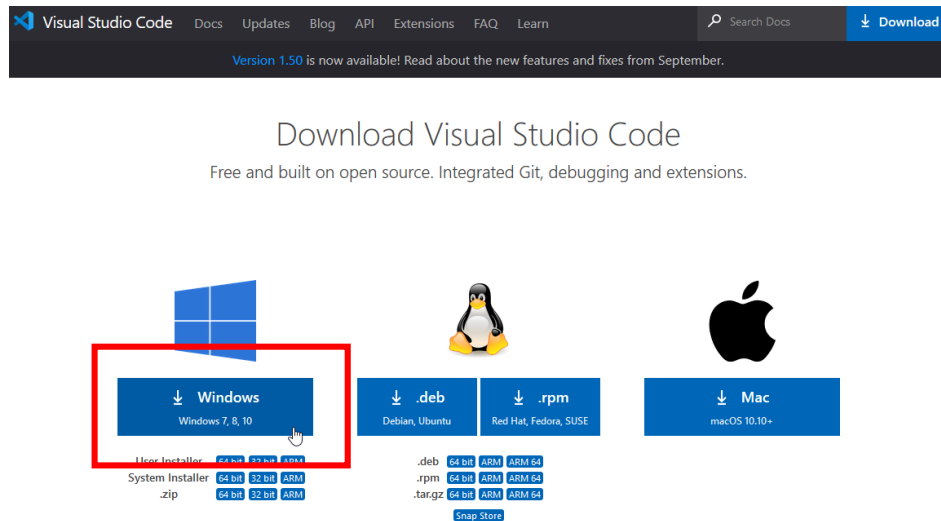


## Download dei software necessari:

VSCode:

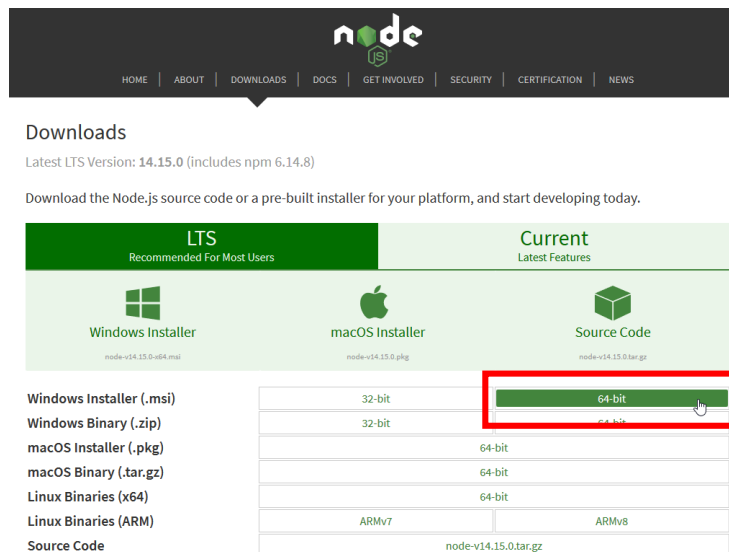
<https://code.visualstudio.com/download>



Salvare il file e procedere con l'installazione guidata.

NodeJs:

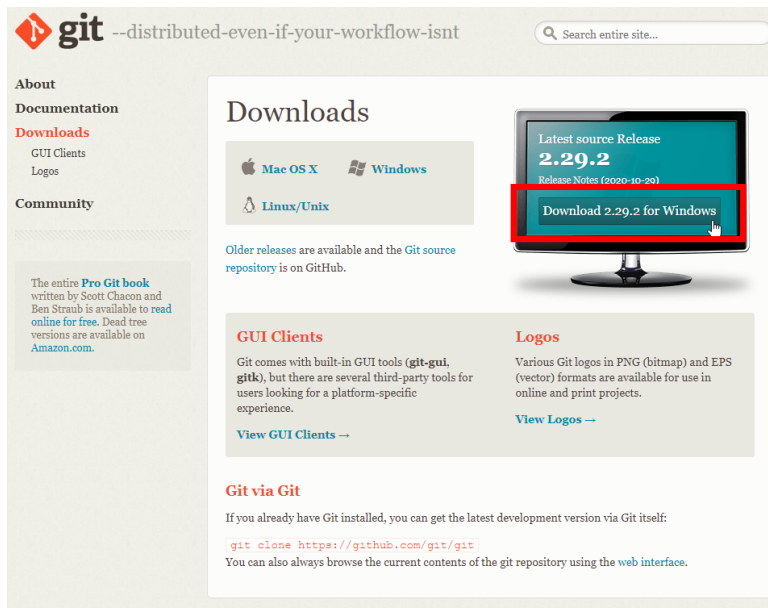
<https://nodejs.org/en/download/>



Salvare il file e procedere con l'installazione guidata.

GIT:

<https://git-scm.com/downloads>



Salvare il file e procedere con l'installazione guidata.

## Comandi principali per lo sviluppo:

Comandi GIT in Git Bash nella cartella:

- Creare repository GIT → `git init`
- Configurare utente GIT → `git config --global user.name "Salvatore", git config --global user.email "salvatore@mail.com"`
- Aggiungere tutti file → `git add -A`
- Fare commit dei file → `git commit -m "Modifiche effettuate"`
- Fare push dei file → `git push`
- Fare pull dei file → `git pull`
- Verificare lo stato dei repository → `git status`
- Creare un nuovo ramo → `git branch login`
- Cambiare ramo → `git checkout login`

Comandi NPM in VSCode da terminale (CTRL + MAIUSC + ò):

- Installare Angular → `npm install -g @angular/cli`
- Creare un progetto Angular → `ng new my-dream-app`
- Spostarsi nella cartella del progetto → `cd my-dream-app`
- Compilare e avviare il progetto → `ng serve`
- Creare un componente → `ng g c my-component`
- Creare un servizio → `ng g s my-service`
- Creare un modulo → `ng g m my-module`

## Come creare l'applicazione web “ChatDemo” client-side con Angular

- Avviamo Visual Studio Code e apriamo un nuovo terminale (shortcut CTRL + MAIUSC + ò)
- Lanciamo nel terminale il comando per installare Angular: `npm install -g @angular/cli`
- Lanciamo nel terminale il comando per la creazione del progetto Angular: `ng new chatdemo`
- Lanciamo nel terminale il comando per spostarsi nella cartella del progetto: `cd chatdemo`
- Spostiamoci all'interno della cartella app: `cd src/app`
- Creiamo un nuovo componente Angular all'interno della cartella app e chiamiamolo “chat”: `ng g c chat`
- Apriamo il component `chat.component.ts` e notiamo come è stato generato. È possibile accedere al template del componente tramite il selector (il tag html sarà `<app-chat>`).

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-chat',
  templateUrl: './chat.component.html',
  styleUrls: ['./chat.component.css']
})
export class ChatComponent implements OnInit {

  constructor() { }
  ngOnInit() {
  }
}
```

- Apriamo il template del component chat: `chat.component.html`
- Inseriamo i primi tag e creiamo un form con alcune label, input e button

```
<body>
  <h1>Una semplice chat demo:</h1>
  <form>
    <label for="inputName" >Nome e cognome: </label>
    <input type="text" id="inputName" name="identity"><br><br>
    <label
for="message">Scrivi il messaggio da inviare:</label><br><br>
    <textarea id="message" rows="6" cols="60" name="text"></textarea>
    <br><br>
    <input type="button" id="inputName" value="Invia"><br><br>
    <p>Clicca "Invia" per inviare il messaggio:</p>
  </form>
</body>
```

Il tag `<body>` identifica il corpo del template e quindi ospiterà tutta la struttura della pagina.

Il tag `<h1>` rappresenta invece un header e in particolare l'header più importante (da `<h1>` a `<h6>`).

Il tag `<form>` è utilizzato per creare un form di inserimento dati da parte di un utente e contiene al suo interno diversi elementi.

Il tag `<label>` serve per definire una label e quindi un testo per diversi elementi. Si associa a un altro tag tramite l'attributo *for* con valore uguale al valore dell'attributo *id* del tag associato.

Il tag `<input>` permette l'inserimento da parte dell'utente e in base all'attributo *type* può assumere diversi valori (testo, button...).

Il tag `<br>` inserisce una linea, testo a capo.

Il tag `<textarea>` definisce un'area di input multilinea.

Il tag `<p>` rappresenta un paragrafo.

- Apriamo il file *chat.component.css* per poter modificare lo stile del template html realizzato

