Manuale Installazione

SnakeByte (Gruppo 1):
Valeria Baleanu, Leonardo Pellizzon, Filippo Venzo, Giuseppe De Fina,
Francesco Pasqual, Christian Libralato, Luca Granziero
(2109911, 2111006, 2113705, 2113187, 2103119, 2101047, 2075512)

Informazioni documento			
Versione	**Data**	**Stato**	**Destinatari**
1.0.0	05/04/2026	Approvato	**Interni:** SnakeByte **Esterni:** prof. Vardanega Tullio, prof. Cardin Riccardo, Vimar

Contatti: snakebyteteam@gmail.com

Registro delle modifiche					
Versione	**Data**	**Autore**	**Verificatore**	**Approvatore**	**Descrizione**
1.0.0	05/04/2026	-	-	C. Libralato	Approvazione
0.1.0	04/04/2026	V. Baleanu	G. De Fina	-	Prima stesura

Indice

1 Introduzione **3**
1.1 Scopo del documento . . . 3
1.2 Glossario . . . 3
1.3 Riferimenti . . . 3
1.3.1 Riferimenti normativi . . . 3
1.3.2 Riferimenti informativi . . . 3

2 Requisiti **3**
2.1 Browser supportati . . . 3
2.2 Requisiti hardware . . . 4
2.3 Requisiti software . . . 4

3 Accesso e installazione **4**
3.1 Installazione dalla *repository* su *GitHub* . . . 4
3.1.1 Preparazione del Sistema . . . 4
3.1.2 Esecuzione del Sistema . . . 6

4 Esecuzione dei test **6**
4.1 Esecuzione dei test di unità . . . 7
4.2 Esecuzione dei test di integrazione . . . 7
4.3 Esecuzione dei test di accettazione . . . 7

1 Introduzione

1.1 Scopo del documento

Lo scopo del Manuale Installazione è quello di descrivere al lettore il procedimento di installazione del Sistema sviluppato da *SnakeByte* per il Capitolato$_G$ C9 proposto da *Vimar S.p.A.*. Il manuale illustra principalmente i requisiti e le modalità di installazione del Sistema. Inoltre sono descritti i vari test che è possibile eseguire per verificare il corretto funzionamento del Sistema.
In particolare, nel presente documento sono contenute le seguenti sezioni:

- **Requisiti**: una definizione di quelli che sono i requisiti minimi di *hardware*, *software* e *browser* per poter correttamente eseguire il Sistema;
- **Accesso e installazione**: una spiegazione dettagliata per poter eseguire il Sistema.
- **Test**: una spiegazione su come eseguire i vari test di unità, integrazione e accettazione;

1.2 Glossario

All'interno del documento possono essere presenti termini il cui significato può risultare ambiguo o sconosciuto, essi sono riportati in italico e alla loro prima occorrenza è associata una G a pedice ad indicare la presenza del suddetto termine all'interno del *glossario*$_G$, che è disponibile al seguente link: Glossario.

1.3 Riferimenti

1.3.1 Riferimenti normativi

- **Norme di Progetto 2.0.0**:
  https://snakebyteteam.github.io/3-PB/Documenti_interni/Norme_di_progetto/Norme_di_Progetto_v2.0.0.pdf
  (consultato il 05/04/2026).
- **Vimar View4Life Capitolato di Ingegneria del Software Università di Padova 2025-2026**:
  https://www.math.unipd.it/~tullio/IS-1/2025/Progetto/C9.pdf
  (consultato il 05/04/2026).

1.3.2 Riferimenti informativi

- **Glossario**:
  https://snakebyteteam.github.io/3-PB/Documenti_interni/Glossario/glossario_v2.0.0.pdf
  (consultato il 05/04/2026).

2 Requisiti

Di seguito sono elencati i requisiti minimi necessari per poter eseguire correttamente il Sistema.

2.1 Browser supportati

I *browser* supportati sono:

- **Safari**: versione 26 o superiore;
- **Chrome**: versione 147 o superiore;
- **Firefox**: versione 149 o superiore;

2.2 Requisiti hardware

In merito agli applicativi sviluppati, trattandosi di una *web application* containerizzata tramite *Docker* e basata su tecnologie moderne, i requisiti *hardware* dipendono principalmente dal numero di *container* in esecuzione e dal carico di lavoro previsto.

