Esempio di documentazione

1 Introduzione 3

1.1 Informazioni sul progetto 3

1.2 Abstract 3

1.3 Scopo 3

2 Analisi 4

2.1 Analisi del dominio 4

2.2 Analisi e specifica dei requisiti 4

2.2.1 Spiegazione elementi tabella dei requisiti: 5

2.3 Use case 5

2.4 Pianificazione 5

2.5 Analisi dei mezzi 5

2.5.1 Software 6

2.5.2 Hardware 6

3 Progettazione 6

3.1 Design dell’architettura del sistema 6

3.2 Design dei dati e database 6

3.3 Design delle interfacce 6

3.4 Design procedurale 6

4 Implementazione 7

5 Test 7

5.1 Protocollo di test 7

5.2 Risultati test 8

5.3 Mancanze/limitazioni conosciute 8

6 Consuntivo 8

7 Conclusioni 8

7.1 Sviluppi futuri 8

7.2 Considerazioni personali 8

8 Glossario 8

9 Bibliografia 9

9.1 Bibliografia per articoli di riviste: 9

9.2 Bibliografia per libri 9

9.3 Sitografia 9

10 Allegati 9

# Introduzione

## Informazioni sul progetto

* Allieva: Linda Bytyqi
* Docente: Ingrid Cereda
* SAMT Progetti
* 12.09.2025 – 19.12.2025

## Abstract

È una breve e accurata rappresentazione dei contenuti di un documento, senza notazioni critiche o valutazioni. Lo scopo di un abstract efficace dovrebbe essere quello di far conoscere all’utente il contenuto di base di un documento e metterlo nella condizione di decidere se risponde ai suoi interessi e se è opportuno il ricorso al documento originale.

Può contenere alcuni o tutti gli elementi seguenti:

* **Background/Situazione iniziale**
* **Descrizione del problema e motivazione**: Che problema ho cercato di risolvere? Questa sezione dovrebbe includere l'importanza del vostro lavoro, la difficoltà dell'area e l'effetto che potrebbe avere se portato a termine con successo.
* **Approccio/Metodi**: Come ho ottenuto dei progressi? Come ho risolto il problema (tecniche…)? Quale è stata l’entità del mio lavoro? Che fattori importanti controllo, ignoro o misuro?
* **Risultati**: Quale è la risposta? Quali sono i risultati? Quanto è più veloce, più sicuro, più economico o in qualche altro aspetto migliore di altri prodotti/soluzioni?

Esempio di abstract:

*As the size and complexity of today’s most modern computer chips increase, new techniques must be developed to effectively design and create Very Large-Scale Integration chips quickly. For this project, a new type of hardware compiler is created. This hardware compiler will read a C++ program, and physically design a suitable microprocessor intended for running that specific program. With this new and powerful compiler, it is possible to design anything from a small adder, to a microprocessor with millions of transistors. Designing new computer chips, such as the Pentium 4, can require dozens of engineers and months of time. With the help of this compiler, a single person could design such a large-scale microprocessor in just weeks.*

## Scopo

Lo scopo del progetto **Nebula** è quello di realizzare un’applicazione interattiva e didattica, sviluppata in **Swift Playgrounds**, che consenta all’utente di esplorare il Sistema Solare in modo semplice, coinvolgente e visivamente accattivante.

L’app è pensata principalmente per **studenti e appassionati di astronomia**, che desiderano approfondire la conoscenza dei pianeti e dei corpi celesti in maniera intuitiva, senza la necessità di utilizzare strumenti complessi o professionali.

Attraverso **animazioni dinamiche** e un’interfaccia accessibile, il progetto intende avvicinare le persone al mondo dell’astronomia, trasformando l’apprendimento in un’esperienza interattiva.  
Ogni pianeta potrà essere selezionato per visualizzare **informazioni, immagini** e una **curiosità del giorno**.

L’obiettivo finale è quello di unire **educazione e tecnologia**, offrendo un prodotto che:

* stimoli la curiosità e l’interesse scientifico;
* renda la conoscenza del Sistema Solare più comprensibile e interattiva;
* favorisca un apprendimento visivo attraverso immagini e animazioni;
* mostri come il linguaggio **Swift** possa essere utilizzato per creare applicazioni multimediali anche in ambito educativo.

Inoltre, il progetto mira a dimostrare l’utilità di **Swift Playgrounds** come strumento non solo per l’apprendimento della programmazione, ma anche per la realizzazione di piccoli prodotti digitali completi e interattivi.

