> By grovesti (so sempre mi)

1. Cos'è il PoC: Definizione Formale

1.1 Definizione secondo Vardanega

> **Il Proof of Concept (PoC) è un eseguibile, sviluppato a inizio progetto, con tempi e costi strettamente limitati, per comprendere al meglio i requisiti tecnici e la fattibilità delle soluzioni tecnologiche individuate per la realizzazione del prodotto.**

1.2 Caratteristiche Fondamentali

Il PoC è caratterizzato da:

- 🎯 **Eseguibilità**: Deve essere codice funzionante, non solo documentazione o diagrammi
- ⏱️ **Tempi limitati**: Sviluppo rapido, non si tratta di un progetto completo
- 💰 **Costi contenuti**: Effort limitato, focus sull'essenziale
- 🔬 **Dimostrabilità**: Prova concreta ("the proof is in the pudding")
- 🛠️ **Fattibilità tecnologica**: Verifica che le tecnologie scelte siano adeguate

1.3 Il Principio "The Proof is in the Pudding"

L'espressione significa **"tastare con mano"** la fattibilità. Non basta dire "useremo React e DynamoDB", bisogna dimostrare che:

- React e DynamoDB si integrano correttamente
- Riusciamo a implementare funzionalità significative con questi strumenti
- La combinazione di tecnologie è adeguata al problema da risolvere

1.4 PoC come Software Esplorativo

Il PoC è **software esplorativo** (quindi diverso dal prodotto finale):

- **Obiettivo**: Esplorare la fattibilità, non costruire il sistema completo
- **Qualità del codice**: Funzionalità dimostrata > eleganza architettturale
- **Completezza**: Funzionalità selezionate > sistema completo

---

2. Cosa NON è il PoC: Errori Comuni

2.1 ❌ Errore #1: "Il PoC è una baseline avanzata del prodotto finale"

FALSO. Il PoC:

- NON è il punto minimo accettabile per il committente
- NON è una versione preliminare del prodotto
- NON è una baseline di prodotto

Perché è sbagliato: Il PoC è un **oggetto usa-e-getta** per capire come realizzare la soluzione. Serve a *voi* per comprendere, non al committente per valutare il prodotto.

2.2 ❌ Errore #2: "Il PoC è un prototipo sviluppato in velocità"

FALSO. Sebbene il PoC sia sviluppato rapidamente, NON è:

- Un prototipo buttato insieme senza criterio
- Codice scritto "a caso" solo per farlo funzionare
- Una dimostrazione superficiale

Perché è sbagliato: Il PoC deve dimostrare una **comprensione profonda** delle tecnologie, non solo farle "girare". La rapidità è nei tempi di sviluppo, non nella superficialità dell'implementazione.

2.3 ❌ Errore #3: "Il PoC è una demo per il cliente"

FALSO. Il PoC:

- NON è pensato primariamente per il proponente
- NON deve impressionare visivamente
- NON serve a vendere l'idea al committente

Perché è sbagliato: Il PoC serve **a voi** per capire se riuscite a costruire il prodotto con le tecnologie scelte. È uno strumento interno di validazione.

2.4 ❌ Errore #4: "Il PoC deve implementare tutti i casi d'uso"

FALSO. Il PoC deve:

- Selezionare 3-5 casi d'uso **rappresentativi**
- Coprire le funzionalità **critiche** e ad **alto rischio tecnologico**
- Dimostrare l'**integrazione** tra tecnologie

Perché è sbagliato: Nel PoC si **limitano le funzionalità** e si **usano le tecnologie al meglio**. Si fa una cernita, scegliendo quelle che possono andare meglio.

2.5 ❌ Errore #5: "Il PoC deve avere codice production-ready"

FALSO. Il PoC può avere:

- Gestione errori semplificata

- Validazione input minimale
- UI/UX funzionale ma non polished
- Codice esplorativo con "rough edges"

Perché è sbagliato: L'obiettivo è dimostrare fattibilità, non produrre codice pronto per il rilascio. Il focus è sulla comprensione tecnologica.

