TecnoCampus Escola Superior Politècnica

Centre adscrit a la upf. Universitat Pompeu Fabra Barcelona

Grau en Enginyeria Informàtica de Gestió i Sistemes d'Informació

GOLF TESTING

Memòria

Judith Barberán Martin

Tutor: Dr. David Ródenas Picó

Curs 2021/2022

TecnoCampus Mataró-Maresme

Agraïments

En primer lloc, agraeixo tota l'ajuda i consells que m'ha donat el meu tutor Dr. David Ródenas durant tot aquest temps.

També m'agradaria agrair a la meva família i a la meva parella tot el suport i paciència que han tingut durant aquesta etapa.

Abstract

The objective of this project is to develop a web application oriented to the learning of the Test Driven Development (TDD) methodology for both companies and freelancers. This web application has been developed based on the Behaviour Driven Development (BDD) methodology. A frontend has been developed using React Redux, a backend using Spring framework and Heroku has been chosen as the platform to deploy the application.

Resum

L'objectiu d'aquest projecte és desenvolupar una aplicació web orientada a l'aprenentatge de la metodologia Test Driven Development (TDD) tant per empreses com autònoms. Aquesta aplicació web s'ha desenvolupat a partir de la metodologia Behaviour Driven Development (BDD). S'ha desenvolupat un frontend utilitzant React Redux, un backend utilitzant el framework Spring i s'ha escollit Heroku com a plataforma per desplegar l'aplicació.

Resumen

El objetivo de este proyecto es desarrollar una aplicación web orientada al aprendizaje de la metodología Test Driven Development (TDD) tanto para empresas como autónomos. Esta aplicación web se ha desarrollado a partir de la metodología de Behaviour Driven Development (BDD). Se ha desarrollado un frontend utilizando React Redux, un backend utilizando el framework Spring y se ha escogido a Heroku como plataforma para desplegar la aplicación.

Índex

Índex de figures

Índex de taules

Glossari de temes

TDD	Test-Driven Development és un procés de desenvolupament de software dirigit per proves de software.
BDD	Behavior-Driven Development és un procés de desenvolupament dirigit per comportament, que ha evolucionat des de TDD.
Kata	Exercici per aprendre TDD.
Waterfall	Metodologia que consisteix a desenvolupar un projecte de forma seqüencial, començant amb les fases d'anàlisi i disseny i acabant amb el testeig i posada en producció.
CI/CD	Pràctiques combinades d'integració continua i entrega continua.
PaaS	Platform as a Service, és un entorn de desenvolupament i implementació al núvol. Aquest entorn ofereix recursos que permeten lliurar des d'aplicacions senzilles basades en el núvol fins a aplicacions empresarials sofisticades habilitades per al núvol.

1. Introducció

Els sistemes d'informació són una part important i integral del nostre dia a dia i en estar desenvolupats per éssers humans, poden aparèixer errors inesperats.

Tradicionalment, les proves de software es plantegen com una fase de verificació després d'una fase de construcció. Aquesta tendència ha canviat durant els darrers anys davant de la imposició dels mètodes àgils. "*És una nova mentalitat, en la que l'assegurament de la qualitat ja no és una disciplina separada i diferent, sinó que està fusionada en el procés general de desenvolupament de software."* [1]

Una de les metodologies àgils de proves de software és el TDD.

La idea d'aquest projecte sorgeix per la mancança en el mercat d'una aplicació d'ensenyament de TDD. Es tracta d'una aplicació pensada per a empreses i autònoms que vulguin aprendre TDD de manera didàctica, resolent kates de diferents nivells.

L'aplicació ha d'oferir als usuaris la possibilitat de resoldre kates de diferents nivells de manera incremental. Un cop resolta la kata, es farà una avaluació de com ha sigut resolta per a incentivar a que es faci TDD correctament.

Per al desenvolupament de l'aplicació, es segueix la metodologia Àgil BDD, la qual sorgeix com a resposta a problemes a l'hora d'ensenyar TDD.

El projecte es divideix en iteracions seguint el model espiral. En cada iteració es fixen uns objectius, es fa un anàlisis de riscs, es desenvolupa i finalment s'analitza la iteració. Un cop acabada cada iteració, hi ha una nova versió funcional de l'aplicació.

2. Marc teòric

Actualment, els sistemes d'informació són una part important i integral del nostre dia a dia. Podem trobar sistemes d'informació en molts àmbits com: telèfons mòbils, ordinadors, rellotges, televisors... Tenint en compte que els sistemes d'informació estan desenvolupats per éssers humans, poden aparèixer errors. *"Si no s'ha identificat aquest defecte i les aplicacions s'executen, hi ha un alt risc que l'aplicació no faci el que hauria de fer o l'objecte pel qual va ser creada, és a dir es genera una fallada o desperfecte".* [2]

És un error comú pensar que el valor de les proves del software és assegurar la qualitat del producte. Aquest fet sorgeix de la relació que es va establir entre les proves del software i la qualitat amb waterfall. "E*l testing de software pot verificar la presència d'errors però no l'absència d'ells. – Edsger Dijkstra* "

Les proves del software aporten valor en tots els nivells del projecte [3], cadascun dels punts s'expliquen en el capítol Comparativa proves tradicionals i àgils.

- Enteniment comú entre tots els participants del projecte, assegurant-se que estiguin alineats.
- Mantenir el codi net i fàcil de canviar, aconseguint que es facin entregues ràpides i a temps.
- Llançar amb seguretat cada vegada més ràpidament.

El World Quality Report és l'únic informe mundial que analitza les tendències de les proves de software i d'enginyeria de qualitat. Aquest informe presentat des de l'any 2009 per Capgemini, fa un anàlisi dels avanços en diferents camps com: àgile, DevOps, IA, automatització d'entorns de prova, dades, seguretat i pressupostos. Aquests anàlisis es fan amb l'ajuda dels punts de vista de 1.750 CIOs i líders tecnològics sèniors de 32 països i 10 sectors. La finalitat d'aquest informe és demostrar la importància de la qualitat i de les mesures que s'han de prendre per assegurar-la en els diferents àmbits.

Fig. 2.1. Portada World Quality Report 2021. Font: World Quality Report, 2021.

"Estem veient un gran canvi cultural. Igual que en la nostra vida diària, on tots volen i esperen el compliment de comandes l'endemà o el mateix dia, en els negocis la gent vol que el seu codi s'entregui immediatament i llest per fer-lo servir. És per això que estem veient un moviment cap a les proves contínues, on el nostre objectiu és prevenir defectes en lloc de trobar-los. És una nova mentalitat, en la que l'assegurament de la qualitat ja no és una disciplina separada i diferent, sinó que està fusionada en el procés general de desenvolupament de software." [1]

A l'informe del 2020, s'assenyalava que l'adopció d'Agile estava evolucionant de manera constant en lloc de ser una gran revolució com s'esperava a causa de les circumstàncies pandèmiques i post-pandèmiques. Tot i això, a mesura que augmenta el teletreball i la vacunació, aquesta tendència ha començat a canviar, el mercat està començant a obrir-se al canvi. Algunes empreses s'estan tornant menys conservadores obrint camí Agile.

A l'informe del 2021, es fa una anàlisis de la importància que li donen les persones enquestades a alguns aspectes principals d'Agile.

La següent imatge *Fig. 2.2,* mostra el resultat de l'enquesta realitzada on cal destacar:

- La importància del stack tecnològic per les persones enquestades ha disminuït 16 punts respecte a l'any anterior, passant del 65% al 49%.
- Augment en la importància de les prioritats comercials i la cultura de l'organització.

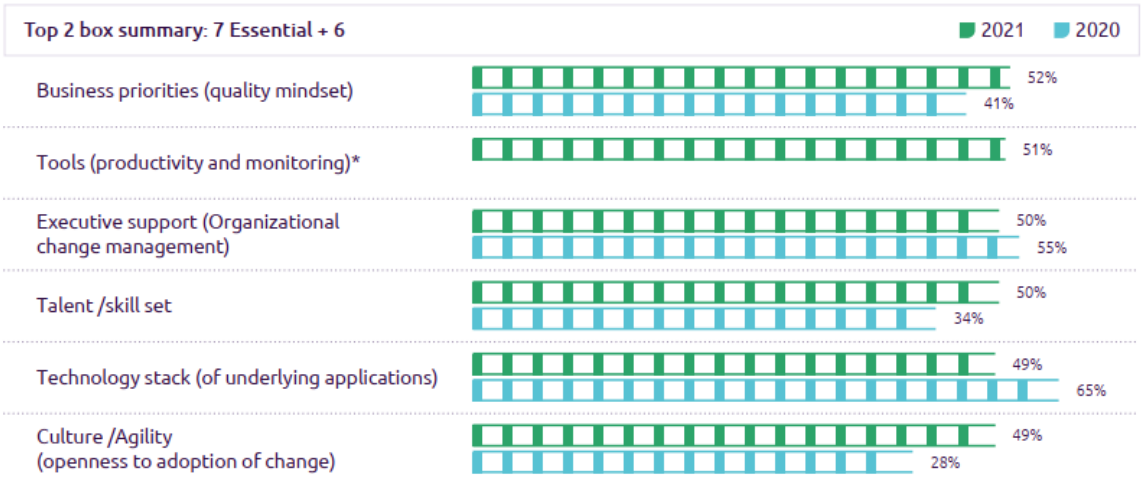

Fig. 2.2. Importància empreses a Agile. Font: World Quality Report, 2021

D'aquesta manera, es pot concloure que hi ha una alineació important de les empreses amb Agile. Les organitzacions estan reconeixent que les necessitats del negoci són més rellevants que la tecnologia subjacent, l'entorn o qualsevol altra cosa. A més reconeixen la necessitat de noves habilitats i d'un canvi de mentalitat.

2.1 Proves tradicionals

Les proves tradicionals, es plantegen com una fase de verificació després d'una fase de construcció. Primer es desenvolupa alguna cosa, i després, es verifica per assegurar-se que funciona. Generalment, s'assumeix que hi ha un pla mestre o una especificació per guiar tots els aspectes del desenvolupament. Aquest pla mestre o especificació és desenvolupat per l'analista, el client i l'arquitecte. [4]

La següent imatge *Fig. 2.3*, mostra gràficament com es desenvolupen les proves tradicionals i quins rols hi participen.

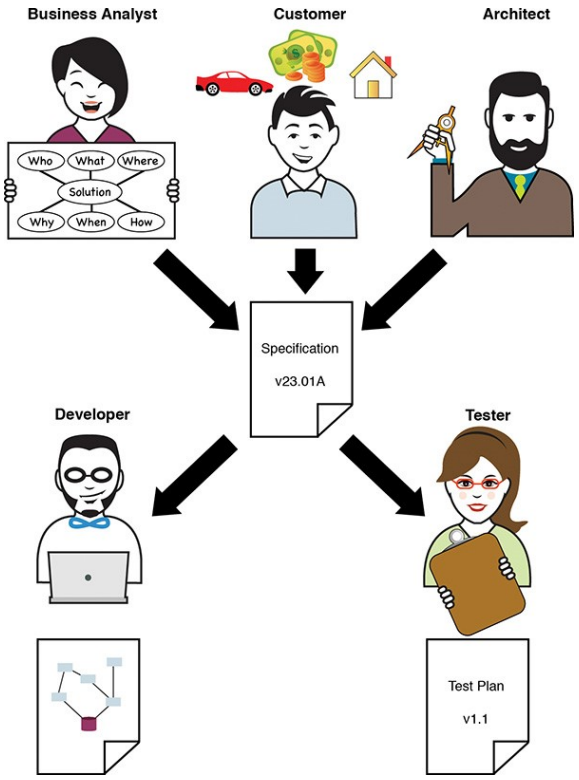

Fig. 2.3. Desenvolupament pla mestre. Font: Developer testing: Building quality into software, 2016

Aquesta metodologia té un clar inconvenient a causa de la divisió de treball. Existeix la possibilitat que els desenvolupadors i els testers desenvolupin una visió antagònica entre si en haver-hi poca comunicació entre els equips. Això pot portar al fet que, un cop acabat de desenvolupar tant el software com els tests, els models han divergit i hi ha una clara discrepància entre el software produït i els tests.

