Documentation for Software engineering project

Nicolas Anselmi, David Guzman Piedrahita and Marco Vinciguerra

14 febbraio 2022

1 Project Plan

1.1 Introduction

Il progetto prevede lo sviluppo di una mobile app per gestire la prenotazione di un negozio di parruccheria. C'è la possibilità di avere due tipi di utente:

La novità di questo progetto consiste nel fatto che è un tipo di sistema P2P in cui un utente può essere contemporaneamente cliente e (se vuole) gestore. Questo modello di business è fatto anche da Uber in cui un autista può essere sia cliente che autista.
Il target della clientela è market driven in quanto il core business dell'azienda si basa su percentuali delle transazioni.
Il cliente ha la possibilità di prenotare diversi tipi di acconciatura direttamente senza interfacciarsi /chiamare direttamente il proprietario del negozio, usando invece tramite l'applicazione. Ogni tipologia di taglio selezionabile e può consentire una data customizzabile di prenotazione da parte del cliente.
I membri del team sono: Nicolas Anselmi, David Guzman Piedrahita e Marco Vinciguerra.

1.2 Process model

Il life cycle del progetto è agile, in particolare la tecnica utilizzata è SCRUM con sprint di circa 5 giorni. Inoltre viene utilizzato un triage per gestire i compiti (MoSCoW).
Inoltre viene applicato un modello di prototipazione incrementale in cui ad ogni git viene aggiunto una funzionalità al sito e in più deve essere sempre disponibile su Github una versione funzionante del progetto.
Per accelerare il processo di apprendimento viene applicato anche il processo di pair programming, pratica molto utilizzata nei metodi AGILE.
Per quanto riguarda i requirements si utilizza il Kano Model el il MoSCoW model. Il periodo di sviluppo parte poco prima di Natale. Ogni giorno verso le 9 30 del mattino c'è un daily scrum tenuto dallo scrum master in cui si discutono le problematiche riscontrate durante il giorno precedente e le possibili soluzioni a queste.

1.3 Organization of the project

Il progetto, dovuto alla sua natura, deve intefacciarsi sia con utenti che usufruiscono del servizio di prenotazione, sia da utenti che mettono a disposizioni i loro servizi commerciali. Il team di sviluppo è composto dai succitati integranti. Per portare a termine l'applicazione, ci sono dei knowledge-gap che dovranno essere colmati tramite la lettura di documentazione e l'uso di risorse online: particolarmente nel caso del framework Flutter per il frontend development.

1.4 Standards, guidelines, procedures

I principali linguaggi di programmazione del progetto sarà dart, quest'ultimo viene esteso tramite flutter. Si usano i coding standards di flutter.
Per gestire l'assegnazione e il corretto sviluppo si usa un template di Notion per gestire i compiti.
Gli IDE che vengono utilizzati sono: VSC e terminale con VIM.

1.5 Management activities

Ogni settimana viene fatto un report informale sui progressi in corso fatti dal team di sviluppo per avere un'idea sull'avanzamento del progetto.
Le modifiche critiche del progetto devono essere accettate dal CCB (a sua volta composto dall'intero team di sviluppo), le altre possono essere fatte liberamente.
Questi report, insieme alle decisioni definite negli scrum meeting, rappresentano la principale strategia per valutare velocemente lo status del progetto, e, di conseguenza, sono utili a fare course-correction.
Difatti, è proprio in questo modo che è previsto bilanciare l'equilibrio tra requirements e l'impegno necessario per soddisfarli.

1.6 Risks

Il rischio principale è di non consegnare in tempo il progetto o di non consegnare un progetto perfettamente funzionante.

1.7 Staffing

I membri del team sono: Nicolas Anselmi, David Guzman Piedrahita e Marco Vinciguerra.
Per provare a vedere come funziona il mestiere il ruolo dello SCRUM master cambia a rotazione e si parte dalla settimana che inizia col 13 dicembre. Il primo SCRUM master sarà David, la settimana dopo Nicolas e quella dopo Marco Vinciguerra e così via...

1.8 Methods and techiniques

Per gestire gli sprint è stato utilizzato un template di Notion in quanto ha la possibilità di schedulare i task in base alla scadenza e in base ad un ordine gerarchico.
Per gestire la fase di testing si usa il tool better flutter tests che fa lui il testing sul framework Flutter. Il test viene scritto automaticamente dal tool e quindi non si applica fin da principio.
Per quanto riguarda la specifica dei requisiti si utilizza lo standard IEEE 830.
Si prevede l'uso di una strategia COTS per la scelta di diversi moduli o, più precisamente, widget di Flutter, che consentono di implementare velocemente elementi UI classici, senza scrivergli da zero.
Per quanto riguarda i test di Dart si utilizza Dart unit testing e si utilizza il TDD e per garantire la continuos integration si usa Travis CI che partirà ad ogni Git e ogni 24 ore.
Per i database si utilizza Firebase per creare e gestire il database delle prenotazioni.
La gestione delle modifiche viene svolta tenendo in conto le considerazione del punto 5 e, soprattutto, il punto 13.

1.9 Quality assurance

Per garantire la qualità del prodotto viene utilizzato lo standard ISO 9001.

1.10 Work packages

Alcuni dei sottoprogetti (work package) che sono stati definiti a priori.
In viste della natura agile del progetto, questi work-package saranno estesi e modificati o evoluti nelle diverse iterazioni del life-cycle:

- Fase di design di schemi UML
- Imparare ad utilizzare Flutter e dart
- Implementare l'applicazione con un'interfaccia grafica (front-end)
- Implementare la domain logic (back-end)
- Uso di un database
- Fare il testing sul prodotto
- Fare testing usando l'applicazioni in telefoni reali (non simulator)

1.11 Resources

Gli obiettivi di prototipazione proposti dal progetto in questione richiedono solo l'uso di computer adatti alla programmazione nei suddetti linguaggi e framework. Dopo diverse iterazioni di prototipazione è prevista la possibilità di usare dei cloud-server che ricevano e gestiscano le richieste degli utenti tramite i loro client/app.

1.12 Budget and schedule

Il tempo preventivato per il progetto è di circa 70 ore a testa, quindi in totale saranno richieste 210 ore.
La documentazione ha la priorità più alta, ma la necessità di imparare a utilizzare Flutter, di imparare a gestire UI per la prima volta, rappresentarà comunque un alto costo in termini di tempo.

1.13 Changes

Col tempo potrebbero cambiare le richieste da parte del cliente durante la fase di validazione di ogni processo.
Al supporto delle attività di change management vengono utilizzati i due tool usati anche per altri aspetti del progetto, ovvero:
Github, che in questo caso funge da piattaforma per accettare o rifiutare le modifiche version-oriented e per consentire di usare il forked develpment;
e Notion che, non essendo un tool specifico per lo sviluppo, serve invece a gestire l'organizzazione delle tempistiche e dei sotto-progetti a un livello di astrazione più alto.
I contenuti di questi tool, e i diversi report dei punti precedenti, servono come guida per la compilazione di un eventuale configuration management plan che conterrà una management section e activities. I documenti generati come risultato del punto 5 sono particolarmente utili per la management section.