---

3. Scopo e Obiettivi del PoC

3.1 Scopi Primari

Il PoC ha **tre scopi fondamentali**:

1. 📊 **Comprendere i requisiti tecnici**
   - Quali sono le reali richieste tecnologiche del capitolato?
   - Come si traducono i casi d'uso in funzionalità concrete?
   - Quali API/interfacce sono necessarie?
2. 🔧 **Dimostrare la fattibilità delle tecnologie**
   - Le tecnologie scelte funzionano insieme?
   - Riusciamo a implementare le funzionalità core?
   - Ci sono limitazioni tecniche inaspettate?
3. ⚠️ **Identificare problemi tecnici e logistici**
   - Quali sono i rischi tecnologici?
   - Dove sono i punti critici dell'architettura?
   - Quali problemi potrebbero emergere durante lo sviluppo?

3.2 Obiettivi Secondari

Oltre agli scopi primari, il PoC permette di:

- **Validare l'architettura iniziale**: Le scelte architetturali reggono nella pratica?
- **Affinare la pianificazione**: Quanto tempo/effort richiedono le tecnologie scelte?
- **Formare il team**: Tutti acquisiscono familiarità con le tecnologie
- **Creare una baseline concettuale**: Punto di partenza (anche se usa-e-getta) per il design futuro

3.3 Domande a cui il PoC Risponde

Un buon PoC risponde a domande come:

- ✅ "Riusciamo a integrare React con DynamoDB?"
- ✅ "L'autenticazione con Cognito funziona con la nostra architettura multi-tenant?"
- ✅ "Le API REST che abbiamo progettato sono sufficienti per i casi d'uso principali?"

- ✅ "Il pattern di traduzione multilingua funziona con il nostro schema database?"

3.4 Domande a cui il PoC NON Risponde

Il PoC **non** deve rispondere a:

- ❌ "Il sistema scala a 10.000 utenti?" (problema di PB, non PoC)
- ❌ "L'UI è piacevole esteticamente?" (non è l'obiettivo)
- ❌ "Tutti i requisiti sono soddisfatti?" (troppo ambizioso)
- ❌ "Il codice è mantenibile a lungo termine?" (focus su fattibilità, non manutenibilità)

---

4. Il PoC nel Contesto della RTB

4.1 Cos'è la RTB (Requirements and Technology Baseline)

La **RTB** è la prima revisione obbligatoria del progetto didattico. Ha tre obiettivi:

1. **Fissare i requisiti** che il fornitore si impegna a soddisfare, in accordo con il proponente
2. **Motivare le tecnologie**, framework e librerie selezionate per la realizzazione del prodotto
3. **Dimostrare adeguatezza e fattibilità** tramite un Proof of Concept eseguibile

4.2 Artefatti della RTB

Per superare la RTB sono necessari:

Artefatto	Versione	Contenuto Principale
Analisi dei Requisiti	v1.0.0	Requisiti funzionali, non funzionali, casi d'uso, tracciamento
Norme di Progetto	v1.0.0	Way of working, strumenti, convenzioni
Piano di Progetto	v1.0.0	Pianificazione, preventivo, analisi rischi
Piano di Qualifica	v1.0.0	Strategia di verifica e validazione, metriche
Proof of Concept	v1.0.0	**Eseguibile + Documentazione tecnica**