```
body {
  width: 500px;
}

form {
  padding-top: 20px;
  padding-left: 20px;
  border-style: groove;
  background-color: lightblue;
}

label {
  font-weight: bold;
}
```

È possibile modificare lo stile dei tag scrivendo direttamente nel file .css il tag e tra parentesi graffe la proprietà da modificare. È possibile modificare il tag anche accedendo alla classe del tag nell'html. L'attributo *width* serve per determinare la larghezza del tag.

Il *padding* inserisce uno spazio all'interno di un'area delimitata.

*Border-style* determina lo stile del bordo.

*Background-color* definisce il colore dello sfondo.

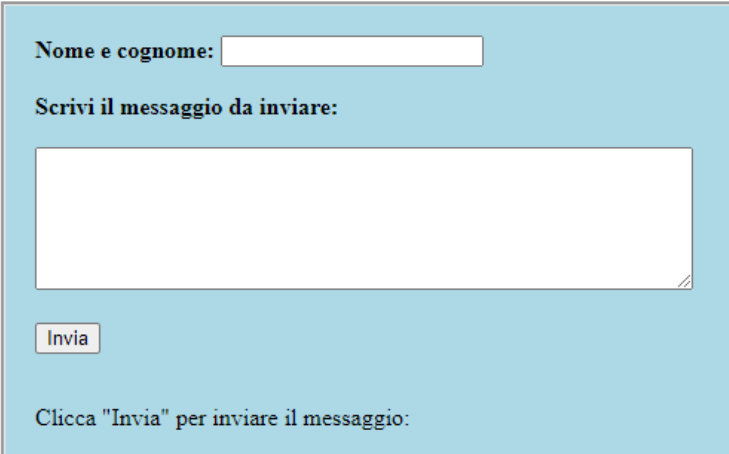
L'attributo *font-weight* determina lo spessore del formato (in questo caso bold – grassetto).

- Apriamo il file *app.component.html* e cancelliamo tutto il suo contenuto
- Sempre in *app.component.html* indicizziamo il nuovo componente *chat.component* appena creato, inserendo il selettore del componente stesso

```
<app-chat></app-chat>
```

- È possibile compilare il progetto realizzato, lanciando il comando *npm start* nel terminale
- Apriamo il browser Google Chrome e scriviamo nella barra dell'url <http://localhost:4200/> al fine di visualizzare l'applicazione sviluppata che dovrebbe presentarsi come segue:

## Una semplice chat demo:



- Il messaggio che verrà inviato avrà una forma standard, in accordo con quello che il server si aspetta di ricevere. E' necessario quindi creare una classe o modello che rappresenta il messaggio che verrà inviato.
- Creiamo una cartella all'interno di `.\src\app` e nominiamola *model*
- Spostiamoci all'interno della cartella con `cd .\src\app\model` e lanciamo dal terminale il comando: `ng g class message --skipTests`
- Per convenzione, rinominiamo il file generato come *message.model.ts*
- All'interno del file creiamo la classe *Message* come di seguito:

```
export class Message {  
  id: number = null;  
  name = '';  
  text = '';  
  createdAt: Date = null;  
}
```

- Spostiamoci all'interno del file *chat.component.ts* e creiamo due variabili di tipo *Message*: una servirà per ricevere dal server **tutti** i messaggi e un'altra servirà per inviare **un** messaggio

```
public messages: Message[];  
public message = new Message();
```

- Sempre all'interno del file creiamo una funzione che verrà richiamata al click del button e mostriamo nella console.log il messaggio che stiamo inviando:

```
sendMessage() {  
    console.log(this.message);  
}
```

- Il messaggio in realtà è vuoto in quanto non è associato ad alcun tag html e quindi ad alcun testo scritto dall'utente.
- Modifichiamo il <form> e inseriamo il riferimento al messaggio in modo che quando l'utente scrive il nome e il testo nelle aree apposite questi valori – stringhe vengono associate agli attributi del modello message (.name e .text). Inoltre richiamiamo il metodo *sendMessage()* al click del pulsante button:

```
<form>  
  <label for="inputName">Nome e cognome: </label>  
  <input type="text" id="inputName" [(ngModel)]="message.name" name="identity">  
<br><br>  
  <label for="message">Scrivi il messaggio da inviare:</label><br><br>  
  <textarea id="message" rows="6" cols="60" [(ngModel)]="message.text"  
name="text"></textarea><br><br>  
  <input type="button" id="inputName" (click)="sendMessage()" value="Invia">  
<br><br>  
  <p>Clicca "Invia" per inviare il messaggio:</p>  
</form>
```

- Infine è necessario importare *FormsModule* all'interno di app.module.ts:

```
import { FormsModule } from '@angular/forms';
```

```
imports: [  
    BrowserModule,  
    AppRoutingModule,  
    FormsModule  
],
```

- È possibile ora configurare un service Angular per comunicare con il server e quindi ottenere i dati tramite richieste http GET e inviare dati tramite http POST
- Un service serve per organizzare e condividere la logica, i modelli e le richieste al server tra diversi componenti
- Posizionarsi all'interno della cartella *chat* tramite il comando: *cd .\src\app\chat*
- Inviamo il comando dal terminale per creare un service: *ng g s chat*
- Apriamo il file generato *chat.service.ts* che si presenterà come segue:

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class ChatService {

  constructor() { }

}
```

- E' necessario ora creare tre variabili: due stringhe (*sendUrl* che rappresenta l'Url del server su cui verranno fatte le http POST, *receiveUrl* per le richieste http GET) e un JSON (httpOptions con appunto le opzioni delle richieste http).
- Per poter istanziare le httpOptions è necessario importare la libreria *HttpHeaders*

```
import { Injectable } from '@angular/core';
import { HttpHeaders } from '@angular/common/http';

@Injectable({
  providedIn: 'root',
})
export class ChatService {
  sendUrl: string;
  receiveUrl: string;
  httpOptions = {
    headers: new HttpHeaders({
      'Content-Type': 'application/json; charset=utf-8',
    }),
  };
  constructor(private http: HttpClient) {
  }
}
```

- All'interno del costruttore è quindi necessario stabilire il valore delle stringhe dell'URL su cui verranno fatte le richieste POST e GET al server:

```
this.sendUrl = 'http://localhost:5000/chat/new';
this.receiveUrl = 'http://localhost:5000/chat/all';
```

- E' possibile ora creare, all'interno del service, i metodi/funzioni che effettueranno le richieste http POST e GET verso il server. In particolare per le richieste POST sarà necessario passare al metodo anche il messaggio che verrà inviato:

```
// Http send Message
public async sendMessage(message) {
  return await this.http.post(this.sendUrl,message,this.httpOptions).toPromise();
}

// Http receive Message
public receiveMessage() {
  return this.http.get(this.receiveUrl).toPromise();
}
```

- Infine il file *chat.service.ts* sarà:

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';

@Injectable({
  providedIn: 'root',
})
export class ChatService {
  sendUrl: string;
  receiveUrl: string;
  httpOptions = {
    headers: new HttpHeaders({
      'Content-Type': 'application/json; charset=utf-8',
    }),
  };
  constructor(private http: HttpClient) {
    this.sendUrl = 'http://localhost:5000/chat/new';
    this.receiveUrl = 'http://localhost:5000/chat/all';
  }
  // Http send Message
  public async sendMessage(message) {
    return await this.http.post(this.sendUrl,message,this.httpOptions).toPromise();
  }

  // Http receive Message
  public receiveMessage() {
    return this.http.get(this.receiveUrl).toPromise();
  }
}
```

- Per ricevere e inviare i messaggi tramite richieste GET e POST da e verso il server, verranno quindi richiamati I metodi che sono stati realizzati in precedenza nel service.



- All'avvio dell'applicazione vogliamo quindi che vengano subito richiesti i messaggi inviati in precedenza:

```
constructor(  
  public chatService: ChatService  
) { }  
  
async ngOnInit() {  
  await this.chatService.receiveMessage().then(x =>this.messages=x as Message[]);  
}
```

- Modifichiamo la funzione *sendMessage()* che si occuperà ora di inviare il messaggio tramite POST. Questa verrà richiamata sempre al click del button "Invia". In questo caso sarà necessario eliminare i campi *id* e *createdAt* in quanto il server non li accetta. Richiediamo nuovamente i messaggi al server tramite GET.