1.14 Delivery

Il project plan verrà consegnato entro il 27 dicembre 2021
La consegna verrà fatta 5 giorni prima dell'esame orale.

2 Scrum life cycle

Il life cyle utilizzato è Scrum. Di conseguenza, saranno usate iterazioni con limiti di tempo predefinito, i cosiddetti sprint. La loro composizione e quantità è soggetta a modifiche. La durata tradizionale degli sprint e di 2 a 4 settimane, ma in questo caso si utilizzano sprint di durata da 5 a 10 giorni per velocizzare il processo.
Pre cercare di capire come funzionano i ruoli all'interno di Scrum, essi vengono cambiati a rotazione settimanale.

2.0.1 Product owner

è l'intermediario tra cliente e dipendenti e solitamente si occupa lui di gestire il backlog. In questo caso non c'è nessuno che si occupa di gestire l'interazione col cliente.

2.0.2 Development team

è il team che si occuopa di sviluppare il progetto. Quando si fa questo ruolo ci si occupa di svolgere i ruoli che sono inseriti nel backlog.

2.0.3 ScrumMaster

Quando uno degli elementi del gruppo diventa ScrumMaster si occupa di aiutare a risolvere i problemi del development team tramite consigli.

2.1 Sprints

2.1.1 State of the backlog

il backlog iniziale si basa completamente sui requisiti che sono disponibile nel corrispettivo documento, nella fase di architettura il backlog tiene considerazione dei suddetti requisiti ma anche del contesto architettonico (vedi documento architettura) .

2.2 Sprint 1

- Creazione della prima parte della UI
- Informarsi sul funzionamento dei database in flutter
- Proseguire con la documentazione
- Informarsi sul testing
- Unit testing

2.3 Sprint 2

- Avanzamento della UI
- Implementazione dell'interfaccia di Login
- Iniziare il testing
- Implementazione di un database relazionale
- Proseguire con la documentazione
- Widget testing

2.4 Sprint 3

- Perfionamento della UI
- Implementazione di un real time database
- Mappare l'interfaccia di Login col database
- Proseguire con la documentazione
- Continuos integration con Travis per il testinf

2.5 Sprint 4: database - experiments

- Perfezionamento del tema e delle migliore tecniche
- Build per l'interfaccia web
- Fine della documetnazione
- Sprint planning meeting
  26/12/21
  Sono stati definiti gli obiettivi base del progetto e dello sprint. Dal backlog (e, a sua volta, dai requirements) sono stati scelti i seguenti elementi per lo sprint backlog:
  - Creazione della documentazione dei requirements.
  - Creazione della documentazione di architettura.
  - Proposte iniziali di schemi UML (soggetti a molte modifiche dovuto al seguente punto).
  - Precisazione delle librerie e framework elements da usare per l'implementazione software.
  - Processo cots per la scelta delle suddette librerie e altri elementi di framework.
- Report degli Scrum meeting più significativi
  28/12/21
  Discussione sui primi schemi UML. Proposte per migliorarli e modificarli + Backlog refinement
  30/12/21
  Discussione su come gestire nel modo più efficiente la costruzione della UI + Backlog refinement
  2/1/22
  Discussione di diversi elementi modulari UI trovati nei giorni precedenti e che erano candidati per uso come UI ufficiale.
  5/1/22
  Discussione sul come costruire la home page dell'applicazione + Retrospective meeting notes
  6/1/22
  Cross-review dei diversi file di documentazione creati (requirements, architecture, testing, ecc)
  12/1/22
  Scrum per gestire la UI
  15/1/22
  Scrum per discutere lato backend e Firebase, abbandonata l'idea di usare un database relazionale classico
  18/1/22
  Discussione su come utilizzare Firebase
  21/1/22
  Implementazione di Firebase + Sprint review meeting
  24/1/22
  Prova generale dell'applicazione
  27/1//22
  Discussione e revisione della documentazione + Retrospective meeting notes

3 People Management and Team Organization

La suddivisione e il funzionamento dello Scrum team è citato al paragrafo precedente.
Il team è formato da 3 persone e la comunicazione avviene tramite chiamate telefoniche.
Come approccio di organizzazione si usa l'approccio di Mintzberg, i punti principali sono: NON SO COSA SCRIVERE

- **simple structure**: Non c'è una persona del gruppo che comanda ma c'è un livello paritetico e la comunicazione è veloce e diretta
- **Machine bureaucracy**: Una volta stabiliti gli output non si interviene sull'andamento dei membri del team ma l'importante è raggiungere gli obiettivi
- **Divisionalized form**: La standardizzazione del processo produttivo la si ha gli standard di qualità del codice di flutter
- **Professional bureaucracy**: Non c'è formalità e nessun tipo di gerarchia
- **Adhocracy**: Tutti i componenti si occupano di fare un po' tutti i ruoli per quanto riguarda lo sviluppo del software, la gestione del database e dei casi di test

4 Software Quality

4.0.1 CMM

Per quanto riguarda il CMM (Capaibility Maturity Model) esistono 5 livelli di maturità del software, essi sono:

- Initial
- Repeatable
- Defined
- Quantitatively managed
- Optimizing

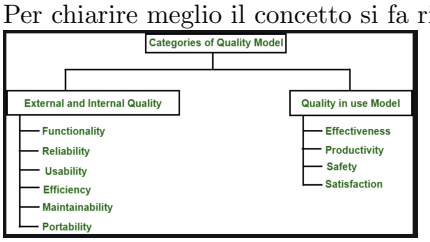
In questo progetto si punta ad utilizzare il terzo livello 3 (Defined) in cui ogni attività viene documentata e standardadizzata per l'intera organizzazione per quanto riguarda il processo per il design, development, testing e via dicendo.

4.0.2 ISO 9216

Per quanto riguarda la product conformance si fa riferimento al seguente link: `https://www.geeksforgeeks.org/iso-iec-9126-in-software-engineering/`
Esso si basa sui 4 seguenti principi:

- **Part 1: "Quality model"**
- **Part 2: "External metrics"**
- **Part 3: "Internal metrics"**
- **Part 4: "Quality in use metrics"**

Per chiarire meglio il concetto si fa riferimento al seguente grafico:



4.1 External and internal quality

Durante la fase di sviluppo del software il focus principale rimane sempre il mantenimento della qualità. Per fare tutto questo bisogna tenere conto delle seguenti 6 caratteristiche.

- **Functionality**
- **Reliability**
- **Usability**
- **Efficiency**
- **Maintanability**
- **Portability**

4.1.1 Functionality

Il codice si occupa di mantenere il giusto grado di sicurezza dei dati e deve essere adeguato e facile da utilizzare per gli utenti.