4.3 PoC e Milestone SEMAT

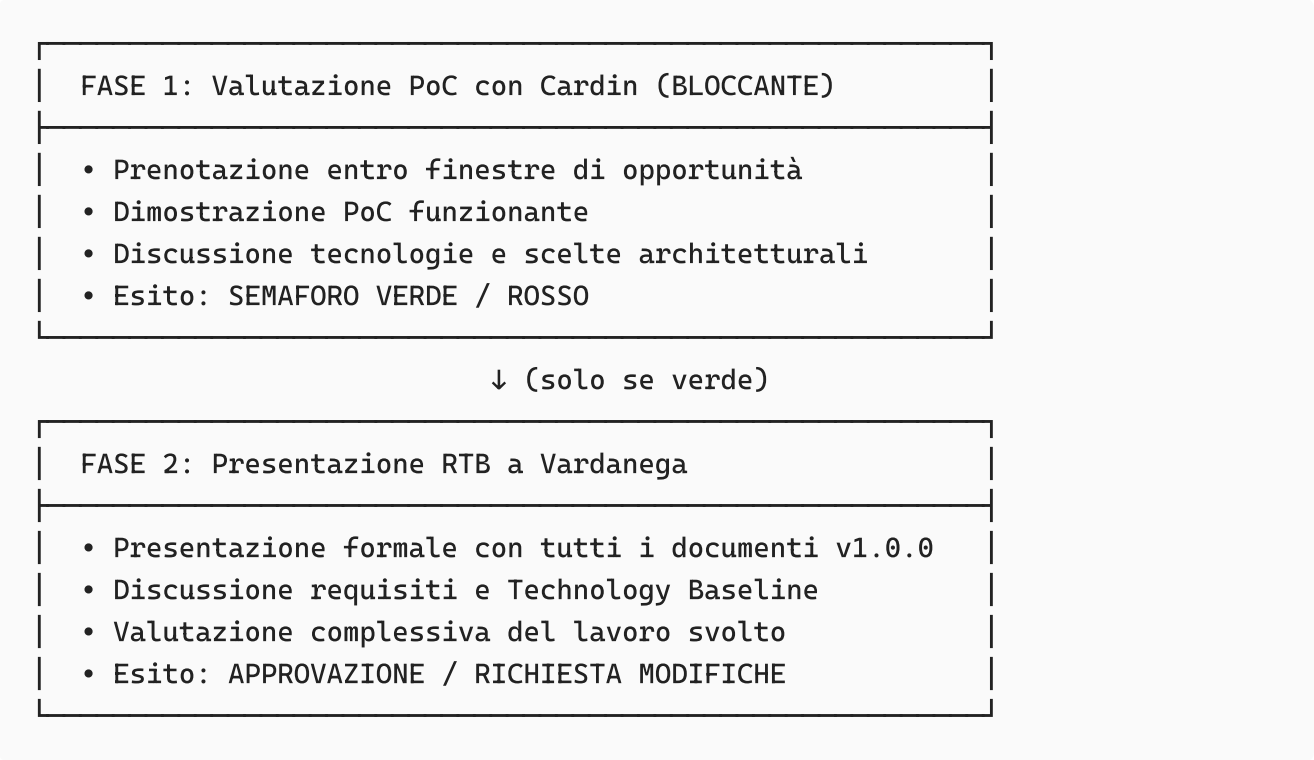
Secondo il modello **SEMAT**, il PoC corrisponde allo stato **"Demonstrable"** dell'architettura software:

- **Architecture Selected** → Scelta dell'architettura e delle tecnologie
- **Demonstrable** ✅ ← **IL POC SI COLLOCA QUI**
  - Dimostrazione delle principali caratteristiche dell'architettura

    - Gli stakeholder concordano sull'architettura
    - Decisione sulle principali interfacce e configurazioni di sistema
- **Usable** → Sistema utilizzabile (Product Baseline - PB)
- **Ready** → Sistema pronto per deployment (Customer Acceptance - CA)

4.4 Processo di Approvazione RTB

La RTB segue un **processo in 2 fasi**:



ATTENZIONE: Senza il semaforo verde da Cardin, NON si può procedere alla presentazione RTB con Vardanega.

4.5 Technology Baseline (TB)

Il PoC è parte integrante della **Technology Baseline**, che comprende:

- **Selezione tecnologie**: Quali tecnologie, framework, librerie sono state scelte?
- **Motivazione scelte**: Perché queste tecnologie? Quali alternative sono state considerate?
- **Dimostrazione adeguatezza**: Il PoC dimostra che le tecnologie funzionano
- **Identificazione limiti**: Quali sono i limiti/rischi delle tecnologie scelte?

4.6 Repository GitHub

Il PoC deve essere:

- Posto in un **repository GitHub pubblico** accessibile ai committenti
- Organizzato con README chiaro e istruzioni di setup/esecuzione

- Completo di documentazione tecnica e commenti nel codice

---

5. Come Realizzare un PoC: Metodologia Pratica

5.1 Step 1: Analisi del Capitolato

Obiettivo: Comprendere profondamente il dominio e i requisiti.