A scopo puramente esemplificativo, si riportano di seguito le specifiche della macchina su cui è attualmente installata un'istanza di *View4Life*:

- **CPU**: 1 Core, 2 GHz;
- **RAM**: 1 GB;
- **Spazio su disco**: 25 GB.

2.3 Requisiti software

In merito al Sistema Operativo, l'esecuzione del Sistema richiede esclusivamente la presenza del *software* di containerizzazione *Docker* (versione 29.4 o superiore), il quale consente di eseguire tutti i servizi applicativi all'interno di appositi *container*, senza la necessità di installare manualmente ulteriori dipendenze sulla macchina ospitante.
Grazie all'utilizzo di *Docker*, tutte le tecnologie necessarie al funzionamento del Sistema sono incluse e configurate all'interno dei *container*, garantendo isolamento, portabilità e semplicità di distribuzione.
La configurazione e l'avvio dei servizi sono gestiti automaticamente tramite strumenti di orchestrazione (es. *Docker Compose*), riducendo al minimo le operazioni richieste all'utente.
Pertanto, per l'esecuzione del Sistema è sufficiente disporre dei seguenti applicativi:

- **Docker**: versione 29.4.0 o superiore;
- **Docker Compose**: versione 5.0.0 o superiore.

3 Accesso e installazione

Il Sistema può essere utilizzato tramite installazione locale dalla *repository* su *GitHub*.

3.1 Installazione dalla *repository* su *GitHub*

Per eseguire il Sistema in locale è necessario soddisfare i requisiti descritti nella Sezione 2.

3.1.1 Preparazione del Sistema

Installazione di *Docker*

1. Verificare che *Docker* e *Docker Compose* siano installati eseguendo:

   ```
   docker --version
   docker compose version
   ```

2. Se non sono installati, seguire le istruzioni ufficiali disponibili su docs.docker.com/get-docker;
3. Assicurarsi che il servizio *Docker* sia in esecuzione:
   - Su Linux:

     ```
     sudo systemctl start docker
     ```

   - Su MacOS:

     ```
     open -a Docker
     ```

   - Su Windows (su PowerShell come amministratore):

     ```
     Start-Service com.docker.service
     ```

Installazione del Software

1. Aprire un terminale in una cartella a propria scelta, quindi clonare la *repository* del progetto:

   ```
   git clone https://github.com/SnakeByte/MVP.git
   ```

2. Spostarsi nella cartella appena creata:

   ```
   cd MVP
   ```

Configurazione del file `.env` Il file `.env` contiene le variabili d'ambiente necessarie al corretto funzionamento del Sistema, tra cui configurazioni relative alla base di dati, ai servizi *backend* e alle eventuali porte di esposizione dei *container Docker*.

1. Nella *directory* radice del progetto, creare un file `.env`;
2. Aprire il file con un editor di testo e inserire le seguenti voci (alle quali associare i propri dati):
   - **Configurazione server**
     - `PORT`: Porta su cui gira il server *backend*.
   - **Autenticazione e OAuth**
     - `CLIENTID`: fornito da *Vimar S.p.A.* dopo la registrazione al portale *MyVimar*;
     - `CLIENTSECRET`: fornito da *Vimar S.p.A.* dopo la registrazione al portale *MyVimar*;
     - `HOST1`: Endpoint di autenticazione OpenID Connect, recuperabile dalla specifica *KNX IoT Third Party API* di *Vimar*;
     - `HOST2`: Endpoint per ottenere il token di accesso, recuperabile dalla specifica *KNX IoT Third Party API* di *Vimar*;
     - `REDIRECT_URI`: URI interno di redirect dopo autenticazione;
     - `ACCESS_SECRET`: Secret per gestione access token;
     - `REFRESH_SECRET`: Secret per gestione refresh token.
   - **API e configurazione KNX/Vimar**
     - `HOST3`: Endpoint base delle API KNX/Vimar, recuperabile dalla specifica *KNX IoT Third Party API* di *Vimar*;
     - `PLANT_DOMAIN`: Endpoint di discovery KNX.
   - **Database**
     - `DBUSER`: Username del *database*;
     - `DBPASSWORD`: Password del *database*;
     - `DBPORT`: Porta del *database*;
     - `DBNAME`: Nome del *database*;
     - `PG_CONNECTION_STRING`: Stringa completa di connessione al *database*.
   - **Webhook e sottoscrizioni**
     - `SECRET_FOR_SUB`: Secret per la gestione delle sottoscrizioni;
     - `NODE_SUB_CALLBACK`: Endpoint interno callback per le sottoscrizioni;
     - `DATAPOINT_SUB_CALLBACK`: Endpoint interno callback per aggiornamenti datapoint.
   - **Integrazioni esterne**
     - `GROQ_API_KEY`: API key per integrazione con servizi Groq.
3. Salvare il file `.env`.