2.2 Proves àgils

Les proves àgils són proves que permeten el desenvolupament àgil. La idea principal és donar-li més valor al tester i augmentar la seva col·laboració dins de l'equip i amb el client. D'aquesta manera el rol de tester passa de ser reactiu a proactiu; deixa de desenvolupar test i esperar que s'executin, a contribuir en el llançament exitós del producte.

Aquesta metodologia s'enfoca en què cada membre de l'equip ofereixi un producte d'alta qualitat a través de la retroalimentació constant. Per fer-ho pot involucrar diferents pràctiques com:

- Integració continua
- TDD (Test Driven Development)
- BDD (Behaviour Driven Development)
- ATDD (Acceptance Test Driven Development)

En aquestes pràctiques, abans de començar a desenvolupar un requeriment, l'equip s'assegura que tots estiguin alineats. Aquesta alineació s'aconsegueix en reunions on participen els anomenats *three amigos:*

- **Client o analista del negoci**: encarregat de detallar cadascun dels requeriments comercials i assegurar-se que tots els membres de l'equip estiguin alineats.
- **El tester/ QA**: encarregat de discutir els casos de prova ja creats pels analistes del negoci i trobar casos extrems i escenaris que falten.
- **Desenvolupadors**: encarregats de discutir els casos de prova ja creats i de plantejar com es durà a terme el desenvolupament.

D'aquestes reunions s'obté una comprensió compartida, noves perspectives i el plantejament de nous dubtes. [4]

La següent imatge *Fig. 2.4*, mostra gràficament com es desenvolupen les proves àgils i quins rols hi participen.

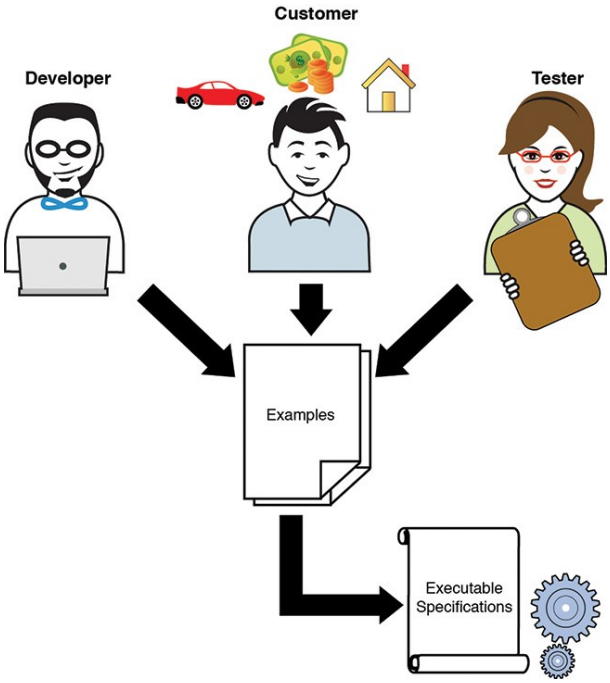

Fig. 2.4. Desenvolupament requeriments amb Agile. Font: Developer testing: Building quality into software, 2016

2.2.1 TDD

El TDD va ser publicat per Kent Beck en el llibre *TDD By Example,* l'any 2002. És una metodologia que forma part de la metodologia *extreme programming*. Tradicionalment, es desenvolupa el software i després s'escriu el test per comprovar el funcionament d'aquest, però TDD inverteix l'ordre. És a dir, primer s'escriu tot allò que es vol testejar i després el codi necessari perquè aquests tests passin. Escriure els tests abans que el software existeixi serveix per guiar el desenvolupament fent ús de l'error.

Es segueixen els següents passos de manera iterativa:

1. Escriure el test automàtic, executar-lo i veure que falla.
2. Desenvolupar el codi mínim imprescindible perquè el test que acabem d'escriure passi.
3. Netejar el codi.

En la següent imatge *Fig. 2.5*, podem observar els passos descrits anteriorment.

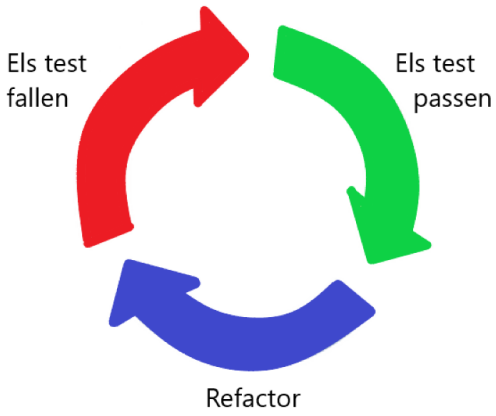

Fig. 2.5. Iteracions TDD. Font: Pròpia, 2021.

Aquesta metodologia aporta diversos beneficis respecte a les metodologies tradicionals de desenvolupament. La següent taula mostra una comparativa amb els avantatges i inconvenients de l'ús del TDD, cadascun dels quals estan demostrats en [5]:

Avantatges	**Inconvenients**
Millor disseny del projecte	Preparació organitzativa
Documentació detallada	Comprensió dels problemes
Reducció de temps de desenvolupament	
Estalvi de costos	
Situacions fiables	

Taula 2.1. Avantatges i inconvenients del TDD. Font: W. Mwaura

Existeixen diverses plataformes que ofereixen aprendre diferents llenguatges de programació. Plataformes com FreeCodeCamp i CodeCademy ofereixen aprendre diferents llenguatges de programació a partir de reptes. També existeixen plataformes com Codewars i CodeCademy que ofereixen aprendre diferents llenguatges de programació a partir de tests ja existents. Cap d'aquests tipus de plataformes, ofereixen aprendre TDD a partir de kates. Actualment per aprendre TDD, únicament existeixen cursos gratuïts o de pagament.

La idea d'aquest projecte sorgeix per la mancança en el mercat d'una aplicació d'ensenyament de TDD partint de kates. Es tracta d'una aplicació pensada per a empreses i autònoms que vulguin aprendre TDD de manera pràctica.

2.2.2 BDD

El BDD, és un procés de desenvolupament de software guiat per proves que deriva del TDD i es guia per l'enfocament de *Specification by Example (SBE)*. Aquest mètode va ser introduït per Dan North com a resposta a problemes que sorgien a l'hora d'ensenyar TDD. *"Els programadors volien saber per on començar, què provar i què no provar, quant provar d'una vegada, com anomenar les seves proves i com entendre per què falla una prova."*. [6]

BDD defineix els requeriments i proves funcionals de l'aplicació mitjançant exemples realistes i no abstractes. Fent ús d'un llenguatge comú basat en el domini i no en la tecnologia. Aconseguint que els documents generats serveixen com a documentació, ja que els entenen tant tècnics com empresarials, analistes i provadors. En la següent imatge *Fig. 2.6,* es veu la unió dels diferents llenguatges obtenint un llenguatge comú.

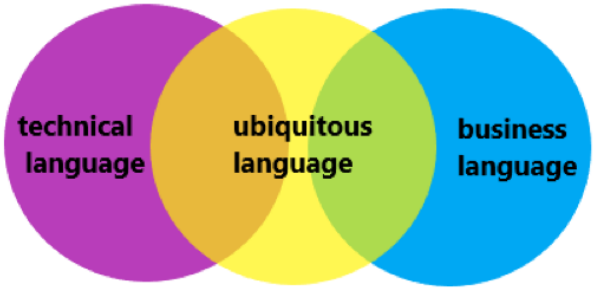

Fig. 2.6. Llenguatge comú BDD. Font: pròpia, 2021

Aquesta metodologia segueix els valors fonamentals del manifest àgil [7]:

- Individus i interaccions sobre processos i eines.
- Programari de treball sobre documentació completa.
- Col·laboració amb el client en la negociació del contracte.
- Respondre al canvi sobre seguir un pla.

La següent taula mostra una comparativa amb els avantatges i inconvenients de l'ús del BDD, cadascun dels quals estan demostrats en [8].

Avantatges	Inconvenients
Reduir els residus	Alt compromís i col·laboració comercial
Reduir els costos	Funciona millor en un context àgil o iteratiu
Canvis més fàcils i segurs	Els test mal escrits poden comportar costos mes elevats de manteniment
Releases més ràpids	

Taula 2.2. Avantatges i inconvenients del BDD. Font: J. F. Smart

El desenvolupament d'aquest projecte seguirà tant la metodologia BDD com TDD de manera conjunta.

2.3 Comparativa proves tradicionals i àgils

En aquest capítol, es fa una comparativa dels punts principals on les proves de software han de donar valor [3]:

- Enteniment dels membres del projecte:
    - Proves de software tradicionals:
        - Quan els desenvolupadors reben una nova funcionalitat, aquesta no pot tenir ambigüitats. En cas que tingui ambigüitats els desenvolupadors fan una interpretació basada en el seu coneixement. De manera que estan creant noves regles de negoci.
        - Quan el QA rep la funcionalitat té una interpretació diferent dels desenvolupadors. Proven coses diferents i amb diferents significats. Això genera errors inesperats per als desenvolupadors.
        - Mercat negre de defectes. Els desenvolupadors poden fer entregues més ràpides; baixant la qualitat i incrementant la quantitat d'errors.

Això fa que els QA trobin molts errors i faci la sensació que fan millor la seva feina tot i no ser així.

  - Proves àgils:
    - Les proves són exemples d'ús; mostren com funciona el producte. Són exemples senzills centrats en el que es vol mostrar. Aquestes proves fan que no hi hagi interpretacions diferents entre el QA i el desenvolupador, és a dir, estan alineats.

- Velocitat d'entrega
  - Proves de software tradicionals:
    - Inicialment, el desenvolupament és ràpid, però a mesura que es van desenvolupant funcionalitats, la velocitat de desenvolupament decreix. Això és degut a que el codi cada vegada és més complex i el desenvolupador necessita temps per realitzar els canvis sense afectar el codi existent.
  - Proves de software àgils:
    - Els desenvolupadors poden modificar i mantenir el codi de manera àgil. Revisen els tests (exemples) i comproven que tot funciona adequadament.
- Llançaments
  - Proves de software tradicionals:
    - Necessiten un equip de QA per realitzar les proves pertinents per aprovar el llançament de l'aplicació.
  - Proves de software àgils:
    - Les proves de software es proven de manera automàtica. Es pot implementar i llançar tan ràpid com l'últim desenvolupament hagi passat per CI/CD.

Les proves de software tradicionals, mostren clars desavantatges respecte a les proves de software àgils. Això és degut a la influència que té waterfall en les proves tradicionals.

En la següent imatge *Fig. 2.7*, es mostren els fluxos de treball tant de les proves de software tradicionals com les proves de software àgils. En les proves tradicionals, es crea el pla mestre o especificació que pot ser interpretat diferent pel desenvolupador i el QA.

En canvi, en les proves àgils, els "three amigos" treballen conjuntament per assegurar-se que sempre estan alineats.