Azioni:

1. Leggere attentamente il capitolato e l'Analisi dei Requisiti (AdR)
2. Identificare i casi d'uso principali
3. Classificare i casi d'uso per:
   - **Criticità**: Quanto è importante per il sistema?
   - **Rischio tecnologico**: Quanto è difficile/incerto implementarlo?
   - **Rappresentatività**: Quanto rappresenta il dominio del problema?

Output: Lista prioritizzata di casi d'uso per il PoC.

Esempio pratico:

```
Capitolato: Sistema di localizzazione multilingua

Casi d'uso identificati:
UC1 - Creazione tenant          [Criticità: ALTA, Rischio: MEDIO,
Rappresentatività: ALTA]
UC2 - Accesso utenti            [Criticità: ALTA, Rischio: BASSO,
Rappresentatività: MEDIA]
UC3 - Configurazione tenant     [Criticità: ALTA, Rischio: ALTO,
Rappresentatività: ALTA]
UC4 - Gestione traduzioni       [Criticità: ALTA, Rischio: ALTO,
Rappresentatività: ALTA]
UC5 - Export traduzioni         [Criticità: MEDIA, Rischio: BASSO,
Rappresentatività: BASSA]
UC6 - Analytics utilizzo        [Criticità: BASSA, Rischio: MEDIO,
Rappresentatività: BASSA]

✅ Selezione per PoC: UC1, UC2, UC3, UC4 (coprono le funzionalità core)
❌ Esclusi dal PoC: UC5, UC6 (non critici per dimostrare fattibilità)
```

5.2 Step 2: Identificazione Tecnologie Critiche

Obiettivo: Identificare quali tecnologie hanno bisogno di essere validate.

Azioni:

1. Estrarre dal capitolato le **tecnologie obbligatorie**
2. Identificare le **tecnologie candidate** per i requisiti non vincolati
3. Classificare le tecnologie per:
   - **Familiarità del team**: Le conosciamo già?
   - **Complessità di integrazione**: Quanto è difficile farle funzionare insieme?
   - **Criticità per il progetto**: Quanto sono centrali per il sistema?

Output: Lista di tecnologie da validare nel PoC.

Esempio pratico:

```
Tecnologie Obbligatorie:
- AWS (infrastruttura cloud)          → Da validare: SÌ (complessa,
centrale)
- DynamoDB (database NoSQL)           → Da validare: SÌ (team non
familiare)
- Cognito (autenticazione)            → Da validare: SÌ (integrazione
complessa)

Tecnologie Candidate:
- React (frontend)                    → Da validare: NO (team già esperto)
- Node.js (backend)                   → Da validare: PARZIALE (setup API)
- Material-UI (UI components)         → Da validare: NO (libreria standard)

Focus PoC:
✅ Integrazione DynamoDB + Cognito + API Gateway
✅ Pattern multi-tenant con DynamoDB
✅ Setup pipeline deploy AWS
❌ UI React dettagliata (non critica per fattibilità)
```

5.3 Step 3: Design dell'Architettura PoC

Obiettivo: Definire un'architettura minimale ma rappresentativa.

