.7 Pagina Richiesta Reset Password



Dalla pagina richiesta reset password è possibile, in caso si abbia dimenticato la password, richiedere l'invio di una email al proprio indirizzo di posta elettronica contenente un link a una pagina per cambiare la password ([sezione 4.8](#)).

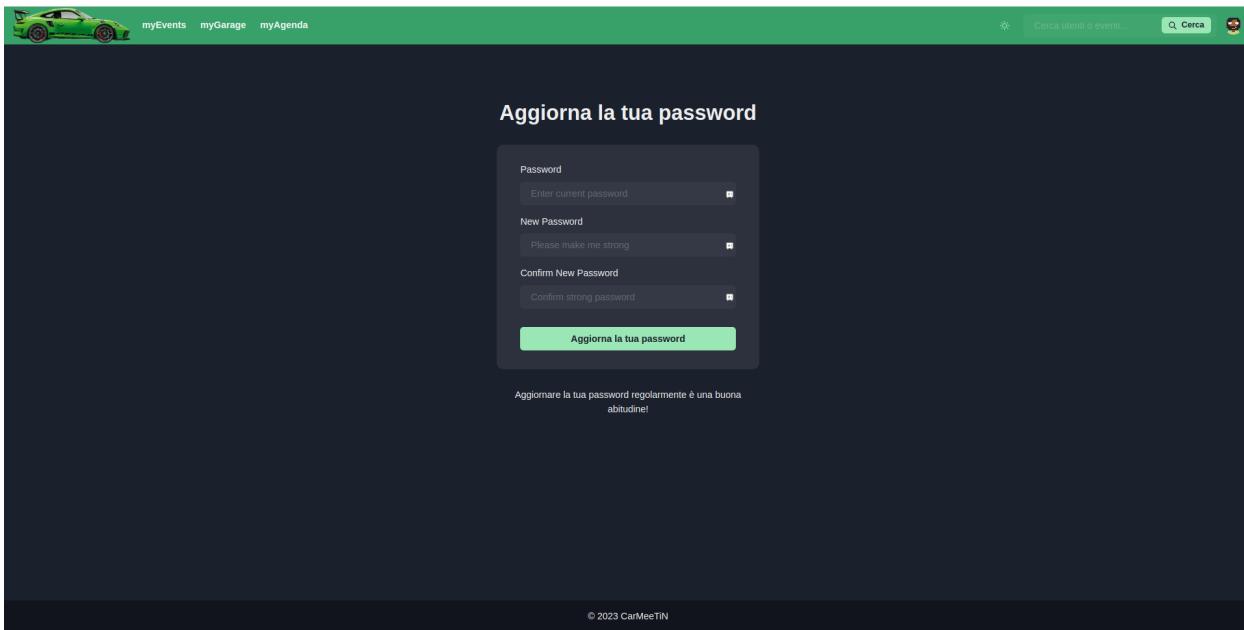
È anche disponibile un pulsante “*Login!*” per accedere alla pagina di login ([sezione 4.6](#)) in caso non si abbia dimenticato la password.