In sintesi, **Nebula** ha come scopo principale quello di **divulgare la conoscenza astronomica in modo moderno e accessibile**, creando un ponte tra **scienza e tecnologia** attraverso un applicativo semplice ma di grande impatto visivo.

# Analisi

## Analisi del dominio

Il progetto nasce dall’esigenza di creare un’applicazione didattica e divulgativa che consente di esplorare il Sistema Solare in modo interattivo. L’astronomia è da sempre un campo affascinante, ma purtroppo spesso e volentieri le risorse disponibili sono adatte per un pubblico esperto, ciò le rende più complesse da comprendere per chi è passionato ma non se ne intende molto. L’obbiettivo quindi è quello di proporre un prodotto semplice, intuitivo e coinvolgente che permette agli appassionati di informarsi sull’astronomia.

Il prodotto è pensato per essere utilizzato principalmente in ambito informativo da chi ne è appassionato oppure in contesto scolastico. Al momento esistono app avanzate come Sky Map, ma c’è bisogno di un ulteriore conoscenza per comprenderle poiché sono abbastanza complesse. Gli utenti principali sono studenti, appassionati di astronomia principianti e professori, l’obbiettivo è quello di avere un’interfaccia semplice che permetta di soddisfare i bisogni principali degli utenti perciò non hanno bisogno di competenze particolari per utilizzare l’applicativo.

## Analisi e specifica dei requisiti

In questo capitolo si descrivono le funzionalità che il sistema dovrà offrire e le caratteristiche qualitative richieste, in modo che lo sviluppo possa basarsi su una base chiara e condivisa.

**Requisiti Funzionali**

|  |  |
| --- | --- |
| **ID: REQ-01** | |
| **Nome** | Schermata Home |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Si entra nell’applicativo |
| **ID: REQ-02** | |
| **Nome** | Sistema Solare |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Si potrà girare il Sistema Solare a piacimento e zoomare. |
| **Sotto requisiti** | |
| **001** | Si necessita il corretto funzionamento del Req-01. |
| **ID: REQ-03** | |
| **Nome** | Scelta del pianeta |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Una volta scelto il pianeta, bisognerà premerlo. |
| **Sotto requisiti** | |
| **001** | Si necessita il corretto funzionamento del Req-02 |
| **ID: REQ-04** | |
| **Nome** | Informazioni pianeta |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Lettura sulle informazioni riguardante il pianeta con foto scattata dalla NASA |
| **Sotto requisiti** | |
| **001** | Informazioni corrette |
| **ID: REQ-05** | |
| **Nome** | Luna |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Premendo sulla Luna si avrà il calendario lunare e le sue varie fasi. |
| **Sotto requisiti** | |
| **001** | Si necessita il corretto funzionamento del Req-02 |
| **ID: REQ-06** | |
| **Nome** | Curiosità del giorno |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Premendo un bottone si aprirà una schermata che mostrerà la curiosità del giorno. |
| **Sotto requisiti** | |
| **001** | File JSON con tutti i fatti. |

**Requisiti Non Funzionali**

Questi requisiti riguardano le caratteristiche di qualità, vincoli e condizioni generali che il sistema deve rispettare.

* **Usabilità**: l’interfaccia deve essere semplice e immediata, adatta a utenti con poca esperienza.
* **Prestazioni**: le animazioni devono essere fluide (es. rotazione dei pianeti senza scatti) e i tempi di risposta rapidi.
* **Compatibilità**: l’app deve funzionare nell’ambiente di sviluppo scelto (Swift Playgrounds su iPad).
* **Affidabilità e correttezza dei dati**: le informazioni mostrate devono derivare da fonti affidabili (es. NASA) e il caricamento dei dati JSON deve avvenire senza errori.
* **Estetica e design**: tema spaziale coerente (sfondo scuro, stelle), testo leggibile con buon contrasto.
* **Scalabilità**: il sistema deve permettere in futuro l’aggiunta di nuovi pianeti o curiosità senza rifacimenti importanti.
* **Sicurezza e privacy**: se sono gestite credenziali, l’accesso deve essere protetto e l’utente deve essere correttamente identificato.

### Spiegazione elementi tabella dei requisiti:

La tabella dei requisiti rappresenta un riepilogo strutturato di tutte le caratteristiche che il sistema deve possedere.  
Ogni requisito è identificato da un codice univoco e descritto attraverso una serie di campi che ne specificano la natura, la priorità, la versione e le eventuali relazioni con altri requisiti.