4.1.2 Reliability

Il codice si occupa della gestione dei guasti e punta ad aver il minor numero possibile di errori.

4.1.3 Usability

Il prodotto finale non deve essere difficile da utilizzare e la user experience deve essere il più semplice possibile per l'utente in quanto non deve essere necessariamente esperto di coding. Ad esempio i bottoni sono studiati per essere il più semplici e intuitivi possibile.
Dal punto di vista delle prestazioni deve essere il più reattivo possibile e facile da capire.

4.1.4 Efficiency

Si vuole creare il programma più efficiente possibile che utilizzi il minimo delle risorse del dispositivo da cui si utilizza l'applicazione.
Per migliorare l'efficienza si è usato per esempio la variabile const nella UI.

4.1.5 Maintanability

Il codice deve essere scritto nel modo più conforme alle regole di standard di programmazione per favorire la leggibilità e la successiva manutenzione.

4.1.6 Portability

Il programma deve funziona su più piattaforme possibilili, Flutter permette con un solo codice sorgente di scrivere programmi per più piattaforme.

4.1.7 Security

I dati vengono criptati tramite la piattaforma encrypt.

4.2 Quality in use Model

Si basa sulle seguenti 4 caratteristiche che si vuole soddisfare:

- Effectiveness
- Productivity
- Safety
- Satisfaction

4.3 Introduzione alla Taxonomy quality

Questo file si occupa del descrivere i requisiti tassonomici di qualità del progetto.
Per quanto riguarda la definizione di McCall.
In particolare si vuole utiizare i seguenti driver/linee guifa per la produzione, revisione e transizione del codice.

4.4 Product operation

- Correctness: se il sistema fa quello richiesto
- Reliability: se il sistema è abbastanza accurato
- Efficiency: se il sistema utilizza l'hardware efficientemente
- Integrity: se il sistema è sicuro
- Usability: se il sistema è utilizzabile

In particolare ci si sofferma sull'usability, correctness e reliability in quanto il prodotto deve funzionare il meglio possibile.

4.5 Product revision

- Maintainability: se il sistema in caso di guasto è riparabile
- Testability: se il sistema è testabile
- Flexibility: se il sistema è facilmente cambiabile

Il prodotto che si sta costruendo in questo caso tende a essere molto testabile in quanto i test per verificare la correttezza vengono creati ed eseguiti poco dopo la creazione delle classi.

4.6 Product transition

- Portability: se il software è utilizzabile su altre piattaforme
- Reusability: se il software è riutilizzabile
- Interoperability: se il sistema è interfacciabile con altri sistemi

Il sistema tenderà a essere molto portabile in quanto è stato fatto con Flutter, il quale tende ad essere molto scalabile.

4.7 Qualità per il codice in Dart

Per il codice in Dart è stato implementato il Dart Analyzer, che si ottiene modificando il file $analysis_options.yaml$ e tramite il comando da terminale dart analyze si fa un test per vedere se le qualità prese in considerazione sono valide e rispettate.
La documentazione necessaria per informarsi su questo tipo di standard è stata presa dal seguente link: `https://pub.dev/packages/analyzer`.
Alcuni tra i parametri presi in considerazione per garantire la qualità sono:

- Maximum nesting level
- Ciclomatic complexity
- Number of parameters
- Source lines of code

Per usare i quality metrics si usano i segueti comandi digitati da terminale:

- Miglioramento delle performance del codice: dart_analyze
- Calcolo complessità: flutter pub run dart_code_metrics:metrics analyze lib
- File non utilizzati: flutter pub run dart_code_metrics:metrics check-unused-files lib

La configurazione è presente nel file analysis_options.yaml.

```yaml
dart_code_metrics:
  anti-patterns:
    - long-method
    - long-parameter-list
  metrics:
    cyclomatic-complexity: 20
    maximum-nesting-level: 5
    number-of-parameters: 4
    source-lines-of-code: 50
  metrics-exclude:
    - test/**
```

Ecco un esempio di output di un test fatto con dart analyze:

```
info • lib/home_page.dart:11:7 • Use key in widget constructors. • use_key_in_widget_constructors
info • lib/home_page.dart:30:13 • Prefer const with constant constructors. • prefer_const_constructors
info • lib/home_page.dart:33:17 • Avoid `print` calls in production code. • avoid_print
info • lib/home_page.dart:35:17 • Avoid `print` calls in production code. • avoid_print
info • lib/home_page.dart:65:13 • Prefer const with constant constructors. • prefer_const_constructors
info • lib/home_page.dart:89:13 • Prefer const with constant constructors. • prefer_const_constructors
info • lib/main.dart:2:8 • Unused import: 'package:hair2/Model/Entity/clientBookings.dart'. Try removing the
       import directive. • unused_import
info • lib/main.dart:4:8 • Unused import: 'package:hair2/home_page.dart'. Try removing the import directive. •
       unused_import
info • lib/main.dart:5:8 • Unused import: 'package:hair2/sign_in_page.dart'. Try removing the import directive. •
       unused_import
info • lib/main.dart:21:7 • Use key in widget constructors. • use_key_in_widget_constructors
info • lib/main.dart:49:7 • Use key in widget constructors. • use_key_in_widget_constructors
info • lib/main.dart:55:14 • Prefer const with constant constructors. • prefer_const_constructors
info • lib/mainUI.dart:1:1 • Name source files using `lowercase_with_underscores`. • file_names
info • lib/mainUI.dart:2:8 • Unused import: 'package:hair2/components/PrenotazioneClienti.dart'. Try removing the
       import directive. • unused_import
info • lib/mainUI.dart:3:8 • Unused import: 'components/Login.dart'. Try removing the import directive. •
       unused_import
info • lib/mainUI.dart:4:8 • Unused import: 'components/Gestore.dart'. Try removing the import directive. •
       unused_import
info • lib/mainUI.dart:6:8 • Unused import: 'components/TuttePrenotazioniCliente.dart'. Try removing the import
       directive. • unused_import
info • lib/mainUI.dart:7:8 • Duplicate import. Try removing all but one import of the library. •
       duplicate_import
info • lib/mainUI.dart:7:8 • Unused import: 'components/Gestore.dart'. Try removing the import directive. •
       unused_import
info • lib/mainUI.dart:8:8 • Unused import: 'components/NextPage.dart'. Try removing the import directive. •
       unused_import
info • lib/mainUI.dart:9:8 • Unused import: 'components/Buffer.dart'. Try removing the import directive. •
       unused_import
info • lib/mainUI.dart:10:8 • Unused import: 'components/Profilo.dart'. Try removing the import directive. •
       unused_import
info • lib/mainUI.dart:11:8 • Unused import: 'components/Settings.dart'. Try removing the import directive. •
       unused_import
info • lib/mainUI.dart:12:8 • Unused import: 'components/Changesetting.dart'. Try removing the import directive.
       • unused_import
info • lib/mainUI.dart:13:8 • Duplicate import. Try removing all but one import of the library. •
       duplicate_import
info • lib/mainUI.dart:13:8 • Unused import: 'components/Settings.dart'. Try removing the import directive. •
       unused_import
info • lib/mainUI.dart:14:8 • Duplicate import. Try removing all but one import of the library. •
       duplicate_import
```

Col tempo questi consigli sono diminuiti di circa l'80 % e il 20% riguarda istruzioni che sono deprecated ma che sono necessarie per il progetto.
Per migliorare l'efficienza si è usato per esempio la variabile const nella UI che toglie un errore in dart_analyze.