4.8 Pagina Reset Password



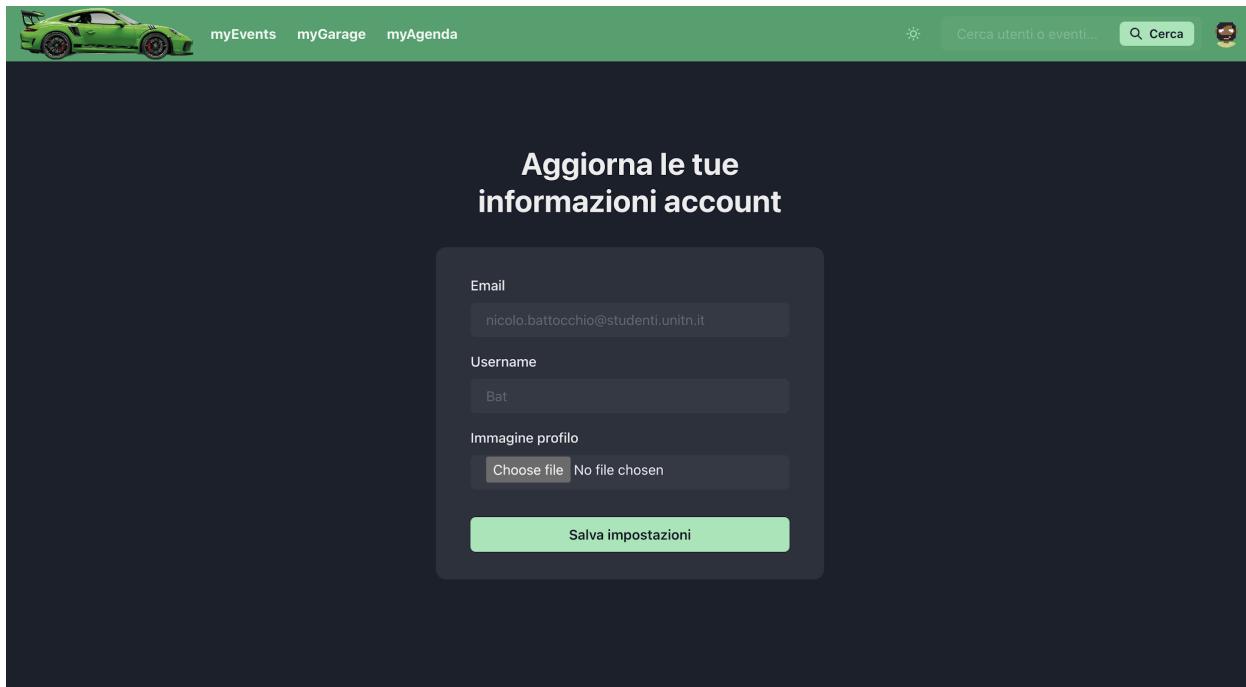
Dalla pagina reset password, accessibile solo tramite un link che viene inviato via email, è possibile cambiare la password del proprio account inserendo la nuova password due volte.

4.9 Pagina Aggiornamento Password



Dalla pagina aggiornamento password è possibile cambiare la propria password inserendo quella vecchia e successivamente due volte quella nuova.

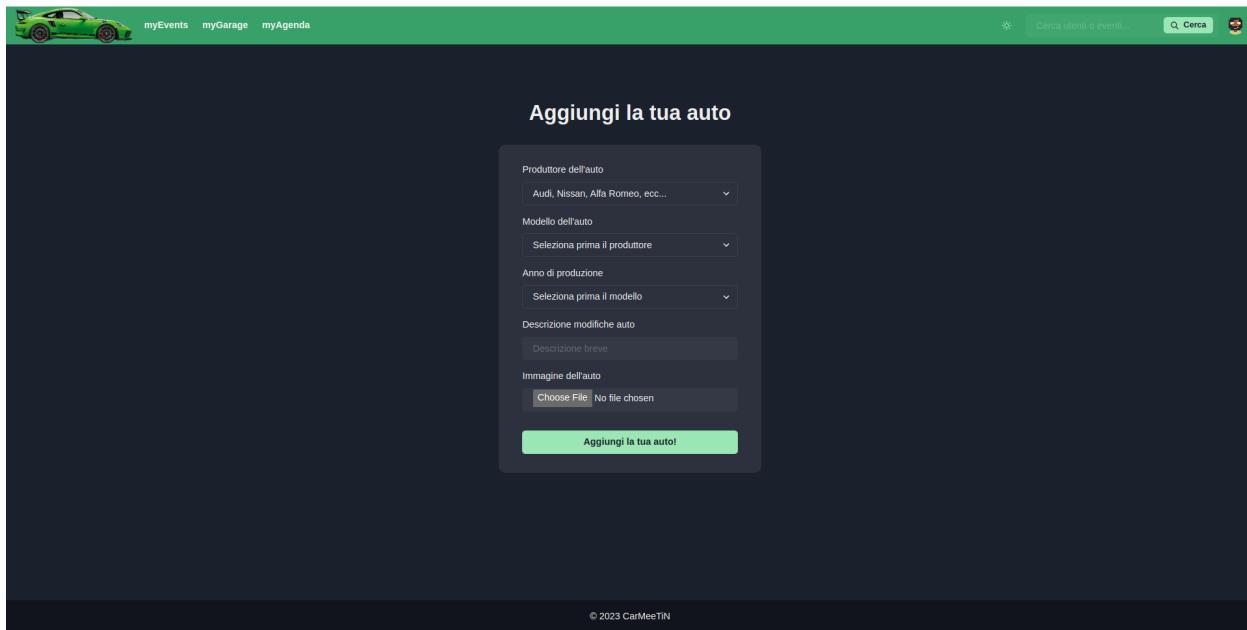
4.10 Pagina Aggiornamento dati dell'utente