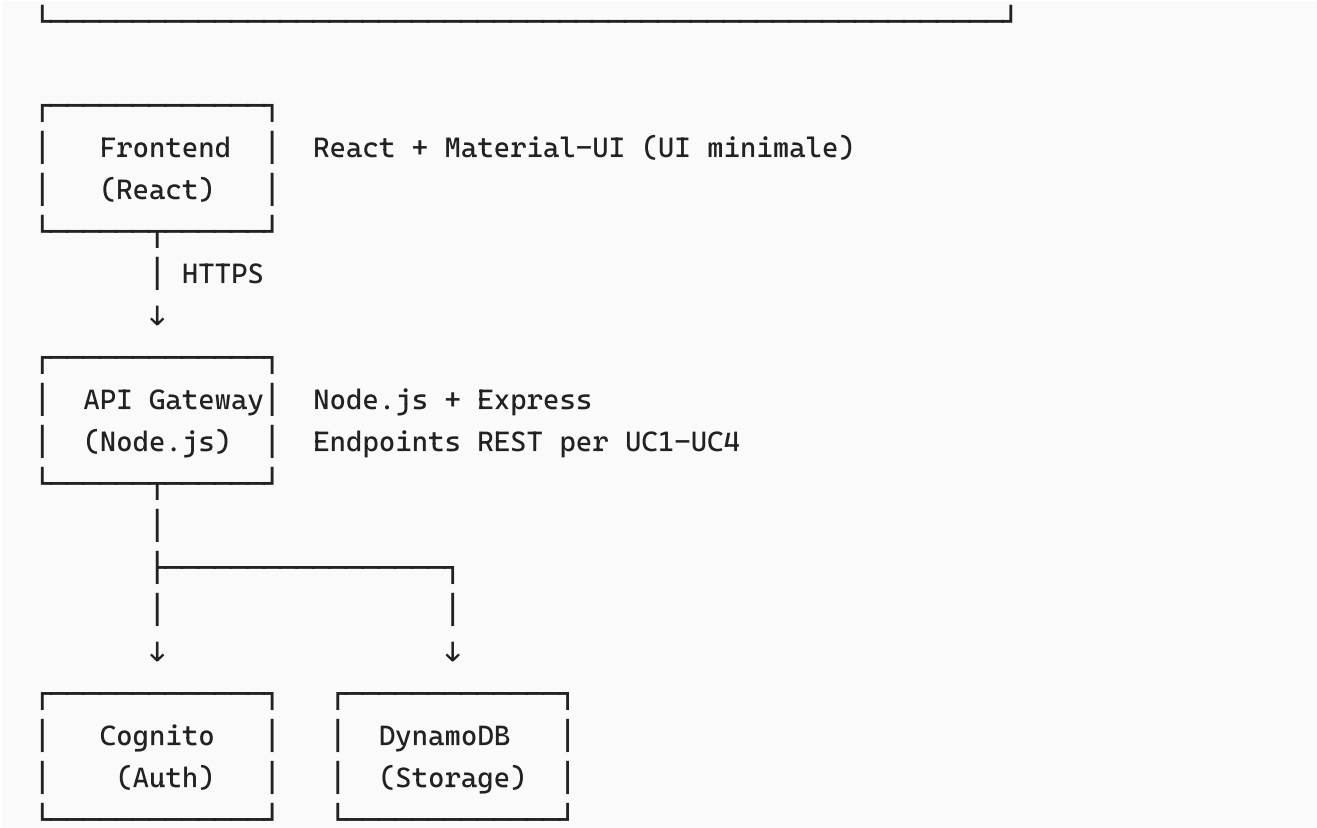
Azioni:

1. Disegnare lo **schema database** (relazionale e/o NoSQL) minimo
2. Definire le **API REST** essenziali per i casi d'uso selezionati
3. Identificare i **componenti principali** (frontend, backend, auth, storage)
4. Mappare **UC** → **API** → **Database** per tracciabilità

Output: Diagrammi architetturali e mapping UC-API-DB.

Esempio pratico:

```
┌─────────────────────────────────────────────────────────┐
│                    ARCHITETTURA PoC                     │
```




```
Database Schema (DynamoDB):
- Table: Tenants     [PK: tenantId]
- Table: Users       [PK: userId, SK: tenantId]
- Table: Texts       [PK: tenantId+language, SK: key+group]

API Mapping:
UC1 → POST /createTenant        → DynamoDB.Tenants.put()
UC2 → POST /login               → Cognito.authenticate()
UC3 → POST /{tenant}/insertText → DynamoDB.Texts.put()
UC4 → GET /{tenant}/Text        → DynamoDB.Texts.query()
```

5.4 Step 4: Implementazione Iterativa

Obiettivo: Costruire il PoC in modo incrementale e verificabile.

Metodologia consigliata: Sviluppo **verticale** (non orizzontale).