### ****Elementi della tabella:****

**ID:**  
È l’identificativo univoco del requisito.  
Serve per distinguere in modo chiaro ogni requisito dagli altri e per poterlo facilmente richiamare all’interno della documentazione.  
Generalmente viene espresso nella forma REQ-XX (es. REQ-01 per il primo requisito).

**Nome:**  
Indica una breve descrizione del requisito, ossia il suo titolo sintetico che riassume in poche parole la funzionalità o la caratteristica richiesta.  
Ad esempio: “Schermata iniziale”.

**Priorità:**  
Specifica l’importanza del requisito all’interno del progetto.  
La priorità viene definita in accordo con il committente e può avere diversi livelli (ad esempio:

* **1 = alta** → requisito fondamentale, necessario per il funzionamento del sistema;
* **2 = media** → requisito importante ma non essenziale;
* **3 = bassa** → requisito secondario o opzionale).  
  La priorità aiuta a stabilire quali requisiti devono essere implementati per primi.

**Versione:**  
Indica la versione del requisito, utile per tenere traccia di eventuali modifiche nel tempo.  
Ogni aggiornamento o revisione comporta un incremento della versione (es. 1.0 → 1.1 → 2.0).  
Nella documentazione principale viene mostrata solo l’ultima versione, mentre le precedenti vengono archiviate nei **diari di progetto**.

**Note:**  
Campo dedicato a eventuali osservazioni aggiuntive, specifiche tecniche, riferimenti a fonti esterne o collegamenti con altri requisiti.  
Ad esempio: “Si necessita il corretto funzionamento del REQ-01” oppure “Immagini fornite dalla NASA”.

**Sotto-requisiti:**  
Elenca gli elementi più specifici o le sotto-funzionalità che compongono il requisito principale.  
Servono per descrivere in modo più dettagliato come il requisito sarà realizzato o quali condizioni devono essere soddisfatte affinché il requisito sia considerato completo.  
Ad esempio:

* REQ-03 (Sistema Solare interattivo) può avere come sotto-requisiti:
* REQ-03.001: “I pianeti devono ruotare intorno al Sole”;
* REQ-03.002: “Deve essere possibile zoomare”.

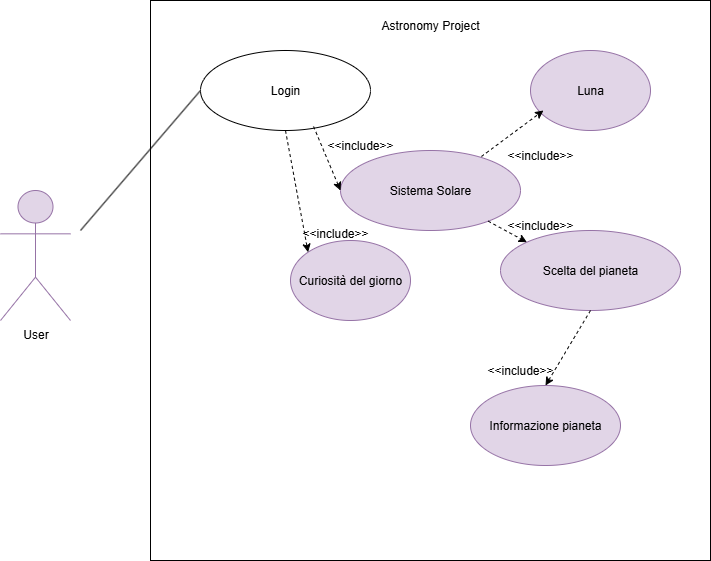
## Use case

Lo Use Case (o Caso d’Uso) rappresenta uno strumento di analisi che serve a descrivere come gli attori interagiscono con il sistema per raggiungere un determinato obiettivo.

In altre parole, uno uso case mostra in modo chiaro che cosa fa il sistema (non come lo fa) e in che modo gli utenti o le altre entità esterne lo utilizzano.

Nel contesto del progetto *Nebula*, l’attore principale è:

* **Utente** → colui che utilizza l’app per esplorare il sistema solare e leggere le informazioni.



## Pianificazione

La pianificazione è necessaria per poter avere una visuale chiara di ciò che si andrà a sviluppare. In seguito si svilupperà un secondo diagramma alla fine del progetto, così da poter confrontare i due per vedere se ciò che si ha pianificato si è avvicinato a ciò che abbiamo realizzato. Per fare ciò ho utilizzato il programma Project per creare un diagramma di Gantt inserendo tutte le attività che andrò a fare in questo progetto e i loro rispettivi tempi.