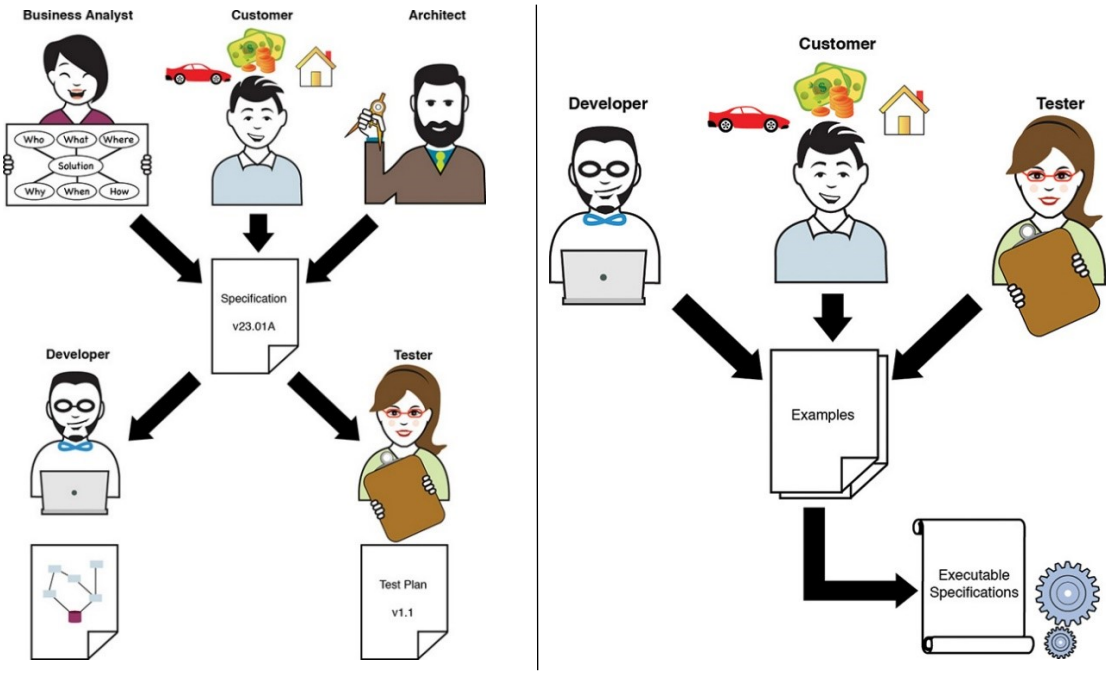

Fig. 2.7. Proves de software tradicionals vs proves de software àgils. Font: pròpia, 2021.

2.4 Backend

En aquest capítol s'analitza Spring, ja que un dels requeriments del projecte és utilitzar-lo per desenvolupar el backend.

El "Java Developer Productivity Report", és un informe fet per JRebel des de l'any 2012, on s'analitzen les tendències en la comunitat de desenvolupadors de Java. Aquest anàlisi es fa amb quasi 900 professionals amb perfils com: desenvolupadors (49%), arquitectes de software (24%), caps de projecte (17%), directors (6%) i consultors (4%).

Entre altres qüestions, es pregunta als desenvolupadors quin framework utilitzen al seu projecte principal, i un 62% dels enquestats responen Spring Boot [12].

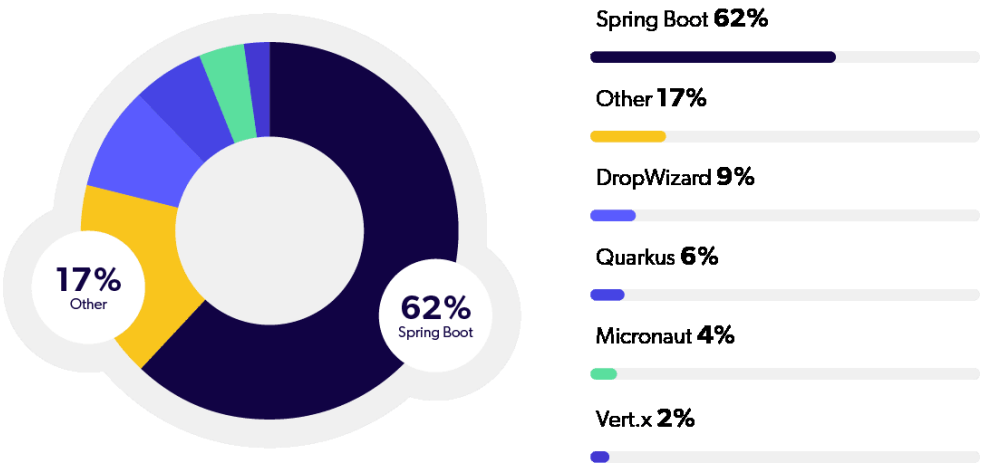

Fig. 2.8. Comparativa Java Framework, Java Developer Productivity Report, 2021.

Spring [13], és un framework pel desenvolupament d'aplicacions i contenidor d'inversió de control open-source per Java, creada l'any 2002. Spring pretén fer la programació de Java més ràpida, fàcil i segura.

Spring boot [14], és un framework de Spring creat l'any 2014. Spring boot permet crear projectes amb Spring, eliminant certes configuracions requerides per desplegar l'aplicació. Això és perquè proporciona una configuració automàtica per Spring Web MVC. Spring boot permet:

- Crear aplicacions Spring independents.
- Injecció de dependències.

- Proporciona connectivitat fluida amb diverses bases de dades (Oracle, MySQL, MongoDB...).
- Seguretat.

2.5 Frontend

2.5.1 React

En aquest capítol s'analitza React.js, ja que un dels requeriments del projecte és utilitzar-lo per desenvolupar el frontend.

Javascript, és un dels llenguatges més populars per al desenvolupament de webs. Aquest llenguatge, va ser creat per Brendan Rich l'any 1995 i estandarditzat per ECMA (European Computer Manufacturers Association) l'any 1997. A part d'aplicacions web, amb Javascript es pot desenvolupar aplicacions d'escriptori, per dispositius mòbils i server-side usant els diferents frameworks que ofereix com Meteor, React Native i Node.js. [15]

La popularitat de React, llibreria creada per Facebook l'any 2013, ha anat creixent durant els darrers anys i no mostra signes d'aturar-se.

La següent imatge *Fig. 2.9*, mostra les tendències d'adopció de les llibreries de JavaScript.

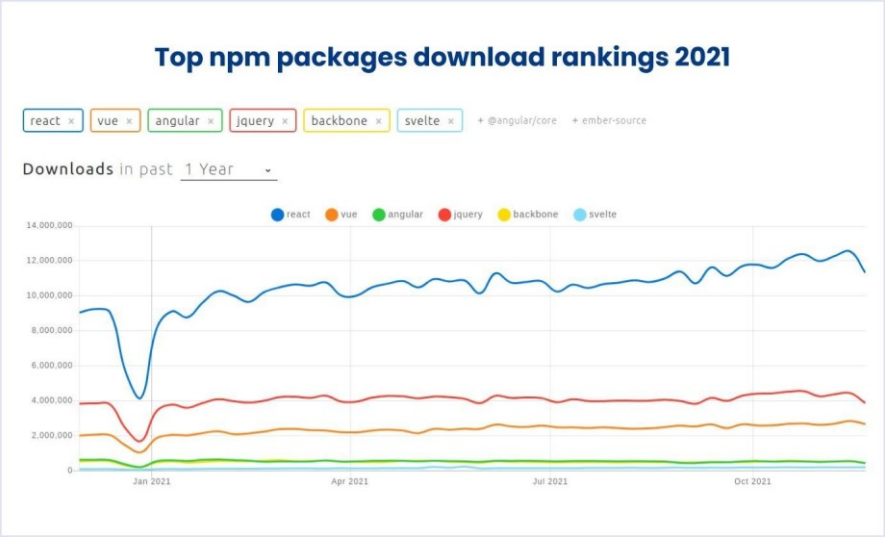

Fig. 2.9. Tendències llibreries de Javascript, Oleksandra, Dmitry i Eugene, 2021.

2.5.2 Redux

Redux, és un contenidor de l'estat d'aplicacions JavaScript, és a dir, és com una base de dades en memòria.

És necessari gestionar l'estat de les aplicacions, ja que no totes tenen accés a un disc o a una base de dades de manera segura. A més, les aplicacions poden tenir múltiples vistes que interactuen amb les mateixes dades.

D'aquesta manera, ajuda a gestionar l'estat accessible per les diferents parts de les aplicacions i a coordinar-les.

Els principals avantatges del Redux són: [16]

- Estat global i immutable.
- Major control de l'estat de l'aplicació i del flux de dades.
- Arquitectura estable de dades.

2.5.3 React Redux

Redux, va ser originalment dissenyat per utilitzar-lo juntament amb React tot i que es pot utilitzar en qualsevol capa de la UI.

El principal inconvenient de React és que, tot i ser generalment ràpid, per defecte quan hi ha qualsevol actualització d'un component, React renderitza tots els components dins de l'arbre de components. Això requereix treball i un esforç innecessari, ja que en molts casos es renderitzen components que no han canviat el seu estat.

Per millorar el rendiment de l'aplicació i evitar renderitzar components de manera innecessària, apareix React Redux [17]. Redux-React, implementa moltes optimitzacions de rendiment perquè els components només es renderitzin quan realment sigui necessari.

En la següent imatge *Fig. 2.10*, es mostra com s'actualitza l'estat en una aplicació React on no s'utilitza Redux i en una aplicació React on s'utilitza Redux.

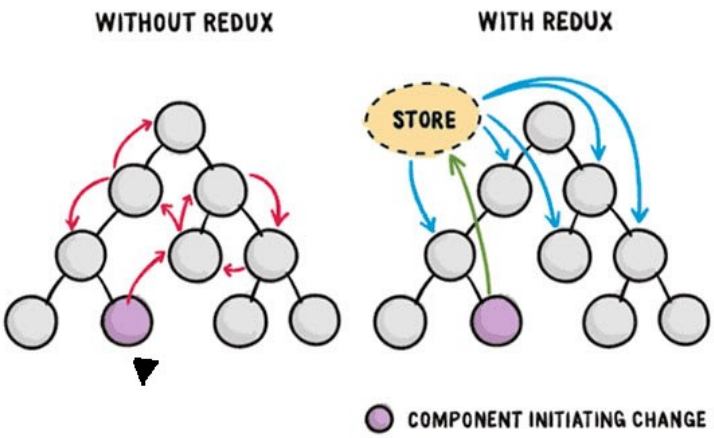

Fig. 2.10. React vs React Redux, Ariel Mirra, 2020.

2.6 Bases de dades

2.6.1 Relacionals

Les bases de dades relacionals, són una col·lecció de dades organitzades en un conjunt de taules les quals disposen d'una relació predefinida dels elements.

Les bases de dades relacionals presenten diferents avantatges [9]:

- Integritat referencial: disposa d'eines per evitar duplicitat dels registres.
- Atomicitat de la informació: Si sorgeix algun problema en el moment de fer una operació a la base de dades, aquesta no s'executa.
- Llenguatge estructurat: ús SQL per les operacions de la base de dades.

2.6.2 No relacionals

Les bases de dades no relacionals, a diferència de les bases de dades relacionals, no requereixen estructures fixes com taules. Aquest tipus de bases de dades, també són conegudes com NoSQL (Not Only SQL) per remarcar el fet que poden suportar llenguatges de consulta SQL i més.

Les bases de dades no relacionals presenten diferents avantatges [9]:

- Dades versàtils: Es pot afegir informació o fer canvis en el sistema sense necessitat d'afegir noves configuracions.

- Creixement horitzontal: Suporta estructures distribuïdes instal·lades en diferents nodes per balancejar la càrrega de treball.

2.6.3 Comparativa bases de dades relacionals i no relacionals.

En la següent taula, es mostra una comparació de les bases de dades relacionals i no relacionals de les estructures de dades, relacions entre aquestes i transaccionalitat.

Descripció	No relacional	Relacional
Estructura de dades	Estructura flexible	Estructura ben predefinida de dades des de l'inici.
Relacions	No hi ha relació entre col·leccions	Les relacions han d'estar ben definides i establertes.
Transaccionalitat	Es perd integritat de les dades en les transaccions	Ús ACID (Atomicitat, Consistència, Aïllament, Durabilitat).

Taula 2.3. Comparativa bases de dades relacionals i no relacionals. Font: R. F. Córdova i B. E. Cuzco

A causa de la flexibilitat que presenten les bases de dades no relacionals en l'estructura de dades respecte a les bases de dades relacionals, aquest projecte utilitzarà bases de dades no relacionals. L'estructura de les dades del projecte pot anar variant durant el temps. És per això que la poca flexibilitat de canvi que presenten les bases de dades relacionals podria frenar les primeres etapes de desenvolupament.

2.7 MongoDB

MongoDB, és un sistema de bases de dades no relacionals, orientat a documents i open source.

Aquest sistema de bases de dades, emmagatzema les dades en estructures de dades BSON (Binary JSON) amb un esquema dinàmic. Això fa que sigui més senzilla i ràpida la integració de dades en certes aplicacions.