3.1.2 Esecuzione del Sistema

Avvio

1. Dalla *directory* radice del progetto, avviare l'infrastruttura eseguendo il comando:

   ```
   docker compose up -d
   ```

2. Attendere il completamento dell'avvio di tutti i *container*. L'operazione è considerata conclusa quando tutti i servizi risultano nello stato *running* o *healthy*.

3. Aprire il *browser* e accedere all'indirizzo:

   ```
   http://localhost:80
   ```

   per visualizzare la schermata iniziale del Sistema.

Spegnimento

1. Per arrestare il Sistema senza perdita dei dati persistenti, eseguire:

   ```
   docker compose down
   ```

2. I dati vengono conservati all'interno dei volumi *Docker*, che rimangono disponibili per successive riaccensioni del Sistema.

Ripristino

1. Per ripristinare il Sistema allo stato iniziale, eliminando anche i dati persistenti e ricostruendo le immagini *Docker*, eseguire:

   ```
   docker compose down -v --remove-orphans && docker compose up -d --build
   ```

2. Tutti i servizi vengono arrestati, i volumi eliminati e successivamente ricreati e riavviati. L'operazione potrebbe richiedere alcuni minuti a causa della ricostruzione delle immagini *Docker*.

Attenzione: il comando `docker compose down -v` elimina in modo irreversibile tutti i dati memorizzati nei volumi *Docker*, inclusi quelli del *database*. L'opzione `--remove-orphans` rimuove inoltre eventuali container non più definiti nel file di configurazione.

4 Esecuzione dei test

I test del sistema sono suddivisi in tre categorie: test di unità, test di integrazione e test di accettazione. Per eseguirli è necessario aver completato la fase di installazione descritta nella sezione 3.1.
I test sono organizzati tra **backend** e **frontend** e vengono eseguiti utilizzando gli strumenti forniti rispettivamente da *NestJS*$_G$ e *Angular*$_G$. Prima di eseguire i comandi descritti nelle sezioni successive, assicurarsi di avere *Node.js* installato. Successivamente, aprire un terminale nella *directory* radice del progetto (`MVP`) ed eseguire:

```
npm install
```

Tale comando installa tutte le dipendenze del progetto, necessarie per il corretto funzionamento dell'ambiente di test.

4.1 Esecuzione dei test di unità

I test di unità verificano il corretto funzionamento dei singoli moduli del Sistema in isolamento.

- **Backend**: aprire un terminale nella *directory* `backend` ed eseguire:

  ```
  npm run test unit
  ```

- **Frontend**: aprire un terminale nella directory `frontend` ed eseguire:

  ```
  ng test
  ```

  Al termine dell'esecuzione, il Sistema mostra un riepilogo dei test eseguiti e dei risultati ottenuti.

4.2 Esecuzione dei test di integrazione

I test di integrazione verificano la corretta interazione tra i diversi moduli del Sistema.

- **Backend**: aprire un terminale nella *directory* `backend` ed eseguire:

  ```
  npm run test integration
  ```

  Nota: è necessario che il *database* sia in esecuzione.

- **Frontend**: aprire un terminale nella directory `frontend` ed eseguire:

  ```
  ng test
  ```

4.3 Esecuzione dei test di accettazione

I test di accettazione (*end-to-end*) verificano il comportamento del Sistema dal punto di vista dell'utente finale, simulando scenari reali di utilizzo.

- **Backend**: aprire un terminale nella *directory* `backend` ed eseguire:

  ```
  npm run test:e2e
  ```

  Nota: questi test utilizzano una configurazione separata e richiedono che il *database* sia attivo.

- **Frontend**: aprire un terminale nella *directory* `frontend` ed eseguire:

  ```
  ng serve
  ```

  ```
  npx cypress run
  ```

  Al primo avvio del comando, `Cypress` potrebbe richiedere l'installazione di alcune dipendenze: in tal caso, confermare digitando `y` (*yes*) e premendo `Invio`.