5 Requirement Engeneering-IEE830

5.1 Purpose

Questo documento segue la struttura proposta dallo standard IEEE 830 per la definizione dei requirements. Questi rappresenta uno dei criteri fondamentali per valutare l'adeguatezza dell'implementazione ottenuta nei diversi sprint e un punto di partenza per le seguenti fasi dello sviluppo dell'applicazione.

5.2 Scope

L'obiettivo centrale è costruire un'applicazione per la gestione di prenotazioni di tagli in saloni di bellezza, eventualmente estendibile ad altri servizi.
In grandi linee, l'applicazione dovrebbe consentire di facilitare il processo della creazione di appuntamenti, evitando l'uso di chiamate, sostituendole con procedure online molto più veloci e friction-less.
L'interfacciamento dell'utente con l'applicazione deve essere tale che, nonostante la più alta complessità inerente a una soluzione model-view-controller rispetto all'uso di semplici chiamate, l'utilizzatore percepisca un netto miglioramento rispetto al solito modo di fare e a prescindere delle loro conoscenze informatiche.
Di conseguenza, l'applicazione non solo deve offrire la funzionalità centrale delle prenotazioni, ma in più deve farlo in un modo intuitivo.

5.3 Definitions, acronyms and abbreviations

5.4 References

5.5 Overview

Dopo questa introduzione, il documento è composto da due ulteriori fasi: la parte 2, che da una descrizione globale dei requirements, e la parte 3, che elenca e classifica tutti i diversi requirements individuali, usando il modello Kano e Moscow.

5.6 Overall description

5.7 Product perspective

Non ci sono database o altre strutture informatiche preesistenti, in quanto il pubblico di destinazione è composto proprio da saloni di bellezza che gestiscono le loro prenotazioni in modo informale. La costruzione del software deve dunque partire da zero.
Per la fase di elicitation sono state usate le strategie di open-ended interview, task analysis derivante anche da esperienze personali e natural language descriptions.
Le successive fasi di V and V e di negotiation saranno svolte dopo la generazione di un primo prototipo.

5.8 Product functions

A continuazione sono elencati le funzionalità centrali, l'elenco dei requirements a una granularità più precisa sono disponibili nella parte 3.

- Possibilità di prenotare tagli e, eventualmente, altri servizi offerti dai saloni.
- Possibilità di visualizzare le prenotazioni future e passate associate al salone in questione.
- Possibilità di tener traccia dei clienti con degli appositi profili utente/cliente associati a informazioni rilevanti.
- Disponibilità di visualizzare l'andamento dei ricavi risultanti dalle prenotazioni.

5.9 User characteristics

Come detto sopra, il prodotto deve essere costruito per soddisfare le esigenze di un pubblico di destinazione con conoscenze tecniche basilari (uso di smartphone e applicazioni user-friendly).
L'UI dell'applicazione deve essere facile da usare da utenti ormai familiarizzatisi con altre applicazioni molto popolari (YouTube, Instagram...), di conseguenza il design-language deve essere coerente con quello al quale i potenziali utenti si sono ormai abituati, riducendo, di conseguenza, la learning-curve per usare il softwarele

5.10 Constraints

Nella sua versione finale, l'applicazione dovrebbe essere utilizzabile sia da cliente che prenotano che da gestori di saloni di bellezza.
I clienti dei saloni non devono avere accesso a informazioni del salone, come le prenotazioni di altri utenti, o i ricavi del salone. Dall'altro canto, i gestori non possono modificare certe informazioni dei profili degli utenti, ma possono modificare dati associati alle prenotazioni.
Per il prototyping iniziale, si deve dare priorità alle funzionalità offerte ai gestori. L'implementazione delle prenotazioni remote fatte direttamente dagli utenti ha una priorità secondaria nelle prime fasi.

5.11 Assumptions and dependencies

I sistemi usati sono: Firebase + Flutter login.

5.12 Requisiti non funzionali

Il sistema deve avere un tempo di risposta minimo, velocità di utilizzo, dati accessibili solo agli utenti specifici e quindi i seguenti requisiti non funzionali:

- Prestazioni
- Sicurezza
- Manutenibilità
- Usabilità
- Affidabilità
- Disponibilità

5.13 Specific requirements

tra parentesi va scritta la priorità nel seguente modo: (Kano, Moscow)

- Possibilità di scegliere i diversi tipi di acconciature e in base al tipo di acconciatura scegliere la durata della prenotazione (Attactive, Must have)
- Possibilità di registrarsi per la prima volta al sito da parte di un cliente. (Must-be, should have)
- Utilizzo di un database per la gestione dei clienti. In alternativa si potrebbero utilizzare variabili per gestire le prenotazioni. (Must-be, should have)
- Creazione di un profilo base utente per le prenotazioni. (Must-be, should have)
- Che il sistema dellle prenotazioni non funzioni correttamente e che si accavallino sulla stessa fascia oraria. (One-Dimensional, should have)
- Possibilità di mettere recensioni da parte dell'utente. (Questionable, could have).

5.14 COTS: scelte di framework e tecnologie esterne

Per soddisfare i requirements funzionali (e non) proposti nel rispettivo documento sono state adottate una serie di strategie standard.
In particolare, è necessario creare delle strutture dati per offrire la funzionalità di effettuare e visualizzare prenotazioni, sia dal punto di vista dal cliente che dal parrucchiere. In più, visto che tutti gli utenti possono essere sia parrucchieri che clienti, entrambe funzionalità devono essere disponibili per tutti gli utenti con un basso overhead di transizione da una modalità all'altra.
Di conseguenza, la scelta delle tecnologie utilizzate deve essere guidata da questi criteri di qualità:

- Flutter: Viene usato il framework Flutter di app-development proposto da Google. Rispetto ad altre alternative come React o Swift, si adatta meglio al nostro requirement di portability, in quanto serve un unico code-base per generare l'applicazione per più piattaforme.

- Firebase Realtime Database: L'uso di un database online è fondamentale per il funzionamento di base. Sono state valutate alternative SQL che si adeguavano meglio al skillset preesistente del team, ma servizi noSQL come firebase offrono database gratuiti e molto veloci, rispettano i requirement non funzionale della responsiveness dell'app. In più, l'integrazione con Flutter è facilitata dal fatto che sono entrambi servizi offerti da Google.
- Firebase Authetication: per i servizi di creazioni di account e log in, Firebase continua a essere la scelta migliore per i motivi citati sopra.