En la següent imatge *Fig. 2.11*, podem observar un exemple de document. Tot document té un camp anomenat _id, el qual és proporcionat per MongoDB. D'altra banda, podem afegir tants camps com desitgem.

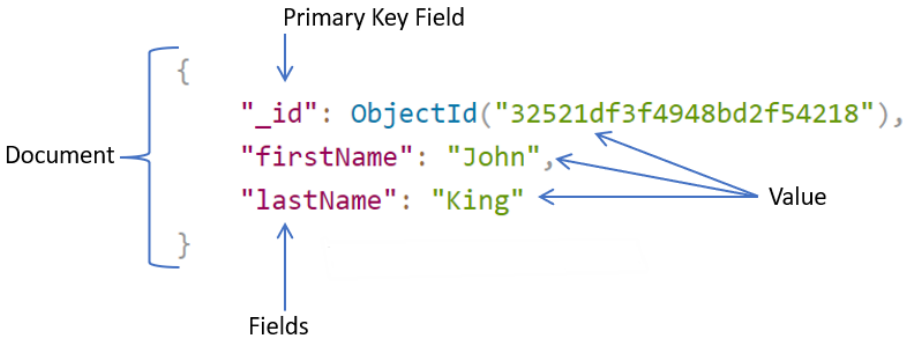

Fig. 2.11. Exemple document. Font: TutorialsTeacher, 2022.

En la següent taula, s'analitzen les avantatges i inconvenients de MongoDB:

Avantatges	Inconvenients
Fàcilment escalable	No és adequat per aplicacions amb transaccions complexes
Cost baix	No té Joins per consultes
Documents com a model de dades	

Taula 2.4. Avantatges i inconvenients MongoDB. Font: MongoDB

2.8 Hosting

El "state of the cloud report", és un informe fet per Flexera on s'analitzen les tendències cloud en les empreses. L'anàlisi es fa amb 750 empreses d'arreu del món.

Entre altres qüestions, es pregunta als enquestats quin PaaS utilitzen actualment per les seves aplicacions, quins es plantegen utilitzar, amb quins estan experimentant i quins no utilitzarien mai. Com s'observa en la següent imatge, el top 3 PaaS utilitzats són AWS, Azure i Google Cloud. [19]

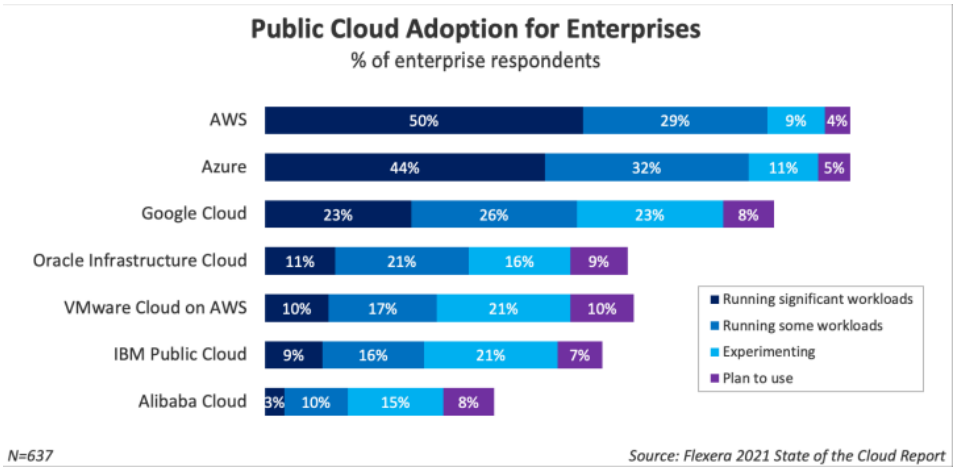

Fig. 2.12. Top Public Cloud en empreses. Font: Flexera, 2021.

Un dels objectius del projecte és fer ús de CI/CD. Ja que fer ús de CI/CD amb AWS, Azure i Google Cloud és de pagament, s'ha estudiat un altre possibilitat tot i no ser un dels PaaS en el top 7 per les empreses.

Heroku, és un PaaS de Salesforce el qual en les seves prestacions bàsiques ofereix l'ús de CI/CD. Aquest PaaS cobreix les prestacions bàsiques necessàries per dur a terme el projecte. En la següent taula es mostren les prestacions que ofereix de manera gratuïta Heroku [20]:

Pla gratuït	**Pla de pagament**
RAM: 512 MB	"Always on" (Dorm després de 30 minuts sense activitat)

Desplegament des de github	No pot combinar múltiples tipus dyno (contenidors gestionats en temps d'execució)
"Patching" automatitzat del sistema operatiu	
Registres unificats	
Dos tipus de processos actius	
Noms de dominis personalitzats	

Taula 2.5. Prestacions Heroku. Font: Heroku

3. Objectius i abast

El producte resultant d'aquest projecte, té com a objectiu que empreses i autònoms aprenguin de manera pràctica TDD, resolent kates de diferents nivells. Cadascuna de les kates tindrà diferents tests, els quals se'ls anirà donant de manera progressiva. Com s'ha explicat anteriorment, en cada iteració de TDD, cal desenvolupar el codi mínim imprescindible perquè el test que acabem d'escriure passi. És per això que, la manera d'avaluació es farà segons la quantitat de caràcters modificats en cada test.

3.1. Usuaris potencials

Es pot distingir quatre tipus diferents d'usuaris potencials, els quals són:

- **Organització/empresa**: Entitat interessada en que els seus treballadors aprenguin TDD per a poder treure benefici econòmic d'aquest coneixement.
- **Treballador** (usuari associat a una organització/empresa): Persona interessada en aprendre TDD per a poder aplicar els coneixements posteriorment en l'empresa.
- **Autònom** : Persona independent interessada en aprendre TDD per motius aliens a empreses.
- **Administrador**: Persona que gestiona el contingut de l'aplicació.

3.2. Objectius dels usuaris

Segons el tipus d'usuari potencial, els objectius varien.

- Objectius com a organització/empresa:
    - Millorar el coneixement de TDD en l'equip de programació.
    - Afegir usuaris en una organització.
    - Consultar estadístiques de les diferents kates resoltes de cada empleat.
    - Consultar ranking dels seus empleats.
- Objectius com a autònom i com a treballador:
    - Aprendre TDD resolent katas de manera incremental.

    - Consultar estadístiques de les kates resoltes.
    - Consultar el ranking.
    - Fer autopromoció.
    - Objectius com a administrador:
- Crear noves kates.
    - Gestionar organitzacions.
    - Gestionar usuaris.
    - Donar/denegar accessos a les organitzacions.

3.3. Requeriments

Un cop analitzats els usuaris potencials de l'aplicació i els objectius que tenen cadascun d'ells, es poden definir els requeriments.

Els requeriments es divideixen en requeriments primaris i secundaris. Els requeriments primaris són aquells que s'han de complir, ja que són la base del producte que es desenvolupa. Els requeriments secundaris són aquells addicionals els quals augmenten el valor del producte, però poden ser desenvolupats un cop finalitzat el projecte.

Requeriments primaris:

- Requeriments funcionals:
    - Crear lògica per resoldre kates.
    - Puntuar les resolucions de les kates segons si s'ha seguit bé TDD o no.
    - Mostrar estadístiques de les resolucions de les kates.
    - Crear nous usuaris.
    - Accedir a l'aplicació amb un nom d'usuari i una contrasenya.
    - Mostrar un historial de les kates resoltes
    - Mostrar un ranking amb les puntuacions de tots els usuaris.
- Requeriments tecnològics:
    - Mantenir persistència de dades.
    - Comunicar l'aplicació web amb el backend a través del protocol HTTP.
    - Adaptar l'aplicació per a que sigui accessible des de qualsevol dispositiu fent-la responsiva.

Requeriments secundaris:

- Requeriments funcionals:
    - Crear nous usuaris amb rol 'organització', 'treballador d'una organització' i 'autònom'.
    - Consultar les estadístiques dels treballadors d'una organització si ens el rol d'organització.
    - Mostrar un ranking amb les puntuacions de tots els treballadors d'una organització.
    - Mostrar un ranking amb les puntuacions de tots els autònoms.
    - Gestionar kates.
- Requeriments tecnològics:
    - Emmagatzemar les contrasenyes encriptades en la base de dades.
    - Comunicar l'aplicació web amb el backend a través del protocol HTTPS.
    - Disposar de SSL en l'aplicació.

4. Anàlisi de referents

4.1 The Transformation Priority Premise

Un dels punts més importants de la metodologia TDD és que, a mesura que s'escriuen tests més específics, el codi es generalitza.

A mesura que s'afegeixen noves restriccions, s'escriuen nous tests, però en cap cas es modifiquen o eliminen els tests ja existents.

D'altra banda, fer que el codi de producció passi el test, no és tan simple com afegir línies de codi, sinó que molts cops requereix modificar el codi ja existent. Aquestes modificacions en el codi alteren el comportament d'aquest.

Un cop fetes les modificacions necessàries, cal netejar el codi perquè sigui més fàcil de mantenir. En aquest cas tot i fer modificacions en el codi de producció, el comportament d'aquest no queda alterat.

El concepte "Transformation Priority Premise" [21], són les petites modificacions al codi de producció que fan que comportament d'aquest quedi alterat i alhora més generalitzat.

Robert "Uncle Bob" Martin proposa seguir el següent ordre per fer transformacions, és a dir, per generalitzar el codi que desenvolupem a mesura que els tests ho requereixen:

- {} → Null
- Null → Constant
- Constant → Variable
- Unconditional → Selection
- Value → List
- Selection → Iteration
- Statement → Recursion
- Value → Mutated Value

"No tinc cap prova matemàtica, i no estic segur que sigui així en tots els casos. El que estic relativament segur és que és probable que acabeu amb millors implementacions si trieu les transformacions en alguna cosa com l'ordre següent"- Robert "Uncle Bob" Martin, 2013.

Tot i no existir cap prova matemàtica que demostri que l'ús de les transformacions assegura l'èxit del TDD, aquesta proposta, es fa a partir del coneixement empíric com a professional del sector de "Uncle Bob".

El principal objectiu de les transformacions, és ajudar a desenvolupar TDD evitant quedar-se encallat a causa d'haver fet tests massa genèrics. Si es duen a terme dues o més transformacions en un mateix test, és possible que anteriorment no s'hagi contemplat algun cas.

Seguir les transformacions en l'ordre especificat, porta a implementar solucions més senzilles i ràpides. De manera que, en cas de fer més d'una transformació en un mateix test, cal plantejar-se un test previ que contempli la situació que s'està oblidant.

4.2 Golf testing

El terme Golf Testing ve derivat d'un joc de paraules usat per Uncle Bob en el curs "Clean code 19 – Advanced TDD" [22].

Uncle Bob es refereix al terme 'fer golf', en aquelles situacions on els tests ens permeten generalitzar el codi fent canvis mínims. És a dir, fa referència al concepte explicat en l'apartat anterior *The Transformation Priority Premise*.

"Voleu saber quin és el secret del Test Driven Development? Apropa't. Aquí està: Com més específics siguin els tests, més genèric és el codi." - *Robert "Uncle Bob" Martin, 2013.*

En el curs Uncle Bob explica que, com el seu fill petit tenia dubtes amb una activitat de càlcul dels nombres primers, va decidir desenvolupar un programa perquè el seu fill

pugues verificar que ha resolt correctament l'activitat. Per desenvolupar aquest programa, va decidir que en lloc de plantejar-se un algoritme nou o fer ús d'algun existent, faria ús dels tests per a trobar la solució. *"Llavors vaig decidir deixar als tests conduir-me a la solució abans d'imposar una solució als tests".- Robert "Uncle Bob" Martin*

Uncle Bob, seguint la metodologia TDD, acaba resolent el problema dels nombres primers. L'algoritme resultant despres d'haver seguit la metodologia TDD és l'algoritme d'Eratosthenes. Uncle Bob arriba a aquesta solució tot i no haver planejat l'algoritme inicialment.

"Si mires detingudament, veuràs que és l'algoritme d'Eratosthenes." - Robert "Uncle Bob" Martin, 2013.