Dalla pagina aggiornamento dati dell'utente è possibile cambiare i seguenti dati del proprio account:

- Email
- Username
- Immagine profilo

4.11 Pagina Aggiunta di un'auto



Dalla pagina aggiunta di un'auto è possibile aggiungere un'auto al proprio garage inserendo i seguenti dati:

- Produttore dell'auto
- Modello dell'auto
- Anno di produzione
- Descrizione modifiche auto
- Immagine dell'auto

4.12 Pagina Creazione di un evento

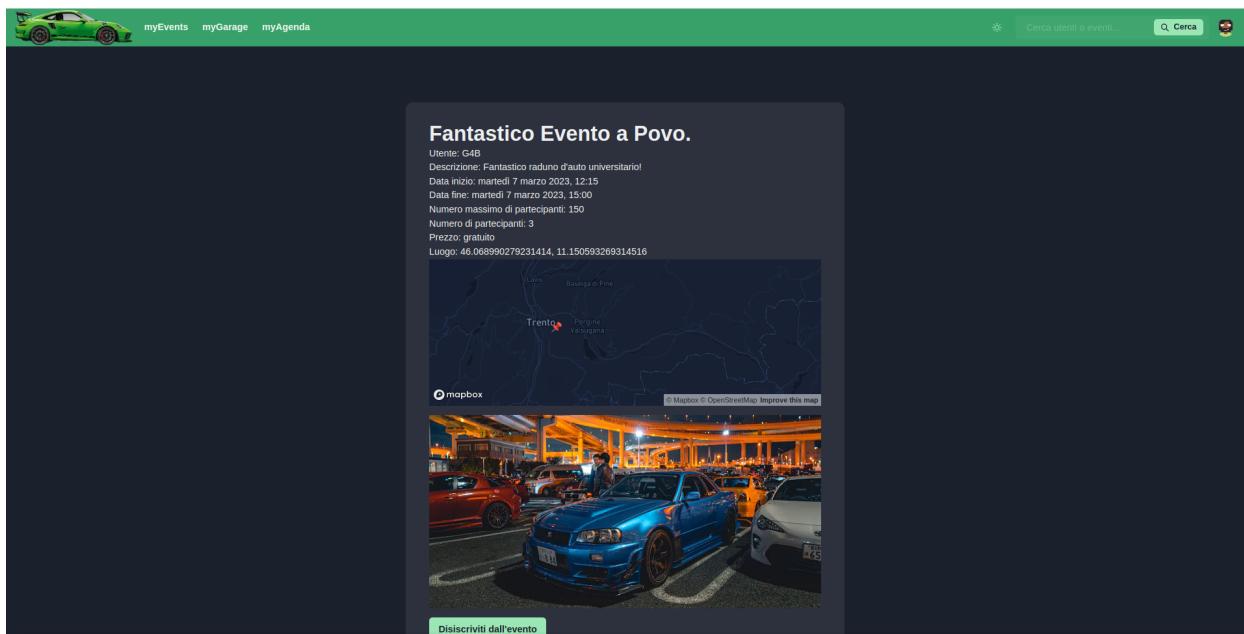
The screenshot shows a dark-themed web application window titled "Crea Evento". Inside, there are several input fields and controls:

- Nome evento:** A text input field with placeholder text "Nome evento".
- Data inizio:** A date input field with placeholder text "mm/dd/yyyy, --:--".
- Data fine:** A date input field with placeholder text "mm/dd/yyyy, --:--".
- Breve descrizione:** A text input field with placeholder text "Descrizione evento".
- Macchine ammesse:** A dropdown menu with placeholder text "Lista...".
- Massimo Partecipanti:** An input field containing the number "10".
- Costo partecipazione:** An input field containing "0,00 €".
- Immagine evento:** A file upload input field with placeholder text "Choose File" and "No file chosen".
- Selezione il luogo:** A map showing the area around Trento, Italy, with a red marker indicating the location.
- mapbox:** A small logo for the map provider.
- Crea Evento!**: A large green button at the bottom right of the form.