La seguente immagine mostra il diagramma di Gantt:

|  |
| --- |
| Figura 1: Esempio di diagramma di Gantt. |

## Analisi dei mezzi

**Swift e SwiftUI** è il framework principale per la realizzazione di applicazioni per dispositivi Apple, tra cui iPhone, iPad, Mac, Apple Watch e Apple TV. Swift è un linguaggio di programmazione moderno, sicuro e veloce, creato da Apple per lo sviluppo di app su tutti i suoi dispositivi. SwiftUI è il framework **dichiarativo** di Apple per costruire interfacce grafiche, introdotto nel 2019, che semplifica la progettazione di layout, animazioni e navigazione tra schermate.

SwiftUI permette di realizzare interfacce utilizzando una **sola codebase** e gestisce automaticamente l’aggiornamento della UI quando cambiano i dati, riducendo il codice necessario rispetto a framework tradizionali come UIKit. Inoltre, consente di personalizzare stili e comportamenti specifici per ciascun dispositivo Apple, mantenendo però una coerenza grafica tra piattaforme diverse.

Questo framework è particolarmente indicato per chi vuole sviluppare applicazioni native per l’ecosistema Apple utilizzando Swift, con un approccio moderno e intuitivo, e per chi desidera integrare animazioni, gesti e dati in modo semplice e reattivo.

Per il progetto, ho utilizzato Swift Playground 4, con **Swift 5.9** come ambiente di sviluppo, con SwiftUI come framework per la creazione dell’interfaccia grafica.

### Software

Swift Playground

### Hardware

Ipad 10th Generation

Keyboard Amazon

# Progettazione

Questo capitolo descrive esaustivamente come deve essere realizzato il prodotto fin nei suoi dettagli. Una buona progettazione permette all’esecutore di evitare fraintendimenti e imprecisioni nell’implementazione del prodotto.

## Design dell’architettura del sistema

Descrive:

* La struttura del programma/sistema lo schema di rete...
* Gli oggetti/moduli/componenti che lo compongono.
* I flussi di informazione in ingresso ed in uscita e le relative elaborazioni. Può utilizzare *diagrammi di flusso dei dati* (DFD).
* Eventuale sitemap

## Design dei dati e database

Descrizione delle strutture di dati utilizzate dal programma in base agli attributi e le relazioni degli oggetti in uso.

Schema E-R, schema logico e descrizione.

Se il diagramma E-R viene modificato, sulla doc dovrà apparire l’ultima versione, mentre le vecchie saranno sui diari.

## Design delle interfacce

Descrizione delle interfacce interne ed esterne del sistema e dell’interfaccia utente. La progettazione delle interfacce è basata sulle informazioni ricavate durante la fase di analisi e realizzata tramite mockups.

## Design procedurale

Descrive i concetti dettagliati dell’architettura/sviluppo utilizzando ad esempio:

* Diagrammi di flusso e Nassi.
* Tabelle.
* Classi e metodi.
* Tabelle di routing
* Diritti di accesso a condivisioni …

Questi documenti permetteranno di rappresentare i dettagli procedurali per la realizzazione del prodotto.

# Implementazione

In questo capitolo dovrà essere mostrato come è stato realizzato il lavoro. Questa parte può differenziarsi dalla progettazione in quanto il risultato ottenuto non per forza può essere come era stato progettato.

Sulla base di queste informazioni il lavoro svolto dovrà essere riproducibile.

In questa parte è richiesto l’inserimento di codice sorgente - Print Screen - di maschere solamente per quei passaggi particolarmente significativi e/o critici.

Inoltre, dovranno essere descritte eventuali varianti di soluzione o scelte di prodotti con motivazione delle scelte.

Non deve apparire nessuna forma di guida d’uso di librerie o di componenti utilizzati. Eventualmente questa va allegata.

Per eventuali dettagli si possono inserire riferimenti ai diari.