Es pot consultar l'exemple sencer de com va resoldre Uncle Bob el problema dels nombres primers, en *Annex I. Golf testing: algoritme nombre primers*.

Uncle Bob, demostra que a mesura que s'escriuen tests més específics, el codi es generalitza. Quan s'aplica una generalització, el codi a afegir és menor al que cal afegir fent el codi concret.

"Ara juguem una mica de golf. Intenta passar el test amb la menor quantitat de caràcters possibles. [...] Puc passar el test modificant un caràcter." - Robert "Uncle Bob" Martin, 2013.

Uncle Bob, fa 'Golf' en diverses situacions, com:

- En el següent exemple podem veure com, en lloc de crear *n* condicionals, un per cada valor possible, generalitzem fent: `primes.add(n)`

EXEMPLE 1:

```
if (n > 1)                          if (n > 1)
    primes.add(2);          ──►         primes.add(n);
```

- En el següent exemple podem veure com, en lloc d'afegir un condicional per cada situació, es fa la transformació del condicional "*if*" a *loop*. D'aquesta manera es comprova la condició tantes vegades com sigui necessari.

EXEMPLE 2:

```
if (n % 2 == 0) {
    primes.add(2);
    n /= 2;
}
```

```
while (n % 2 == 0) {
    primes.add(2);
    n /= 2;
}
```

4.3 Problema del TDD

El principal problema que es troben els programadors quan fan ús del TDD, és utilitzar un ordre inadequat dels tests o avançar-se en el moment d'escriure'ls.

Escriure les proves per situacions complexes abans d'haver provat les situacions més simples, pot comportar a situacions on és impossible avançar en el codi. De manera que, per continuar avançant, s'ha de començar de zero o resoldre tot l'algoritme.

Un professional del sector, en el curs "Clean code 19 – Advanced TDD" [22] afirma:

"Estava treballant a Motor Yard i vaig escriure una prova de test que era homogènia; provava tot a tot el programa. Vull dir que aquesta prova era un monstre. Eren pàgines i pàgines i pàgines. I llavors per aconseguir que la prova passes, vaig haver d'escriure muntanyes senceres de codi de producció" - Ruby Rod, 2013.

D'altra banda, un company seu, seguint altres passes acaba en la mateixa situació:

"A vegades estic a mig camí en un problema i llavors escric alguna petita prova simple, i l'única manera de fer que aquesta prova passi és fer que tot el programa funcioni", Jerry Java, 2013.

Estar encallat, és un símptoma que quelcom s'està fent malament. Pot ser perquè s'està escrivint malament els tests o perquè el codi de producció és massa genèric.

Uncle Bob en el curs "Clean code 19 – Advanced TDD" demostra amb l'exemple del *Word Wrap problem,* com l'ordre dels tests influeix en la resolució del problema. Aquest problema consisteix en el següent:

Es tracta de desenvolupar una funció la qual té dos paràmetres d'entrada, un String i un nombre enter. Aquesta funció retorna el String, però amb salts de línia inserits als llocs adequats, per assegurar-se que cap paraula és més llarga que el nombre enter.

En l'exemple, Uncle Bob, demostra com aplicar correctament TDD partint d'un exemple mal resolt:

Des de la primera iteració, comença amb el cas més degenerat possible:

Cas més simple:

```
wordWwrapProblem("", 2)
```

Cas més complexa:

```
wordWwrapProblem("co co", 2)
```

Posteriorment, s'ha d'anar pujant cada cop més la complexitat. En cada iteració s'ha de passar la prova generalitzant el codi de producció, en comptes de fer una correcció específica per passar una prova en concret.

"Estic força d'acord que cal mitigar el risc des del principi. La qüestió és identificar el risc més gran. El risc més gran del desenvolupament basat en proves és quedar-se encallat. Mitiguem aquest risc evitant els comportaments que hi condueixen, és a dir, els comportaments que salten a la complexitat d'un algorisme i eviten l'enfocament incremental d'augmentar gradualment aquesta complexitat." Robert "Uncle Bob" Martin, 2013.

Després de demostrar una manera errònia de resoldre el problema, planteja la resolució dels tests des del cas més senzill fins al més complex. Sense escriure situacions complexes abans de resoldre les situacions més simples.

Es pot consultar l'exemple sencer de com va resoldre Uncle Bob, en *l'Annex II. World Wrap Problem.*

L'aplicació desenvolupada, ofereix als usuaris la possibilitat de resoldre kates de diferents nivells de manera incremental. Per a la resolució de les kates, apareixen tests a mesura que es resolen els anteriors. D'aquesta manera, a mesura que resol la kata, l'usuari no s'encalla i aprèn bones pràctiques des d'un inici.

5. Metodologia

El projecte serà desenvolupat seguint el *model espiral* [11]. El model espiral és un procés evolutiu del procés de software. Aquest model es guia pel control i la gestió de riscs per l'anàlisi i estructuració del procés de desenvolupament. Com el seu nom indica, és un model amb naturalesa iterativa.

En cada iteració es segueixen quatre etapes, les quals podem veure representades la següent figura:

- **Determinar o fixar objectius:** Definició dels objectius i alternatives per poder identificar les limitacions del procés o del sistema. També es dissenya una planificació de la gestió i s'identifiquen els riscs.
- **Anàlisis de riscs:** Anàlisi detallat per cadascun dels riscs identificats en el projecte. Es realitza un anàlisis d'alternatives i identificació i resolució per cada risc.
- **Desenvolupar, verificar i validar:** Activitats relacionades amb el desenvolupament del producte. Inclou la construcció de prototips de nivell cada cop més refinats, fins a arribar al producte final.
- **Planificar:** Revisió i presa de decisió de si segueix el projecte endavant.

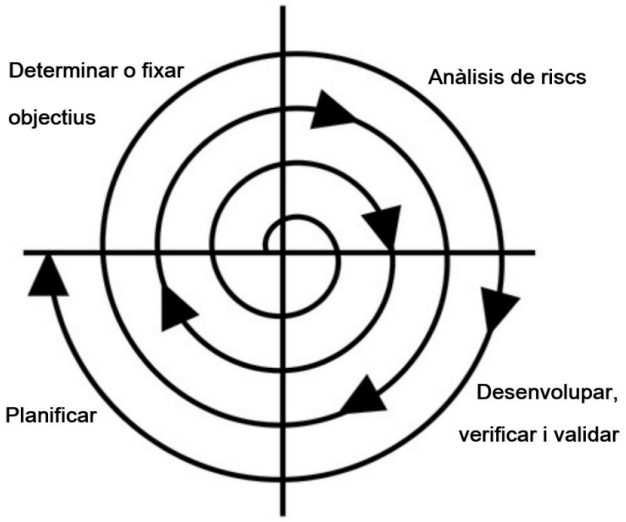

Fig. 5.1. Fases model espiral. Font: pròpia, 2021.

En la següent taula podem observar la comparativa dels principals avantatges i desavantatges d'aquesta metodologia.

Avantatges	Desavantatges
No requereix la definició completa de requeriments per començar.	Complicació en el moment d'avaluació de riscs.
Es poden anar afegint nous requisits des del final de la primera iteració.	Participació contínua del client.
Els retards són un risc menor, ja que hi ha maneres de solucionar-ho.	Cada iteració no té una durada concreta.

Taula 5.1. Avantatges i inconvenients model espiral. Font: G. Fariño.

Aquesta metodologia és adequada en aquest projecte, ja que aporta els següents punts:

Característica del model espiral	Definició	Aportació al projecte
Comunicació amb el client	Comunicació entre el desenvolupador i el client.	El projecte serà desenvolupat seguint el procés de desenvolupament de software BDD. Procés en el qual pot participar activament el client en el moment de redactar els requeriments.
Planificació	Requeriments del projecte (recursos, temps...)	En cada iteració es farà una predicció de temps i recursos necessaris. A mesura que el projecte avanci, les prediccions seran més realistes gràcies a l'experiència obtinguda en les anteriors.
Anàlisis de riscs	Avaluació dels riscs tècnics i altres informacions relacionades amb el projecte.	El projecte és de curta durada, de manera que, qualsevol risc pot comportar no assolir tots els objectius definits.

Enginyeria	Construcció de la versió *n* del projecte.	En finalitzar cada iteració de l'espiral, s'obtindrà una nova versió funcional del projecte.
Construcció i adaptació	Construir, provar, instal·lar i proporcionar suport al client.	-
Evolució del client	Avaluació del client de les diferents versions de software creats durant l'etapa de desenvolupament.	Avaluació continua en les reunions de seguiment del projecte i en les dues fites principals del projecte.

Taula 5.2. Model espiral en el projecte. Font: G. Fariño.

1.1. Iteracions

Com s'ha explicat anteriorment, el mètode espiral està dividit en iteracions i en cadascuna de les iteracions hi ha quatre etapes: determinar/fixar objectius, anàlisi de riscs, desenvolupar i planificar. En el moment de finalització de la primera fita (avantprojecte) i inici de la següent (memòria intermèdia) començaran les iteracions del projecte.

1.1.1. Primera iteració

La primera iteració comença just en el moment de finalització de la primera fita. En aquesta iteració es prepara el *setup* del projecte. Aquest setup inclou:

- Configuració del sistema de control de versions del software Github.
- Anàlisis dels diferents llenguatges de programació de frontend i configuració.
- Anàlisis dels diferents llenguatges de programació de backend i configuració.
- Configuració host.

1.1.2. Segona iteració

En la segona iteració es comença a desenvolupar la primera funcionalitat de l'aplicació. Es tracta de la creació de la lògica per resoldre kates. Cal analitzar les diferents llibreries existents.

En finalitzar la iteració s'obté una primera versió funcional del projecte.

1.1.3. Tercera iteració

En la tercera iteració es crea la lògica per mostrar una estadística en el moment de finalitzar una kata. En aquesta estadística es mostra l'evolució que hi ha hagut en la resolució de la kata segons la quantitat de caràcters modificats en cada test.

En finalitzar la iteració s'obté una nova versió funcional del projecte on es mostra una estadística en finalitzar una kata.

1.1.4. Quarta iteració

En la quarta iteració es crea la lògica per iniciar sessió i per registrar-se. Es requereix de persistència de dades.

En finalitzar la iteració s'obté una nova versió funcional del projecte on hi ha registre d'usuaris.

1.1.5. Cinquena iteració

En la cinquena iteració s'afegeix la possibilitat de resoldre diferents kates.

En finalitzar la iteració s'obté una nova versió funcional del projecte on hi ha diverses kates a resoldre.

1.1.6. Sisena iteració

En la sisena iteració es crea la lògica perquè cada usuari tingui un registre de les seves kates, de manera que, es guarda un registre de les kates resoltes per usuari i cada usuari pot consultar les seves estadístiques quan ho desitgi.

D'altre banda, es comença a fer un anàlisis dels resultats possibles de cada kata per buscar un patró en les estadístiques.

- Resolució amb refactor.
- Resolució sense refactor.
- Solució massa concreta.
- Solució massa generalitzada.

En finalitzar la iteració s'obté una nova versió funcional del projecte on hi ha el registre de kates per cada usuari.

1.1.7. Setena iteració

En la setena iteració s'afegeix una puntuació en finalitzar la kata. Aquesta puntuació segons si s'ha aplicat correctament TDD o no. Per poder puntuar la resolució de la kata, es fa ús de l'anàlisi fet en la iteració anterior.

D'altra banda, es crea la lògica de rankings. Cadascun dels usuaris pot consultar un ranking comú segons la suma de totes les puntuacions obtingudes en les kates.

En finalitzar la iteració s'aconsegueix una nova versió funcional del projecte on hi ha el ranking d'usuaris.

1.1.8. Vuitena iteració

En la vuitena iteració es crea un nou rol, el rol d'organització. Es modifica la lògica per registrar-se, hi ha tres possibilitats:

- Registrar-se com a organització.
- Registrar-se com a usuari membre d'una organització.

- Registrar-se com a autònom, és a dir, com a usuari que no pertany a cap organització.

D'altra banda, s'afegeix la lògica on cada organització pot consultar les estadístiques de cadascun dels usuaris membres de l'organització.