Dalla pagina creazione di un evento è possibile creare un evento inserendo i seguenti dati:

- Nome evento
- Data inizio
- Data fine
- Breve descrizione
- Macchine ammesse
- Massimo Partecipanti
- Costo partecipazione
- Immagine evento
- Seleziona il luogo

4.13 Pagina Evento



Dalla pagina di un evento è possibile visualizzare le informazioni relative a tale evento e iscriversi o disiscriversi da esso.

4.14 Pagina Ricerca Utente/Evento

Al momento questa pagina non è stata implementata.

Dalla pagina ricerca utente/evento è possibile ricercare un utente tramite il suo username oppure un evento tramite i dati: nome, luogo, data, caratteristiche dei veicoli che possono partecipare e stato dell'evento.

4.15 Pagina Informativa privacy

Al momento questa pagina non è stata implementata.

Dalla pagina informativa privacy è possibile leggere l'informativa sulla privacy e la policy sui cookie, tramite un avviso, e l'utente potrà in caso accettare le condizioni proposte.

4.16 Pagina Modifica di un evento

Al momento questa pagina non è stata implementata.

Dalla pagina modifica di un evento è possibile modificare i dati scelti in fase di creazione dell'evento stesso.

4.17 Pagina Modifica di un'auto

Al momento questa pagina non è stata implementata.

Dalla pagina modifica di un'auto è possibile modificare i dati scelti in fase di aggiunta dell'auto stessa.

5. GITHUB REPOSITORY AND DEPLOYMENT INFO

Il progetto CarMeeTiN è disponibile al seguente link:

<https://github.com/UNITN-IdS-2022-T11>

Esso è suddiviso in due repository: "Deliverables", "Documents" e "CodeBase".

- "Deliverables" contiene i PDF di tutti i derivable;
- "Documents" contiene gli allegati delle immagini in vettoriale;
- "CodeBase" contiene tutto il codice relativo al FrontEnd ed al BackEnd.

Gli account dei membri del gruppo sono:

- Battocchio Nicolò: <https://github.com/nicolobattocchio>
- Bocchi Gabriele: <https://github.com/GabrieleBocchi>
- Miazzo Luigi: <https://github.com/LuigiMiazzo17>

Il deployment del progetto è stato effettuato con la piattaforma [Vercel](#). Di seguito il link per visualizzare la web app:

<https://carmeetin.vercel.app>

N.B: visto che la verifica dell'indirizzo di posta elettronica sarebbe gestita da un servizio esterno, che al momento non è attivo (anche se le funzionalità sono state testate e sono state riscontrate funzionanti), all'interno del Database sono già presenti 3 utenti attivi per effettuare una prova del sito:

- Username: "*Minimi*", pwd: "aaaAAA12!";
- Username: "*Bat*", pwd: "bbbBBB12!";
- Username: "*G4B*", pwd: "cccCCC12!".

Si noti inoltre che, per la stessa ragione, le funzionalità di cambio dell'indirizzo email e di recupero password, non sono al momento abilitate, nonostante, eseguendo il codice in locale (tramite dei "console.log" dei link che sarebbero stati inviati all'utente) si sono dimostrate funzionanti.

6. TESTING

Per eseguire il testing abbiamo utilizzato il pacchetto [JEST](#) insieme a [node-mocks-http](#) ed è stata definita nel nostro progetto la cartella “tests” al cui interno sono presenti numerosi file *.test.js che andranno a definire dei test cases per le API utilizzate nella nostra web app. La nomenclatura scelta di questi è stata la seguente: “api.ROUTE.TO.API.test.js”, quindi, ad esempio la route “HOST/api/car/getCars” si converte in “api.car.getCars.test.js”.