6 Software Architecture

6.1 Architecture design metod

Per derivare una prima architettura funzionante partendo dai requirements specificati nell'apposito documento, si segue il criterio Generalized Model di Hofmeister et al. (2007).
Questo architecture design method è particolarmente utile per il progetto in questione, in quanto è strutturato utilizzando la terminologia e i ragionamenti di SCRUM, che è il life-cycle agile usato per l'applicazione e il suo sviluppo.
In particolare, usa lo stesso concetto di backlog per elencare i diversi problemi che devono essere gestiti, e da esso è possibile scegliere gli elementi ritenuti più importanti in ogni particolare iterazione.
Il backlog ha in input i requirements del progetto, con particolare attenzione ai requisiti di qualità; il contesto, che contiene idee aggiuntive sul come implementare un sistema che soddisfi i requisiti e, infine, i risultati della valutazione dell'architettura dell'iterazione precedente.
Nel progetto verranno presi in considerazioni questi tre fattori per determinare le scelte di design.

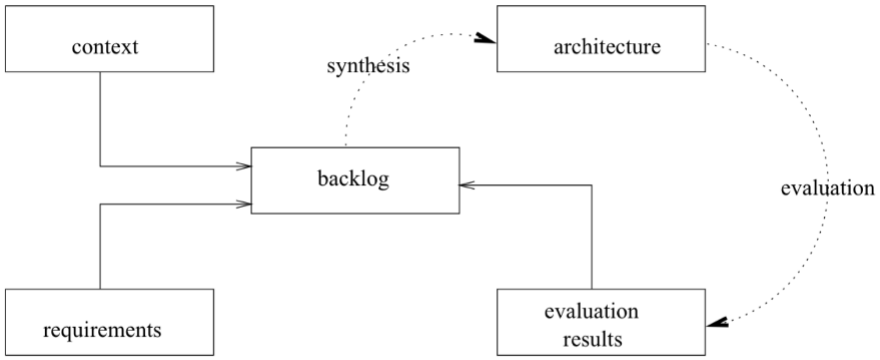


Figure 11.2 Global workflow in architecture design

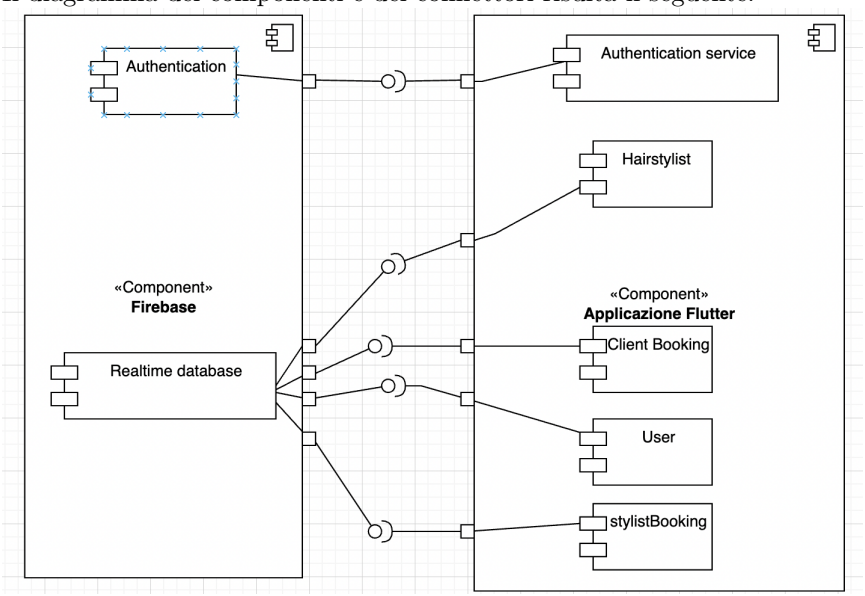
6.2 Decisioni di design

6.2.1 Decision 1

- Issue
- Decision
- Status
- Assumptions
- Alternatives
- Rationale
- Implications
- Notes

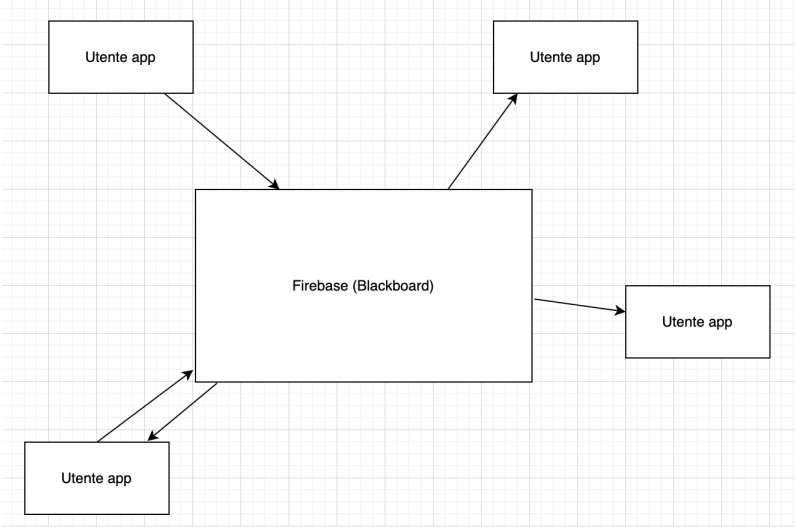
E' stato creato nei diagrammi UML diverse versioni dello stesso tipo di diagramma a seconda dello stakeholder con cui ci si sta confrontando.

Il diagramma dei componenti e dei connettori risulta il seguente:



6.3 Stile di architettura

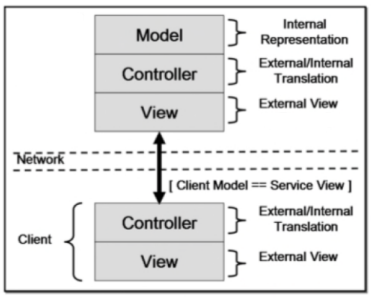
Lo stile architetturale dell'applicazione può essere assimilabile al repository style, in quanto ci sono due blocchi fondamentali: i client, composti del codice Dart/Flutter e disponibile come applicativo mobile, e il database, implementato usando i servizi non relazionali di Firebase Realtime Database offerti da Google.



Distinguiamo quindi come:

- System model: un sistema centralizzato di informazione strutturata che deve essere disponibile in scrittura e lettura per i diversi client, che sono invece gli elementi computazionali che sono indipendenti da esso.
- Components: si tratta di un unico componente memory (il database) e una quantità indefinita di client (a seconda del quantitativo di utenti) che sono invece dei component di tipo manager, in quanto mantengo, per un certo periodo di tempo, informazioni di stato e offrono diverse operation o funzionalità.
- Connectors: chiamate sincrone e asincrone http tramite il web per contattare Firebase, comunicando tramite file json.