En finalitzar la iteració s'obté una nova versió funcional del projecte on hi ha un nou rol d'usuari anomenat organització el qual té privilegis diferents dels usuaris estàndard.

1.1.9. Novena iteració

En la novena iteració es modifica la lògica de rankings, hi ha dos possibilitats:

- Ranking d'organització: ranking on només participen usuaris membres d'una organització.
- Ranking dels autònoms: ranking on només participen usuaris que no són membres de cap organització.

En finalitzar la iteració s'obté una nova versió funcional del projecte on existeixen dos tipus de ranking diferent en lloc d'un únic global.

1.1.10. Desena iteració

En la desena iteració, s'afegeix un nou rol, el rol d'administrador. Es crea la lògica perquè els usuaris amb rol administrador puguin afegir noves kates des de la mateixa aplicació.

En finalitzar la iteració s'obté una nova versió funcional del projecte on hi ha un nou rol d'usuari anomenat administrador el qual té privilegis diferents dels usuaris estàndard.

6. Desenvolupament

En aquest capítol s'expliquen els resultats de cada iteració i les decisions preses. Es poden consultar els detalls de cada iteració en els annexos.

Com s'explica en l'apartat *Metodologia* s'utilitza la metodologia espiral per desenvolupar l'aplicació.

En cada iteració es segueixen quatre etapes:

- **Determinar o fixar objectius:** Definició dels objectius platejats en la iteració.
- **Anàlisi de riscs:** Per fer l'anàlisi de riscs de cada iteració, es quantifica la probabilitat i l'impacte de cada risc detectat. A partir d'aquesta informació s'obté la severitat del risc, és a dir, la seva criticitat [23].
  A partir de la severitat de cada risc, es marca la prioritització a l'hora de prendre accions mitigadores.

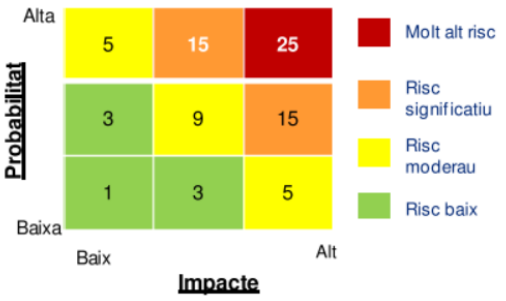

Fig. 6.1. Anàlisi riscs, Generalitat Catalunya, 2021

- **Desenvolupar, verificar i validar:** Totes aquelles activitats relacionades amb el desenvolupament del producte.
- **Planificar:** Revisió i presa de decisió de si segueix el projecte endavant.

6.1 Primera iteració

Desenvolupar, verificar i validar:

Configuració Github Actions

GitHib Actions [24], és una plataforma d'integració i entrega continua (CI/CD), que permet automatitzar el canal de construcció, prova i desplegament. Permet crear fluxs de treball anomenats *jobs* per crear i provar cada sol·licitud de canvi en el repositori.

GitHub proporciona màquines virtuals Linux, Windows i macOS per a executar els fluxs de treball. Tot i això, també proporciona la possibilitat d'hostejar servidors propis ja siguin físics on en cloud.

S'ha decidit distribuir el projecte en diferents branques:

- **Main**: únicament hi ha el codi que està en producció.
- **Iteration n**: una per cada iteració. En cada branca es desenvolupa les funcionalitats pertinents de la iteració.

Fig. 6.1. Flux desenvolupament. Font: pròpia, 2022.

D'aquesta manera s'han plantejat dos fluxos de treball, el de la branca main i el de les branques de les diferents iteracions.

- **Flux main**: S'executa una vegada es fa merge de les diferents *Pull Request* de cada iteració. La seva finalitat és fer el desplegament automàtic a Heroku. D'aquesta manera no és necessari fer el desplegament manualment en finalitzar cada iteració.
- **Flux iteracions**: S'executa cada vegada que es fa un commit a qualsevol branca. La seva finalitat és verificar que els tests passen i que el codi té un coverage del 100%. D'aquesta manera ens assegurem que sempre tenim el codi funcionant i net.

Es pot consultar l'estructura d'un flux i els fluxos creats en *l'Annex X. Github Actions.*

Anàlisis dels diferents llenguatges de programació de backend.

S'han analitzat les tendències de la comunitat de desenvolupadors de Java.

L'anàlisi de les tendències de la comunitat, es pot consultar en el capítol *Llenguatge backend*.

Configuració backend.

Com s'explica en l'apartat *BDD*, s'utilitza aquesta metodologia per a desenvolupar l'aplicació. Per poder-ho fer, és necessari l'ajuda d'una llibreria.

Cucumber [25] és una llibreria que llegeix especificacions escrites en un llenguatge ordinari anomenat *Gherkin,* i valida que el software faci el que les especificacions diuen.

Gherkin és un llenguatge específic de domini (DSL). Els llenguatges específics de domini, són llenguatges dissenyats per resoldre algun problema. En aquest cas, *Gherkin* està plantejat per solucionar el problema de la comunicació entre perfils tècnics i de negoci.

Gherkin utilitza un conjunt de paraules clau per donar una estructura i significat a les especificacions.

Es pot consultar el funcionament del llenguatge Gherkin en *l'Annex XI. Gherkin.*

Es pot consultar la instal·lació i el funcionament de la llibreria Cucumber per Java en *l'Annex XII. Cucumber per Java.*

Fig. 6.2. Logo cucumber. Font: Cucumber, 2022.

Anàlisi dels diferents llenguatges de programació de frontend.

S'han analitzat els llenguatges de programació React, Redux i React Redux.

L'anàlisi dels llenguatges, es pot consultar en l'aparat *Llenguatge frontend*.

Configuració frontend.

Com s'ha explicat en l'apartat *Backend* al desenvolupar seguint la metodologia BDD, és necessari l'ajuda de la llibreria *Cucumber*.

Es pot consultar la instal·lació i el funcionament de la llibreria Cucumber per Java en *l'Annex XIII. Cucumber per Javascript*

Anàlisi, elecció i configuració del host

S'han analitzat les tendències de cloud en les empreses i s'ha escollit un host tenint en compte les prestacions que es necessiten.

L'anàlisi i elecció del host, es pot consultar en l'aparat *Hosting*.

La configuració de Heroku, és més complex de l'esperat degut a la falta de documentació.

En la següent imatge *Fig. 6.3*, es mostra la configuració del host. El domini és https://golftesting-barberan.herokuapp.com/

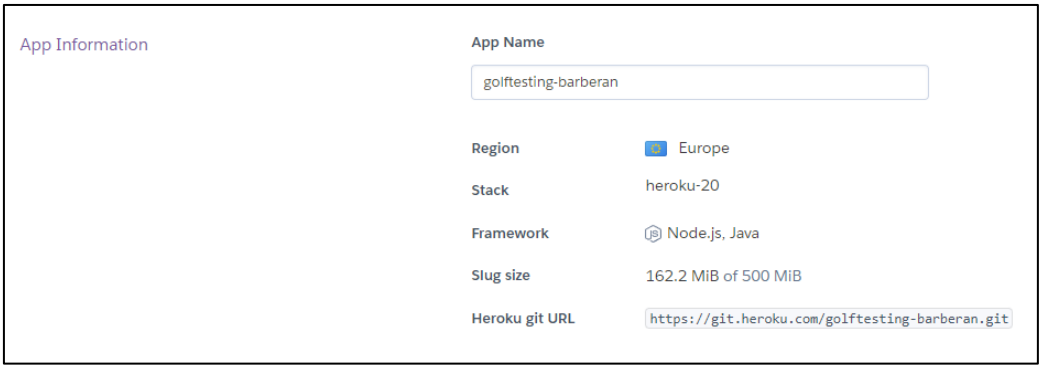

Fig. 6.3. Configuració host. Font: pròpia, 2022.

6.2 Segona iteració

Desenvolupar, verificar i validar:

Escriure històries d'usuari per la funcionalitat a desenvolupar

S'han escrit tres històries d'usuari:

- COM usuari de l'aplicació VULL veure el test que he de passar PER A poder aprendre TDD.
- COM usuari de l'aplicació VULL poder passar al següent test quan l'actual ja l'hagi resolt PER A resoldre una kata.
- COM usuari de l'aplicació VULL un editor per a resoldre tests d'una kata PER A poder resoldre una kata.

Escriure els diferents escenaris seguint la metodologia BDD

Partint de les històries d'usuari, s'han escrit dos escenaris:

- Escenari 1:
    - Exemple: Resoldre un test correctament.
    - Contra exemple: Resoldre un test incorrectament
- Escenari 2: Un cop resolt el test actual, passar al següent test.

Desenvolupament.

En aquesta segona iteració, no es necessita backend ni persistència de dades.

Tenint en compte que es segueix la metodologia BDD i ja s'han escrit els escenaris necessaris, cal seguir els següents passos:

1. Escriure el test, executar-lo i veure que falla.
2. Desenvolupar el codi mínim imprescindible perquè el test que acabem d'escriure passi.
3. Netejar el codi.

Desplegament de la primera versió de l'aplicació

En finalitzar la iteració, tenim una única pàgina on l'usuari pot resoldre els tests que van apareixent. Perquè l'usuari sàpiga si el test passa o no, aquest es mostra en verd o vermell respectivament.

En haver configurat les github actions, en el moment de fer el "merge" amb la branca main, el desplegament es fa de manera automàtica.

6.3 Tercera iteració

Desenvolupar, verificar i validar:

Escriure històries d'usuari per la funcionalitat a desenvolupar

S'han escrit quatre històries d'usuari:

- COM usuari de l'aplicació VULL veure quantes tecles he polsat en el test actual PER A poder saber quantes tecles porto en el test actual.
- COM usuari de l'aplicació VULL veure quantes tecles he polsat en total PER A poder saber quantes tecles porto en total.
- COM usuari de l'aplicació VULL veure una estadística un cop finalitzada la kata PER A poder veure la meva evolució.
- COM usuari de l'aplicació VULL veure una puntuació un cop finalitzada la kata PER A poder saber si he resolt bé la kata o no.

Escriure els diferents escenaris seguint la metodologia BDD

Partint de les històries d'usuari, s'han escrit dos escenaris:

- Escenari 1: L'usuari veu en temps real la quantitat de tecles polsades en el test actual.
- Escenari 2: L'usuari veu en temps real la quantitat de tecles polsades en total.
- Escenari 3: L'usuari veu una estadística un cop resolta la kata.
- Escenari 4: L'usuari veu una puntuació un cop resolta la kata.

Desenvolupament.

En aquesta tercera iteració, no es necessita backend ni persistència de dades.

Tenint en compte que es segueix la metodologia BDD i ja s'han escrit els escenaris necessaris, cal seguir els següents passos:

1. Escriure el test, executar-lo i veure que falla.
2. Desenvolupar el codi mínim imprescindible perquè el test que acabem d'escriure passi.
3. Netejar el codi.

Desplegament de la nova versió de l'aplicació

En finalitzar la iteració, tenim una nova versió de l'aplicació on hi ha dues pàgines.

- Pàgina inicial: L'usuari pot resoldre les kates i tenir constància de la quantitat de caràcters modificats.
- Pàgina amb estadística: L'usuari pot veure una estadística amb l'evolució de la resolució de la kata. Aquesta estadística es fa en funció de la quantitat de caràcters modificats en cada test.

En haver configurat les github actions, en el moment de fer el "merge" amb la branca main, el desplegament es fa de manera automàtica.

6.4 Quarta iteració

Desenvolupar, verificar i validar:

Escriure històries d'usuari per la funcionalitat a desenvolupar

S'han escrit quatre històries d'usuari:

- COM usuari de l'aplicació VULL poder crear-me un compte PER A poder tenir un compte a l'aplicació
- COM usuari de l'aplicació VULL poder iniciar sessió PER A poder accedir a l'aplicació.