```
async sendMessage() {  
  console.log(this.message);  
  delete this.message.id;  
  delete this.message.createdAt;  
  await this.chatService.sendMessage(this.message);  
  await this.chatService.receiveMessage().then(x =>this.messages=x as Message[]);  
  this.message.text = '';  
}
```

- Creiamo all'interno del template html una tabella per visualizzare i messaggi. Poiché non sappiamo quanti messaggi riceveremo e di conseguenza il numero di righe della tabella, sarà necessario effettuare un ciclo for in cui per ogni messaggio ricevuto verrà aggiunta una riga alla tabella.
- Spostiamoci nel file *chat.component.html* e all'interno del `<body>` e dopo la chiusura del tag `</form>` inseriamo la tabella come segue:

```
<table>  
  <colgroup>  
    <col style="width: 30%;">  
    <col style="width: 40%;">  
    <col style="width: 30%;">  
  </colgroup>  
  
  <thead>  
    <tr>  
      <th>Nome e cognome</th>  
      <th>Messaggio</th>
```

```
        <th>Creato</th>
      </tr>
    </thead>

    <tbody>
      <tr *ngFor="let message of messages">
        <td>{{message.name}}</td>
        <td>{{message.text}}</td>
        <td>{{message.createdAt | date:'dd-MMM-y, hh:mm'}}</td>
      </tr>
    </tbody>
  </table>
```

Il tag <table> ci permette di realizzare appunto una tabella.

Il tag <colgroup> definisce le colonne e lo stile da assegnare. In questo caso 3 colonne (Nome, messaggio, Creato) con la larghezza divisa in percentuale.

Il tag <thead> identifica l'intestazione della tabella, mentre <tr> la riga. All'interno di <tr> definiamo l'intestazione delle tre colonne assegnando il titolo.

Il tag <tbody> rappresenta il corpo della tabella ed è qui che verranno visualizzati i messaggi.

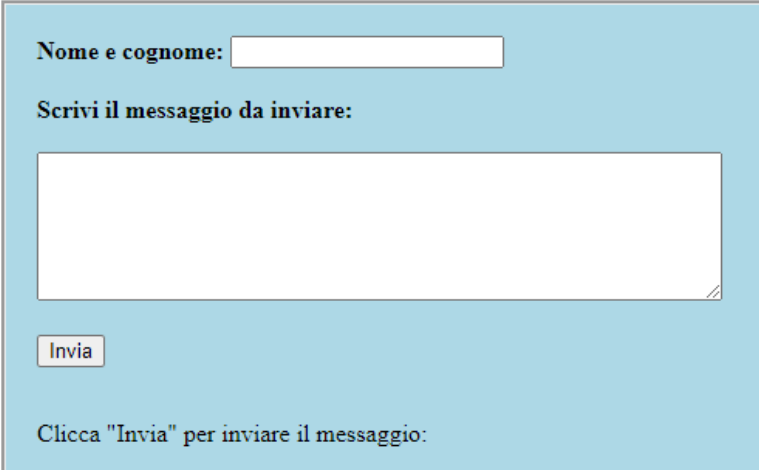
Poiché non sappiamo il numero di righe e quindi quanti <tr> aggiungere, all'interno del tag <tr> inseriamo un ciclo for che ci permette di aggiungere tante righe quanti sono i messaggi.

Infine <td> rappresenta la singola cella di ogni riga.

Tramite {{message. \*}} siamo in grado di accedere all'informazione associata a quell'attributo e in questo caso ricevuta tramite una GET al server.

- L'applicazione complessiva si mostrerà come segue:

## Una semplice chat demo:



Nome e cognome	Messaggio	Creato
Salvatore	Ciao Antonio	12-Nov-2020, 11:02