Approccio CORRETTO (Verticale):

```
Iterazione 1: UC1 completo (Frontend → API → DB)
  ✅ Form creazione tenant
  ✅ Endpoint POST /createTenant
  ✅ Scrittura DynamoDB
  ✅ Test end-to-end

Iterazione 2: UC2 completo (Frontend → Auth → DB)
  ✅ Form login
  ✅ Integrazione Cognito
```

```
   ✅ Verifica autenticazione
   ✅ Test end-to-end

Iterazione 3: UC3 completo (Frontend → API → DB)
   ✅ Form inserimento testo
   ✅ Endpoint POST /insertText
   ✅ Validazione e storage
   ✅ Test end-to-end

Iterazione 4: UC4 completo (Frontend → API → DB)
   ✅ Visualizzazione testi
   ✅ Endpoint GET /Text
   ✅ Query DynamoDB
   ✅ Test end-to-end
```

Approccio SBAGLIATO (Orizzontale):

```
❌ Fase 1: Tutto il frontend (senza backend funzionante)
❌ Fase 2: Tutte le API (senza database)
❌ Fase 3: Tutto il database (senza frontend)
❌ Fase 4: Integrazione finale (spesso fallisce)
```

Perché verticale è meglio:

- Dimostra fattibilità **concreta** ad ogni iterazione
- Permette **testing end-to-end** frequente
- Identifica **problemi di integrazione** subito
- Fornisce **feedback tangibile** al team

5.5 Step 5: Testing e Validazione

Obiettivo: Verificare che il PoC funzioni e dimostri la fattibilità.

Tipi di test nel PoC:

1. **Test Funzionali Manuali**
   - Eseguire manualmente ogni caso d'uso
   - Verificare che le API restituiscano i dati attesi
   - Controllare che l'integrazione frontend-backend funzioni
2. **Test di Integrazione**
   - Verificare l'integrazione tra tecnologie (React ↔ API ↔ DynamoDB)
   - Testare l'autenticazione end-to-end
   - Validare lo schema database con dati realistici
3. **Test di Fattibilità Tecnologica**
   - Le tecnologie si integrano senza problemi bloccanti?

- Ci sono limitazioni tecniche inaspettate?
- Le performance sono accettabili (anche se non ottimali)?

Output: Report di testing con evidenze che il PoC funziona.

5.6 Step 6: Documentazione

Obiettivo: Produrre documentazione chiara per i committenti e il team.

Documenti necessari:

1. **Documento PoC (v1.0.0)** - Vedi sezione 6
2. **README.md** nel repository GitHub con:
   - Descrizione breve del PoC
   - Istruzioni di setup e esecuzione
   - Prerequisiti (Node.js, npm, AWS CLI, etc.)
   - Comandi per avviare il sistema
   - Credenziali di test (se necessarie)
   - Link al documento PoC completo
3. **Video/Screencast Dimostrativo** (5-10 minuti):
   - Presentazione rapida dell'architettura
   - Esecuzione live dei casi d'uso
   - Evidenziazione delle tecnologie integrate

Esempio README.md:

```
# PoC - Sistema Localizzazione Multilingua

Proof of Concept per il capitolato [Nome Capitolato] del corso di Ingegneria
del Software 2024/2025.

## 🎯 Obiettivo
Dimostrare la fattibilità tecnica di un sistema multi-tenant per la gestione
di traduzioni utilizzando React, Node.js, DynamoDB e Cognito.

## 🛠️ Tecnologie Validate
- **Frontend**: React 18.2 + Material-UI 5.14
- **Backend**: Node.js 20.x + Express 4.18
- **Database**: AWS DynamoDB
- **Autenticazione**: AWS Cognito
- **Deployment**: AWS API Gateway + Lambda (opzionale)

## 📋 Casi d'Uso Implementati
- UC1: Creazione tenant
- UC2: Accesso utenti
- UC3: Configurazione tenant (inserimento testi)
```

- UC4: Visualizzazione testi multilingua

🚀 Setup e Esecuzione

Prerequisiti
- Node.js >= 20.x
- npm >= 10.x
- AWS CLI configurato con credenziali valide
- Account AWS con accesso a DynamoDB e Cognito

Installazione
\`\`\`bash
Clone repository
git clone https://github.com/[TEAM]/proof-of-concept.git
cd proof-of-concept

Install dependencies
npm install

Configure AWS credentials
aws configure

Setup DynamoDB tables (script provided)
npm run setup-db
\`\`\`

Avvio
\`\`\`bash
Start backend
cd backend
npm start # Server runs on http://localhost:3000