Escriure els diferents escenaris seguint la metodologia BDD

Partint de les històries d'usuari, s'han escrit dos escenaris:

- Escenari 1: L'usuari es vol crear un compte
    - Exemple: L'usuari es crea un compte de manera satisfactòria.
    - Contra exemple: L'usuari no introdueix totes les dades necessàries
    - Contra exemple: L'usuari vol crear-se un compte amb un correu ja existent
- Escenari 2: L'usuari vol iniciar sessió.
    - Exemple: L'usuari inicia sessió de manera satisfactòria.
    - Contra exemple: L'usuari no introdueix totes les credencials necessàries
    - Contra exemple: L'usuari introdueix malament les credencials.

Desenvolupament de la nova versió funcional de l'aplicació.

En aquesta quarta iteració, és necessari afegir persistència. Com s'explica en el capítol Bases de dades, s'utilitza mongodb com a base de dades.

Tenint en compte que es segueix la metodologia BDD i ja s'han escrit els escenaris necessaris, cal seguir els següents passos tant pel backend com pel frontend:

1. Escriure el test, executar-lo i veure que falla.
2. Desenvolupar el codi mínim imprescindible perquè el test que acabem d'escriure passi.
3. Netejar el codi.

Desplegament

En finalitzar la iteració, tenim una nova versió de l'aplicació on hi ha tres pàgines més:

- Pàgina inici: L'usuari pot veure una pàgina on es resumeix l'aplicació. D'altra banda, pot accedir a Sign up i Sign in.
- Pàgina Sign Up: L'usuari pot crear-se un compte
- Pàgina Sign In: L'usuari pot iniciar sessió al seu compte

S'ha afegit una nova configuració tant en el backend com en el backend perquè el desplegament es faci de manera satisfactòria. És necessari la configuració de la connexió de la base de dades i les crides a l'API.

Un cop afegida la configuració, s'ha fet el "merge" amb la branca main de manera que s'ha fet el desplegament automàticament.

6.5 Cinquena iteració

Desenvolupar, verificar i validar

Escriure històries d'usuari per la funcionalitat a desenvolupar

S'han escrit quatre històries d'usuari:

- COM usuari de l'aplicació VULL poder escollir entre diferents kates PER A poder resoldre-la.
- COM usuari de l'aplicació VULL poder saber la dificultat de cada kata PER A poder escollir la kata segons el nivell.
- COM usuari de l'aplicació VULL poder saber el llenguatge de programació de la kata PER A poder escollir millor la kata a resoldre.

Escriure els diferents escenaris segons BDD

Partint de les històries d'usuari, s'han escrit dos escenaris:

- Escenari 1: L'usuari consulta les diferents kates que es poden resoldre
  - Exemple: L'usuari consulta la llista de kates.
- Escenari 2: L'usuari escull la kata que vol resoldre
  - Exemple: L'usuari escull una kata de la llista de kates.

Desenvolupament de la nova versió funcional de l'aplicació

Creació de noves kates.

S'han plantejat tres kates noves, de manera que, l'usuari té la possibilitat d'escollir entre quatre kates diferents.

- PrimeFactors: funció què a partir d'un número, calcular els nombres primers d'aquest.
- Factorial: funció què a partir d'un número, calcular el factorial d'aquest.
- FizzBuzz: funció que retorna el valor entrat, substituint els múltiples de 3 per "Fizz", els múltiples de 5 per "Buzz" i els múltiples tant de 3 com de 5 per "FizzBuzz"
- LeapYear: funció què a partir d'un any, retorna si aquest és any de traspàs.

Per cadascuna de les kates, s'han plantejat els tests perquè l'usuari vagi aprenent a mesura que van passant els tests.

Es pot consultar els tests de cada kata en *l'Annex XXI. Kates.*

Tenint en compte que es segueix la metodologia BDD i ja s'han escrit els escenaris necessaris, cal seguir els següents passos tant pel backend com pel frontend:

1. Escriure el test, executar-lo i veure que falla.
2. Desenvolupar el codi mínim imprescindible perquè el test que acabem d'escriure passi.
3. Netejar el codi.

Desplegament

En finalitzar la iteració, tenim una nova versió de l'aplicació on hi ha una nova pàgina:

- Pàgina amb les kates: En aquesta nova pàgina, es mostra a l'usuari les diferents kates que pot resoldre. En cadascuna de les kates es mostra la dificultat i el llenguatge de programació

S'ha hagut de modificar les Github Actions per a poder fer el desplegament.

- Els tests del backend fan crides a l'API per poder testejar el seu comportament. És per això que per a passar els tests el backend ha d'estar executant-se.
- Els tests de backend utilitzen les bases de dades. Les diferents crides a l'API afegeixen i recullen informació de les bases de dades. És per això que és necessari tenir una instància de mongoDB executant-se per a passar els tests.

Es pot consultar la nova versió de les Github Actions en *l'Annex III.I. Iteració 6: Modificació Github Actions.*

6.6 Sisena iteració

Desenvolupar, verificar i validar

Escriure històries d'usuari per la funcionalitat a desenvolupar

S'han escrit dues històries d'usuari:

- COM usuari de l'aplicació VULL que es guardi un registre de les kates que he resolt PER A poder consultar-les.
- COM usuari de l'aplicació VULL consultar els resultats de les kates resoltes anteriorment PER A poder veure la meva evolució.

Escriure els diferents escenaris segons BDD

Partint de les històries d'usuari, s'han escrit dos escenaris:

- Escenari 1: L'usuari finalitza una kata
  - Exemple: L'usuari finalitza una kata i es guarda la solució i la data.
- Escenari 2: L'usuari vol consultar el registre de les kates resoltes anteriorment.
  - Exemple: L'usuari consulta el registre d'una kata.
  - Contra exemple: L'usuari no pot consultar el registre de kates que no ha resolt.

Desenvolupament de la nova versió funcional de l'aplicació

Tenint en compte que es segueix la metodologia BDD i ja s'han escrit els escenaris necessaris, cal seguir els següents passos tant pel backend com pel frontend:

1. Escriure el test, executar-lo i veure que falla.
2. Desenvolupar el codi mínim imprescindible perquè el test que acabem d'escriure passi.
3. Netejar el codi.

Anàlisi dels possibles resultats de cada kata

Per a fer l'anàlisi de les kates, s'han resolt cadascuna de les kates tenint en compte tres casos:

- Seguint correctament TDD
- Generalitzant en excés, és a dir, trobant la solució abans d'hora.
- No seguint TDD. Per simular no seguir TDD es resol la kata resolent cada test de manera individual amb condicionals.

En les següents imatges es mostren les estadístiques obtingudes de cadascuna de les kates. Com podem observar, no existeix cap patró de solució entre elles.

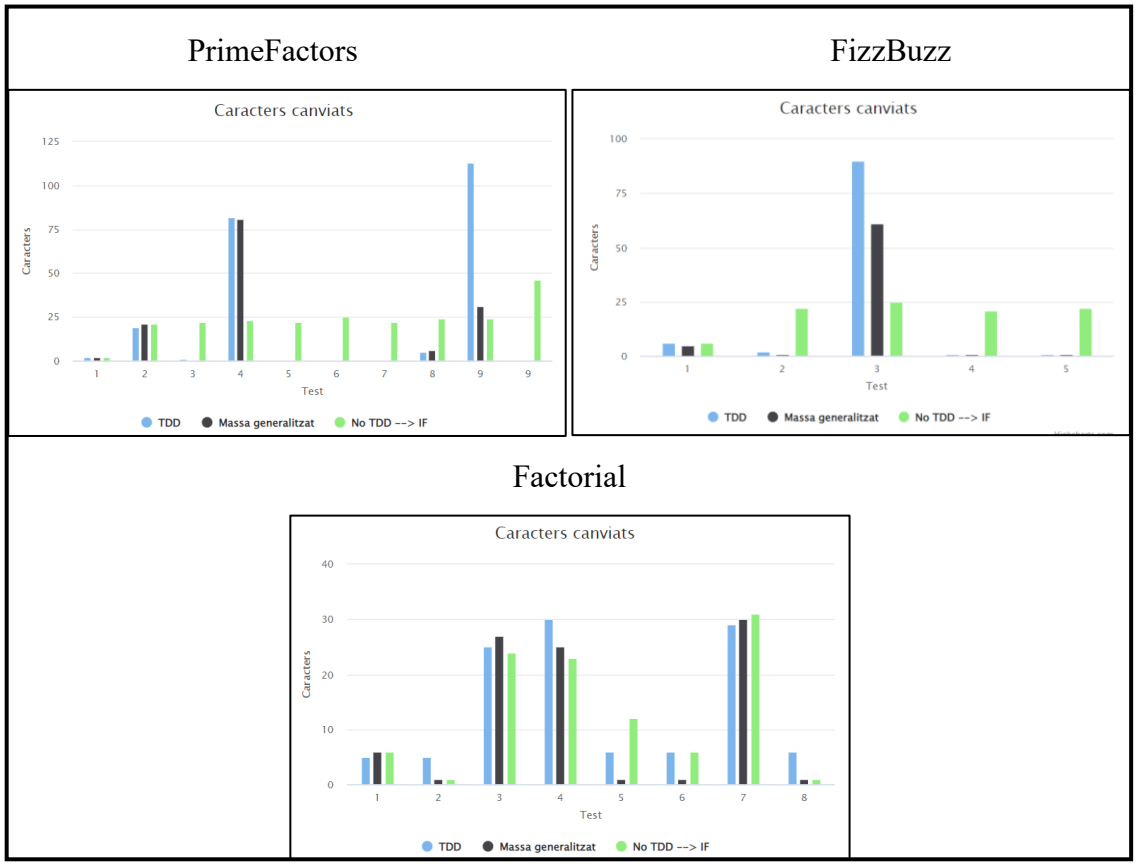

Fig. 6.4 Anàlisi kates. Font: pròpia, 2022

No es pot concloure cap patró a partir d'aquestes tres estadístiques. Per a poder crear un algoritme genèric per avaluar la resolució d'una kata seguint la metodologia TDD, és necessari un major volum major d'usuaris i de kates a resoldre.

A partir de la solució considerada correcta de cada kata, s'ha creat un algoritme per avaluar la resolució de les kates. Aquest algoritme avalua per cada test la diferència que

hi ha entre els caràcters modificats per l'usuari i els caràcters esperats. Es té en compte que l'usuari pot equivocar-se a l'hora d'escriure, és per això que es dóna un marge de 5 caràcters.

Desplegament

En finalitzar la iteració, tenim una nova versió de l'aplicació on s'ha modificat la pàgina on es mostra la llista amb les diferents kates a resoldre i s'ha afegit una nova pàgina:

- Pàgina amb les kates: S'afegeix un botó per a poder consultar les solucions anteriors de cada kata. En cas que no hi hagi cap report, el botó no fa res.
- Pàgina reports: Mostra les estadístiques de les solucions fetes anteriorment i la data en què es va resoldre cada kata.

En haver configurat les github actions, en el moment de fer el "merge" amb la branca main, el desplegament es fa de manera automàtica.

6.7 Setena iteració

Desenvolupament

Escriure històries d'usuari per la funcionalitat a desenvolupar

S'han escrit tres històries d'usuari:

- COM usuari de l'aplicació VULL veure una puntuació de la kata en el moment de finalitzar-la PER A poder saber el marge de millora.
- COM usuari de l'aplicació VULL poder veure en els reports la puntuació treta PER A saber la meva evolució.
- COM usuari de l'aplicació VULL poder veure un ranking amb tots els usuaris PER A poder veure l'evolució general.

Escriure els diferents escenaris seguint la metodologia BDD

Partint de les històries d'usuari, s'han construït dos escenaris diferents:

- Escenari 1: L'usuari finalitza una kata i obté una puntuació
    - Exemple: L'usuari resol la kata Factorial seguint correctament TDD.
    - Exemple: L'usuari resol la kata Factorial generalitzant en excés.
    - Exemple: L'usuari resol la kata Factorial sense seguir TDD
    - Exemple: L'usuari resol la kata Prime Factors seguint correctament TDD.
    - Exemple: L'usuari resol la kata Prime Factors generalitzant en excés.
    - Exemple: L'usuari resol la kata Prime Factors sense seguir TDD
    - Exemple: L'usuari resol la kata FizzBuzz seguint correctament TDD.
    - Exemple: L'usuari resol la kata FizzBuzz generalitzant en excés.
    - Exemple: L'usuari resol la kata FizzBuzz sense seguir TDD
- Escenari 2: L'usuari consulta el ranking.
    - Exemple: L'usuari consulta el ranking de puntuacions dels usuaris.