Se, in più, teniamo in considerazione il ruolo centrale dei servizi offerti da Firebase, i quali non si limitano soltanto al database ma anche alle funzionalità di login e authentication gestite separatamente, possiamo definire l'architettura dell'app come una a servizi.



Dove si segue un modello basato sul classico model-view-controller, ma dove il model del client, al posto di essere implementato e gestito in locale, si appoggia a servizi esterni e standardizzati.

- Controller: il controller, a differenza del model, è implementato come parte del client vero e proprio, ed è stato costruito riflettendo, per quanto possibile, le diverse entità del mondo reale che interagiscono con l'applicazione o che rappresentano tipologie di dati pertinenti. La sua composizione e descritta per esteso nel class diagram.
- View: interamente costruita utilizzando i servizi offerti dal framework flutter, dove tutti i diversi tasti e pagine sono rappresentate da cosiddetti widget, che a usa volta sono delle classi dart. Dovuto all'elevato numero di classi che ne risulta e considerando che sono perlopiù assenti di metodi e campi nel senso tradizionale, non sono stati rappresentati nel class diagram.

6.4 Rappresentazione architettura con IEEE 1471

Le rappresentazioni più concrete dell'architettura dell'applicativo vengono descritte attraverso degli schemi UML. Questi possono essere categorizzati come delle View indirizzate a diversi Stakeholder e definite tramite diversi Viewpoint, come formalizzato dallo standard IEEE 1471.
In particolare, i Viewpoint utilizzati per definire le View sono stati scelti tra quelli proposti da Bass et al. (2003).

6.4.1 Module View

I Viewpoint nella categoria module costituiscono una rappresentazione statica del sistema. Per questo progetto la module view più importante è Class.
Le view costruite seguendo le specifiche del viewpoint Class descrivono il sistema in termini delle relazioni di eredità degli elementi. Il class diagram UML del progetto mette a disposizione questa informazione, assieme ad altre precisazioni dovute alla natura object-oriented del linguaggio di programmazione.
Add image

6.4.2 Component and connector View

Il tipo di viewpoint scelto in questa categoria è il process viewpoint. Questo definisce il sistema in termini di comunicazione e sincronizzazione di processi.
Come tutti gli altri viewpoint in questa categoria, offre una descrizione dinamica del software e, nel caso particolare del process viewpoint, è particolarmente utile per valutare performance e availability del sistema.
In termini di UML, questa informazione è disponibile principalmente tramite il Sequence diagram. Questo schema spiega come i diversi elementi (classi) del sistema comunicano tra di loro attraverso procedure calls **(di tipo sincrono e asincrono)**.
Il component è la memory, il connector è la memory (database).

6.4.3 Allocation View

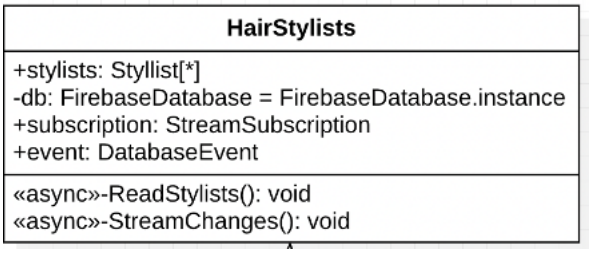
Questa categoria di viewpoint descrive la relazione tra il sistema e il mondo che lo circonda. Di conseguenza, può essere utilizzato per definire come viene assegnato hardware al software o come quest'ultimo viene mappato al file system.
Per il progetto in questione, questo tipo di problematiche vengono gestite automaticamente dai framework utilizzati e quindi non sono competenza degli stakeholder che altrimenti ne farebbe uso, come i programmatori e i maintaner.
I viewpoint di tipo work assignment potrebbero essere utile per rappresentare graficamente la distribuzione del lavoro, ma in questo sprint vengono saltati per brevità.

7 Software Design

Per informazione sulla complessità e altri criteri di qualità, fare riferimento al documento della qualità.

7.1 Funzionamento delle classi Dart, in conformità con i requisiti funzionali e non:

Come evidenziato dall'UML Class Diagram, le classi che si interfacciano con i servizi di Firebase sono state modellate partendo da entità del mondo reale, rappresentando quindi diversi ruoli e azioni previste nell'utilizzo dell'app. In particolare, un fattore che ha guidato il loro design è stata la necessità di gestire dati che non sono disponibili in locale con altri che invece lo sono. Tutto questo deve essere invisibile dal punto di vista dell'utente, usando delle loading screen nel caso peggiore.
Di conseguenza, le classi che raggruppano informazioni disponibili nel database sono tutte munite da un collegamento diretto a esso tramite un attributo che è un oggetto speciale offerto dalle librerie di Firebase.
Ad esempio, la classe HairStylists, che ha il compito di essere un registro di tutti i parrucchieri che offrono i loro servizi ad ogni singolo istante, ha il campo privato db di tipo FirebaseDatabase, che di default viene inizializzato con l'oggetto che consente alla classe di comunicare con il database a un alto livello di astrazione. In più questa classe ha una particolarità: per consentire di avere un elenco di parrucchieri dinamico, e che quindi si tiene costantemente aggiornato rispetto al database, la classe conta con un metodo StreamChanges, eseguito nel costruttore, che crea una Subscription al database, dove la classe diventa un listener di tutte le modifiche del sotto-albero del json che contiene l'informazione pertinente.
In alternativa, è stato i implemento il metodo ReadStylists, che legge il contenuto del database un'unica volta, e quindi senza creare una Subscription; non è in uso nell'implementazione più recente, ma resta disponibile qualora ci fossero problematiche nel funzionamento di StreamChanges.



7.2 Uso di strategie standard del linguaggio Dart e il Framework Flutter:

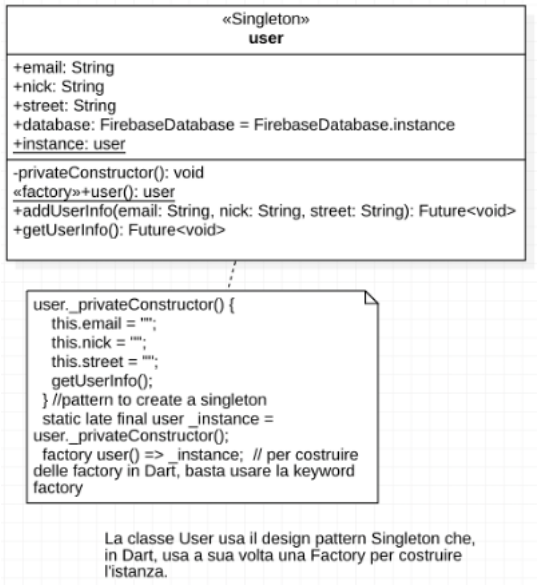
- - Provider: usare flutter implica che l'interfaccia grafica è strutturata come un albero di 'widget', ovvero tasti e altri elementi interattivi. Spesso è necessario propagare informazione che è stata ottenuta in un punto dell'albero ad altri punti dell'albero (e.g quando si passa da una schermata alla prossima), pertanto esistono diverse strategie soddisfare questa necessità.
  Per questo progetto, si usano i cosiddetti Provider, i quali sono la migliore alternativa perché consentono di rendere oggetti che sono in un nodo dell'albero raggiungibili per tutti i suoi figli (al posto di solo passare dei parametri, e di dover fare il hard-coding di questo passaggio di parametri in ogni transizione di nodo a nodo). Le classi che ne fanno uso devono estendere la classe ChangeNotifier, e questo è proprio il caso delle summenzionate classi che contattano il database, in quanto i loro dati possono essere necessari in diversi punti dell'UI.