Start frontend (new terminal)
cd frontend
npm start # React app runs on http://localhost:3001
\`\`\`

Test
\`\`\`bash
Run integration tests
npm test

Test manual: navigate to http://localhost:3001
Credentials di test:
Username: test@example.com
Password: TestPassword123!
\`\`\`

📚 Documentazione
- [Documento PoC completo](./docs/Proof_of_Concept_v1.0.0.pdf)

```
- [Video dimostrativo](https://youtu.be/[VIDEO_ID])
- [API Documentation](./docs/API.md)

## 👥 Team
- [Nome1] - Redattore
- [Nome2] - Redattore
- [Nome3] - Verificatore
- [Nome4] - Verificatore
- [Nome5] - Verificatore
- [Nome6] - Amministratore

## 📝 Note
Questo è un Proof of Concept usa-e-getta. Il codice prodotto ha lo scopo di
validare la fattibilità tecnologica e NON è inteso come baseline di
prodotto.
```

---

6. Checklist Pre-Consegna

6.1 Checklist Documento PoC

Prima di considerare il documento PoC completo, verificare:

- [ ] **Frontespizio completo** con ruoli, destinatari, versione 1.0.0
- [ ] **Registro modifiche** popolato (anche se per v1.0.0 può avere solo entry finale)
- [ ] **Introduzione** spiega scopo e obiettivi del PoC
- [ ] **Casi d'uso implementati** chiaramente identificati e descritti
- [ ] **Architettura** documentata con diagrammi leggibili
- [ ] **Tecnologie** elencate con versioni specifiche e motivazioni
- [ ] **Database schema** presente (relazionale e/o NoSQL)
- [ ] **API specificate** in dettaglio (request/response)
- [ ] **Frontend** documentato con mockup/screenshot
- [ ] **Setup e esecuzione** con istruzioni passo-passo
- [ ] **Riferimenti** all'Analisi dei Requisiti v1.0.0 e Capitolato
- [ ] **Glossario** per termini tecnici non presenti in AdR
- [ ] **Indice** automatico funzionante
- [ ] **Numerazione pagine** corretta
- [ ] **Revisione ortografica** completata
- [ ] **Link repository GitHub** incluso nel documento

6.2 Checklist Repository GitHub

Prima di prenotare la revisione con Cardin, verificare:

- [ ] **Repository pubblico** e accessibile
- [ ] **README.md** completo con:
  - [ ] Descrizione breve
  - [ ] Tecnologie utilizzate
  - [ ] Prerequisiti
  - [ ] Istruzioni installazione
  - [ ] Istruzioni esecuzione
  - [ ] Credenziali di test
  - [ ] Link al documento PoC
- [ ] **Codice sorgente** organizzato in cartelle logiche (frontend/, backend/, docs/)
- [ ] **Dipendenze** specificate (package.json, requirements.txt, etc.)
- [ ] **Script di setup** per database/configurazione
- [ ] **.gitignore** configurato correttamente (no node_modules, no .env, etc.)
- [ ] **Commenti nel codice** per parti critiche
- [ ] **Documento PoC PDF** presente in `/docs/Proof_of_Concept_v1.0.0.pdf`
- [ ] **Licenza** (MIT o altra appropriata)
- [ ] **No credenziali hardcoded** nel codice (usare variabili ambiente)

6.3 Checklist PoC Eseguibile

Prima della dimostrazione con Cardin, verificare:

- [ ] **PoC avviabile** senza errori critici
- [ ] **Tutti i casi d'uso funzionano** end-to-end
- [ ] **Database** popolato con dati di test significativi
- [ ] **Autenticazione** funzionante (se presente)
- [ ] **API** restituiscono risposte corrette
- [ ] **Frontend** si connette correttamente al backend
- [ ] **Gestione errori minimale** presente (non crash su input inatteso)
- [ ] **Logging** utile per debugging (console logs, file logs)
- [ ] **Performance accettabili** (non deve essere lentissimo)
- [ ] **Cross-browser testing** (almeno Chrome/Firefox)
- [ ] **Video dimostrativo** registrato (5-10 minuti)
- [ ] **Screenshot** di ogni schermata principale

6.4 Checklist Preparazione Revisione Cardin

Prima della sessione con Cardin, preparare:

- [ ] **Laptop funzionante** con PoC già avviato e testato
- [ ] **Backup plan** (seconda macchina, cloud demo, video)
- [ ] **Slide di supporto** (5-10 slide max):

- [ ] Overview architettura
- [ ] Tecnologie validate
- [ ] Casi d'uso implementati
- [ ] Demo flow
- [ ] Problemi/rischi identificati

- [ ] **Risposta alle domande previste**:
  - [ ] "Perché avete scelto tecnologia X invece di Y?"
  - [ ] "Come gestite problema Z?"
  - [ ] "Questa soluzione scala?"
  - [ ] "Quali sono i limiti della vostra implementazione?"
- [ ] **Timing demo** provato (max 15 minuti per demo + discussione)
- [ ] **Tutti i membri del team** informati e pronti a rispondere

---

7. Valutazione con Cardin

7.1 Cosa Valuta Cardin

Il Prof. Cardin valuta il PoC su questi criteri:

1. **Fattibilità tecnica dimostrata** (40%)
   - Il PoC funziona realmente?
   - Le tecnologie sono integrate correttamente?
   - I casi d'uso implementati sono significativi?
2. **Profondità comprensione tecnologie** (30%)
   - Il team ha capito come funzionano le tecnologie?
   - Le scelte sono motivate tecnicamente?
   - Sono state considerate alternative?
3. **Qualità documentazione** (20%)
   - Il documento PoC è completo e chiaro?
   - Le API sono specificate in dettaglio?
   - Il setup è riproducibile?
4. **Identificazione problemi/rischi** (10%)
   - Il team ha identificato limitazioni?
   - Sono stati evidenziati rischi tecnologici?
   - C'è consapevolezza dei trade-off?

7.2 Domande Tipiche di Cardin

Preparatevi a rispondere a domande come:

Scelte Tecnologiche:

- "Perché avete scelto DynamoDB invece di PostgreSQL?"
- "React vs Angular vs Vue: perché React?"
- "Avete considerato l'uso di [tecnologia alternativa]?"

Integrazione e Architettura:

- "Come comunica il frontend con il backend?"
- "Come gestite l'autenticazione multi-tenant?"
- "Dove sono salvate le configurazioni sensibili?"

Fattibilità e Scalabilità:

- "Questa architettura scala a 1000 utenti? E a 100.000?"
- "Cosa succede se DynamoDB è temporaneamente non disponibile?"
- "Come gestite la consistenza dei dati?"

Problemi e Limitazioni:

- "Quali sono i limiti principali della vostra soluzione?"
- "Quali problemi avete incontrato durante lo sviluppo?"
- "Cosa cambiereste se doveste rifare il PoC?"

Testing e Qualità:

- "Come avete testato l'integrazione?"
- "Avete identificato bottleneck di performance?"
- "Il codice è pronto per essere esteso nella PB?"

7.3 Come Rispondere Efficacemente

DO ✅:

- Essere onesti su limitazioni e problemi
- Dimostrare comprensione profonda delle tecnologie
- Riferire scelte a requisiti specifici del capitolato
- Ammettere quando non si sa qualcosa e proporre come si può approfondire
- Mostrare tracciabilità UC → API → DB

DON'T ❌:

- Bluffare o inventare risposte tecniche
- Dire "abbiamo scelto X perché è popolare/di moda"
- Rispondere vagamente senza motivazioni tecniche
- Difendere scelte sbagliate invece di riconoscere errori
- Ignorare domande sui rischi/limitazioni

8.4 Esito Possibili

Semaforo VERDE 🟢:

- PoC dimostra chiaramente la fattibilità
- Tecnologie ben comprese e integrate
- Documentazione completa
- → Si può procedere con RTB Vardanega

Semaforo GIALLO 🟡:

- PoC funziona ma con alcune lacune
- Richiesti chiarimenti o piccoli fix
- Documentazione da migliorare
- → Revisione rapida richiesta prima di RTB

Semaforo ROSSO 🔴:

- PoC non funziona o non dimostra fattibilità
- Gravi problemi tecnici non risolti
- Documentazione insufficiente
- → Necessario rifare parti significative del PoC