Sono stati creati numerosi casi di test per verificare che le API funzionino correttamente.

```
▽ __test__  
  JS api.car.getCars.test.js  
  JS api.car.test.js  
  JS api.event.test.js  
  JS api.user.getUsername.test.js  
  JS api.user.login.test.js  
  JS api.user.signup.test.js
```

Queste precedentemente elencate sono le funzioni che testano le api. Tutte le altre sono state testate in fase di development ma non di testing.

6.1 Test API

Di seguito sono mostrati i test effettuati alle API precedentemente elencate:

6.1.1 test /api/car/getCars

```
PASS __test__/api.car.getCars.test.js  
/api/car/getCars test  
  ✓ Should return the list (2 ms)  
  ✓ Should return 405, Method not allowed
```

6.1.2 test /api/car

```
PASS __test__/api.car.test.js
/api/car test
✓ Should return a existing car (160 ms)
✓ Should return a car of a user (15 ms)
✓ Should return error 400, bad request (1 ms)
✓ Should create a car (57 ms)
✓ Should return 400, Bad Request (28 ms)
✓ Should delete the newly created car (67 ms)
✓ Should return 401, Unauthorized
✓ Should return 405, Method not allowed (1 ms)
```

6.1.3 test /api/event

```
PASS __test__/api.event.test.js
/api/event GET method test
✓ Should return a existing event (156 ms)
✓ Should return all programmed event (15 ms)
✓ Should return error 404, event not found (15 ms)
✓ Should return events of a user (15 ms)
✓ Should return 405, Method not allowed
```

6.1.4 test /api/user/getUsername

```
PASS __test__/api.user.getUsername.test.js
/api/user/getUsername test
✓ Should get the correct username (239 ms)
✓ Should return 404, User Not Found (17 ms)
✓ Should return 405, Method not allowed
```

6.1.5 test /api/user/login

```
PASS __test__/api.user.login.test.js
/api/user/login test
  ✓ Should login the user and return the user session (222 ms)
  ✓ Should return 401, Invalid username and password combination (13 ms)
  ✓ Should return 405, Method not allowed (1 ms)
```

6.1.6 test /api/user/signup

```
PASS __test__/api.user.signup.test.js
/api/user/signup test
  ✓ Should create a new user (246 ms)
  ✓ Should return 500, Username not valid (20 ms)
  ✓ Should return 400, Bad Request (1 ms)
  ✓ Should return 405, Method not allowed (1 ms)
```

6.2 Code coverage, test suites e tests

Di seguito si potrà trovare il code coverage generato da JEST, insieme al numero di API testate (test suites) e il numero di test (tests):

File	%Stmts	%Branch	%Funcs	%Lines	Uncovered Line #s
All files	60.91	56.88	56.84	61.5	
lib	52.41	44.11	48.71	52.77	
auth-cookies.js	65	40	66.66	61.11	19,30-39,42-43
auth.js	94.11	50	100	93.75	24
dbCar.js	67.74	100	80	67.74	53-70
dbConnect.js	90.9	57.14	100	90.47	9,39
dbEvent.js	25.49	0	46.15	27.65	8-44,69-113
dbUser.js	33.33	50	34.78	32.85	13,35-46,61-64,76-128
email.js	32	60	0	33.33	11-62
yupSchemas.js	83.87	100	37.5	92	103-108
lib/models	95	100	75	100	
car.js	100	100	100	100	
event.js	100	100	100	100	
recoverPwd.js	83.33	100	0	100	
user.js	100	100	100	100	
pages/api/car	75	68	100	75	
getCars.js	70	50	100	71.42	26-27
index.js	76	69.56	100	75.55	183-184,191,227-240
pages/api/event	50	59.09	100	50	
index.js	50	59.09	100	50	179-224
pages/api/user	79.1	45	100	85.45	
getUsername.js	73.33	57.14	100	81.81	48-49
login.js	81.81	66.66	100	84.21	56,65-66
signup.js	80	14.28	100	88	69-71

Test Suites:	6 passed, 6 total
Tests:	25 passed, 25 total
Snapshots:	0 total
Time:	2.667 s, estimated 4 s
Ran all test suites.	