- - Futures: Il linguaggio di programmazione Dart offre un tipo di oggetto che facilita la gestione delle chiamate sincrone e asincrone di dati, anche nel caso di informazioni che devono essere reperite dall'esterno, come nel caso del Realtime Database di Firebase.
  I metodi che interagiscono con oggetti di tipo Future sono identificati nel class diagram e nel codice con lo stereotype async, il quale sta a indicare che la classe scambia dati in modo asincrono, ma anche possibilmente in modo sincrono, quando dentro il corpo del metodo si usa la parola chiave await.

7.3 Uso di design pattern usati sia in dart che in altri linguaggi:

- - Singleton e factory: per gestire le informazioni dell'utente che, come nel caso di altre informazioni, può essere disponibile sia in locale che nei server, si usa una classe singleton di nome User che pertanto può essere istanziata una sola volta. Questo vincolo è abbastanza naturale per questo tipo di informazione, dato che il suo contenuto si limita a una sola occorrenza per ogni singolo client.
  Inoltre, per quanto riguarda Dart, è common practice usare delle Factory per la gestione dei singleton. La

creazione delle Factory è semplificata in quanto basta aggiungere al costruttore la parola chiave Factory.



- - Abstraction occurrence: questo design pattern è utilizzato nel caso dei singoli parrucchieri che vengono raggruppati nella classe HairStylists o nel caso delle prenotazioni e le classi per raggruppare le prenotazioni (StylistsBookings e ClientBookings), dove la relazione è composta da una classe che funge da Registro di tutte le occorrenze vere e proprie di una entità del mondo reale, che è rappresentata dalla seconda classe nella relazione.
- - Façade: La classe AuthenticationService e la classe DatabaseService sono delle façade in quanto hanno il compito di offrire tutta una serie funzionalità (per il login e per gestione del database rispettivamente) usando i metodi di un'unica classe. I metodi offerti da queste classi si interfacciano direttamente con i servizi offerti di firebase, per i quali non sono noti i class diagram, di conseguenza queste associazioni non vengono descritte esplicitamente nel class diagram.
- - Observer: la dinamica Observer-Observable è utilizzata con il meccanismo dei provider, che offrono dei metodi per rendere alcuni classi Observable e poter accedere a diversi valori e metodi e soprattutto ricevere notifiche di modifiche dal punto di vista di altre classi che diventano invece degli observer.

7.4 Design methods

Per sviluppare il design dell'applicazione si seguono le best practice e i design pattern consigliati per applicazioni dart con il framework Futter e, in più, considerando che si tratta di un software che usa un linguaggio di programmazione ad oggetti, si segue approssimativamente il design Method di Booch nel quale si parte per identificare le classi e oggetti, si procede a identificare il loro comportamento e attributi, successivamente si identificano le relazioni tra di loro e infine si fanno dei raffinamenti per procedere con una nuova iterazione del design.
Nella pratica, l'applicazione di Booch al design specifico di questo applicativo coinvolge piccole fasi di implementazione dopo ogni singolo step, per informare meglio le scelte di design e valutare se le scelte fatte precedentemente sono fattibili nel contesto di Flutter e tenendo in conto le conoscenze iniziali limitate del team di development.

8 Software Testing

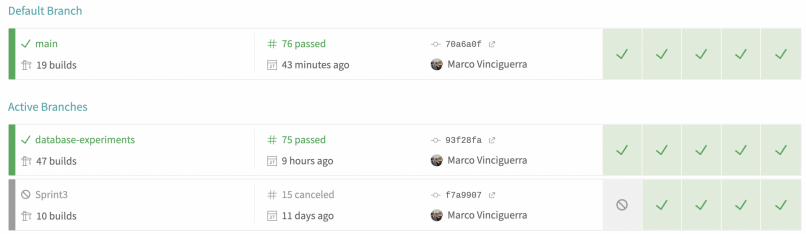
8.0.1 IEEE928

Questo tipo di documento si occupa di specificare e documentare come avviene la fase di testing del progetto. Questo documento rappresenta anche il test plan. Siccome stiamo facendo un tipo di sviluppo di software AGILE i test non vengono fatti alla fine ma vengono svolti assieme allo sviluppo delle classi.
Viene svolto un tipo di sviluppo di tipo TDD in cui si scrivono le classi dei test in contemporanea (se non prima) delle classi che servono per il sito. Le fasi del testing sono le seguenti:

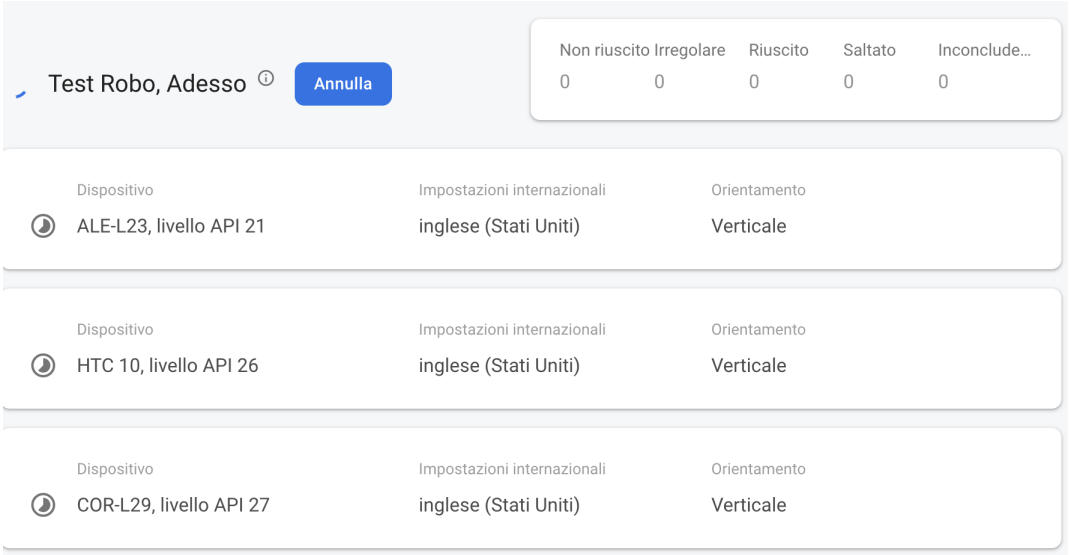
- Preparation of tests
- Running the tests
- Completion of testing