# Test

## Protocollo di test

Definire in modo accurato tutti i test che devono essere realizzati per garantire l’adempimento delle richieste formulate nei requisiti. I test fungono da garanzia di qualità del prodotto. Ogni test deve essere ripetibile alle stesse condizioni.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-001  REQ-012 | **Nome:** | Import a card with KIC, KID and KIK keys, but not shown with the GUI |
| **Descrizione:** | Import a card with KIC, KID and KIK keys with no obfuscation, but not shown with the GUI | | |
| **Prerequisiti:** | Store on local PC: Profile\_1.2.001.xml (appendix n\_n) and Cards\_1.2.001.txt (appendix n\_n).  PIN (OTA\_VIEW\_PIN\_PUK\_KEY) and ADM (OTA\_VIEW\_ADM\_KEY) user right not set. | | |
| **Procedura:** | 1. Go to “Cards manager” menu,  in main page click “Import Profiles” link, Select the “1.2.001.xml” file, Import the Profile 2. Go to “Cards manager” menu,  in main page click “Import Cards” link, Select the “1.2.001.txt” file, Delete the cards,  Select the “1.2.001.txt” file, Import the cards 3. Research the “41795924770” Card, Click the imsi card link Check the card details 4. Execute the SQL: SELECT imsi, dir, keyset, cntr, rawtohex(kickey), rawtohex(kidkey), rawtohex(kikkey), rawtohex(chv), rawtohex(dap)FROM otacardkey a where imsi='340041795924770' ORDER BY keyset; | | |
| **Risultati attesi:** | Keys visible in the DB (OtaCardKey) but not visible in the GUI (Card details) | | |

## Risultati test

Tabella riassuntiva in cui si inseriscono i test riusciti e non del prodotto finale. Se un test non riesce e viene corretto l’errore, questo dovrà risultare nel documento finale come riuscito (la procedura della correzione apparirà nel diario), altrimenti dovrà essere descritto l’errore con eventuali ipotesi di correzione.

## Mancanze/limitazioni conosciute

Descrizione con motivazione di eventuali elementi mancanti o non completamente implementati, al di fuori dei test case. Non devono essere riportati gli errori e i problemi riscontrati e poi risolti durante il progetto.

# Consuntivo

Consuntivo del tempo di lavoro effettivo e considerazioni riguardo le differenze rispetto alla pianificazione (cap. 1.7) (ad esempio Gantt consuntivo).

# Conclusioni

Quali sono le implicazioni della mia soluzione? Che impatto avrà? Cambierà il mondo? È un successo importante? È solo un’aggiunta marginale o è semplicemente servita per scoprire che questo percorso è stato una perdita di tempo? I risultati ottenuti sono generali, facilmente generalizzabili o sono specifici di un caso particolare? ecc.

## Sviluppi futuri

Migliorie o estensioni che possono essere sviluppate sul prodotto.

## Considerazioni personali

Cosa ho imparato in questo progetto? ecc.

# Glossario

Inserite una semplice tabella con due colonne che spieghi i termini specifici del progetto (lista dei termini in ordine alfabetico A-Z)

Esempio:

|  |  |
| --- | --- |
| **Termine** | **Descrizione** |
| AJAX | **Asynchronous JavaScript And XML**: una tecnica che permette di eseguire richieste ed ottenere dati da una pagina web in modo asincrono. |
| CSS | **Cascading Style Sheets**: linguaggio che permette di definire il layout e la grafica di una pagina web. |

# Bibliografia

## Bibliografia per articoli di riviste:

1. Cognome e nome (o iniziali) dell’autore o degli autori, o nome dell’organizzazione,
2. Titolo dell’articolo (tra virgolette),
3. Titolo della rivista (in italico),
4. Anno e numero
5. Pagina iniziale dell’articolo.

## Bibliografia per libri

1. Cognome e nome (o iniziali) dell’autore o degli autori, o nome dell’organizzazione,
2. Titolo del libro (in italico),
3. ev. Numero di edizione,
4. Nome dell’editore,
5. Anno di pubblicazione,
6. ISBN.

## Sitografia

1. URL del sito (se troppo lungo solo dominio, evt completo nel diario),
2. Eventuale titolo della pagina (in italico),
3. Data di consultazione (GG-MM-AAAA).

**Esempio:**

* http://standards.ieee.org/guides/style/section7.html, *IEEE Standards Style Manual*, 07-06-2008.

# Allegati

Elenco degli allegati, esempio:

* Diari di lavoro
* Codici sorgente/documentazione macchine virtuali
* Istruzioni di installazione del prodotto (con credenziali di accesso) e/o di eventuali prodotti terzi
* Documentazione di prodotti di terzi
* Eventuali guide utente / Manuali di utilizzo
* Mandato e/o QdC
* Prodotto
* …