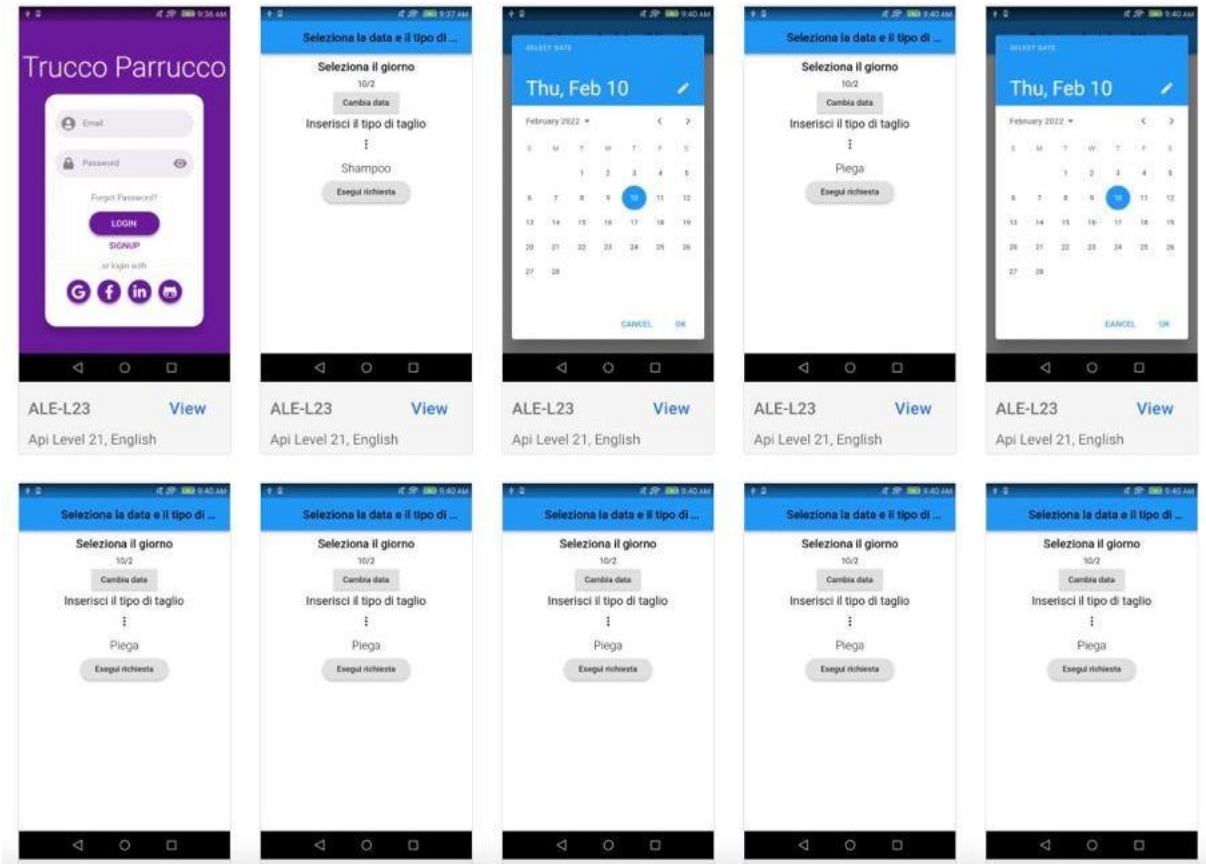
8.0.2 Preparation of tests

I test vengono fatti in contemporanea allo sviluppo delle classi, talvolta per alcune classi sono stati fatti prima i test pensando poi alle classi che sarebbero state implementate successivamente. I test per le classi in dart vengono scritti anche loro in linguaggio dart ed eseguiti tramite Junit.
Per quanto riguarda il testing di flutter si utilizza la funzione testWidgets che si importa dal pacchetto $flutter_test$ e permette di fare i test sulla web app.
Per garantire la CI (continuos integration) si utilizzerà Travis CI con i test che partiranno ad ogni git e ogni 24 ore.
I test tenderanno ad essere il più possibile di tipo coverage in quanto si cercerà di coprire il più possibile i casi di test.
I test di Travis partono ogni 24 h sul main e ogni git su un branch specifico porta all'esecuzione del test in background su Travis. Ad ogni test viene fatta in automatico anche una build che permette di vedere se ci sono eventuali errori diortografia all'interno del codice.
L'unico problema di Travis CI è che flutter non è supportato e non gestisce gli import correttamente e quindi è stato necessario importare (copiando) le classi che servivano per il testing.
L'interfaccia di Travis è la seguente:



E' stato fatto anche il firrebase testing per vedere se l'applicazione funziona correttamente sui dispositivi mobile. In questo tipo di test sono stati presi diversi tipi di cellulari e installato in modo automatico l'apk e l'applicazione è stata testata per vedere se girasse in modo corretto.





All'interno del progetto flutter vengono utilizzate 2 classi differenti che si occupano dello sviluppo del testing, essi sono:

- *UnitTest.dart*: per fare unit testing
- *WidgetTest.dart*: per fare i test delle widget

8.0.3 Running the tests

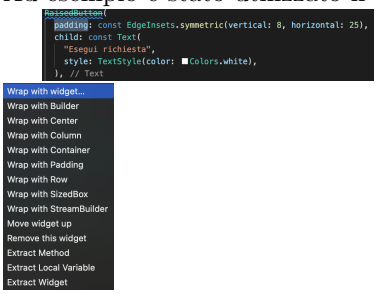
Una volta fatte le classi ed eseguiti i test si procede in modo iterativo a modificare il codice finchè i test non danno tutti risultati positivi.

8.0.4 Completion of testing

I dati expected sono inventati e ad ogni iterazione si commentano i risultati per capire se va tutto bene e se c'è qualcosa che non quadra.

9 Software Maintenance

Siccome non sapevamo fin da principio come utilizzare Flutter e firebase abbiamo alternato momenti di forwading engeneering con momenti di reverese engeneering in cui si è sistemata la documentazione in funzione del codice che è stato scritto (redocumentation) facendo così un processo di round trip engineering.
Il refactoring è stato fatto alla fine per separare le classi di dart per la view e per il model in delle cartelle apposite.
Per quanto riguarda il refactoring sono stati utilizzati i tool che Visual code per eseguire il refactoring (come ad esempio la possibilità di wrappare le widget o cambiare a cascata il nome di una classe).
Alla fine del progetto sono state rimosse anche le classi e i metodi non utilizzati (dead code).
Ad esempio è stato utilizzato il wrapp per creare il container di un widget in dart nel seguente modo:



Un esempio di refactoring che è stato fatto è quello di aggiungere la parola const davanti agli elementi dei widget per aumentare le prestazioni