Desenvolupament

Tenint en compte que es segueix la metodologia BDD i ja s'han escrit els escenaris necessaris, cal seguir els següents passos tant pel backend com pel frontend:

1. Escriure el test, executar-lo i veure que falla.
2. Desenvolupar el codi mínim imprescindible perquè el test que acabem d'escriure passi.
3. Netejar el codi.

Desplegament

En finalitzar la iteració, tenim una nova versió de l'aplicació on s'ha afegit la possibilitat de consultar un ranking dels diferents usuaris:

- Pàgina ranking: S'afegeix la possibilitat de consultar un ranking on apareixen tots els usuaris. La puntuació es basa en la suma de la màxima puntuació obtinguda de cadascuna de les kates resoltes. Es mostra el correu electrònic de cada usuari, ja que la finalitat és que els usuaris es puguin promocionar.

En haver configurat les github actions, en el moment de fer el "merge" amb la branca main, el desplegament es fa de manera automàtica.

7. Evolució pantalles

En aquest capítol es mostra l'evolució de les pantalles a mesura que passen les iteracions.

7.1 Primera iteració

En finalitzar la primera iteració no s'obté cap pantalla, ja que no es desenvolupa.

7.2 Segona iteració

L'objectiu de la segona iteració és crear la lògica per a resoldre kates. En finalitzar la iteració s'obté una única pantalla:

En la següent imatge *Fig.7.1*, es mostra com inicialment apareix un test i una funció buida. Com el test no s'ha resolt i aquest no passa, està de color vermell i no podem veure el següent.

Fig.7.1. Segona iteració, usuari no passa el test. Font: pròpia, 2022.

En la següent imatge *Fig.7.2*, es mostra com en el moment en què s'escriu el codi per a passar el test, aquest canvia a color verd. En el moment en què passen tots els tests, apareix botó 'Next', per a obtenir el següent test.

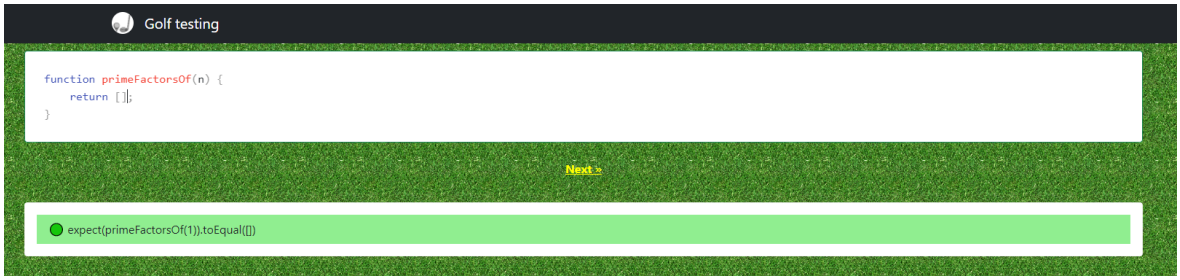

Fig.7.2. Segona iteració, usuari passa el test. Font: pròpia, 2022.

En la següent imatge *Fig.7.3*, es mostra com en el moment en què es polsa al botó 'Next', apareix el següent test a passar. D'altra banda, podem veure com el primer test passa i el segon no. És necessari que tots els tests passin per poder accedir al següent.

Fig.7.3. Segona iteració, usuari accedeix al següent test. Font: pròpia, 2022.

En la següent imatge *Fig.7.4*, es mostra com a partir dels tests, s'ha trobat un algoritme que resol tots els tests.

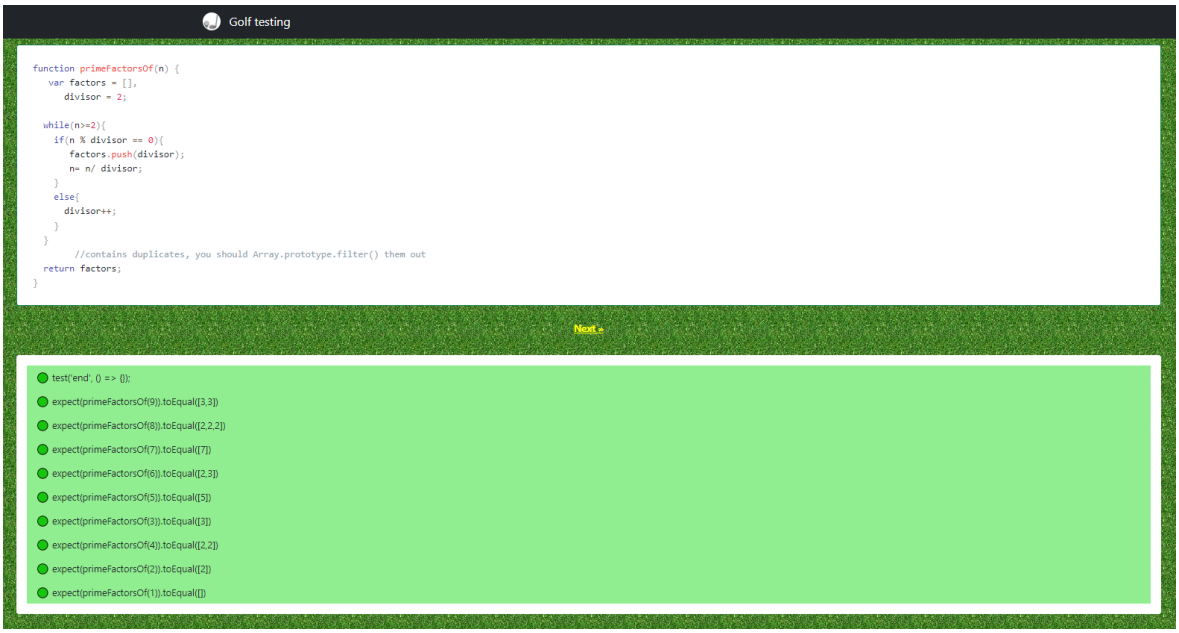

Fig.7.4. Segona iteració, usuari finalitza una kata. Font: pròpia, 2022.

7.3 Tercera iteració

L'objectiu de la tercera iteració és crear la lògica per a mostrar una estadística en el moment de finalitzar la kata. D'altra banda, també s'afegeixen dos comptadors perquè l'usuari sàpiga quants caràcters ha modificat en el test actual i en total. En finalitzar la iteració s'obtenen dues pantalles.

En la següent imatge *Fig.7.5*, es mostra com igual que en l'anterior iteració, apareix una pàgina per a resoldre un test. Tot i això, ara podem veure un marcador que indica la quantitat de caràcters polsats en el test actual i en tots els tests.

Fig.7.5. Tercera iteració, usuari veu comptadors. Font: pròpia, 2022

En la següent imatge *Fig. 7.6*, es mostra com a mesura que es polsen caràcters, ambdós comptadors s'actualitzen.

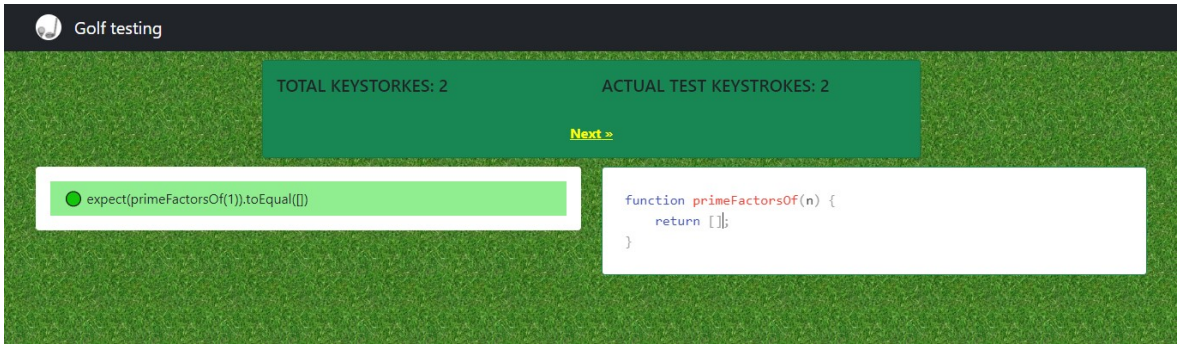

Fig. 7.6. Tercera iteració, comptadors s'actualitzen. Font: pròpia, 2022

En la següent imatge *Fig. 7.7*, es mostra com en finalitzar la kata, s'han fet 233 modificacions. En el test actual, en canvi, no s'ha fet cap modificació.

Fig. 7.7. Tercera iteració, usuari finalitza la kata. Font: pròpia, 2022

En la següent imatge *Fig. 7.8*, es mostra com en finalitzar la kata, es mostra una estadística de columnes. Aquesta estadística mostra per cada test, la quantitat de caràcters que s'han modificat.

Fig. 7.8. Tercera iteració, usuari consulta estadística. Font: pròpia, 2022

7.4 Quarta iteració

L'objectiu de la quarta iteració és crear la lògica perquè l'usuari pugui crear-se un compte i pugui iniciar sessió. En finalitzar la iteració s'obtenen dues pantalles.

En la següent imatge *Fig. 7.9*, es mostra la nova pàgina de benvinguda. Quan l'usuari accedeix a l'aplicació, com que no ha iniciat sessió, veu la pàgina de benvinguda on hi ha un resum de l'aplicació.

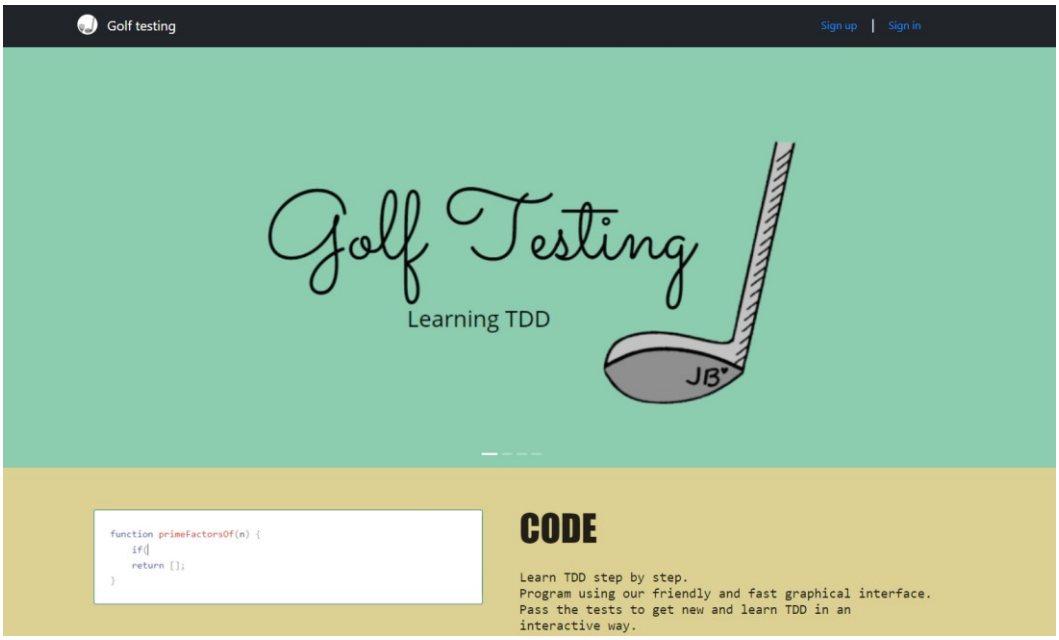

Fig. 7.9. Quarta iteració, pàgina benvinguda. Font: pròpia, 2022

En la següent imatge *Fig. 7.10*, s'observa la pàgina per a crear-se un compte. En cas d'introduir malament les dades, apareix un missatge d'error. En cas contrari, crea el compte i redirigeix a l'usuari a resoldre la kata.

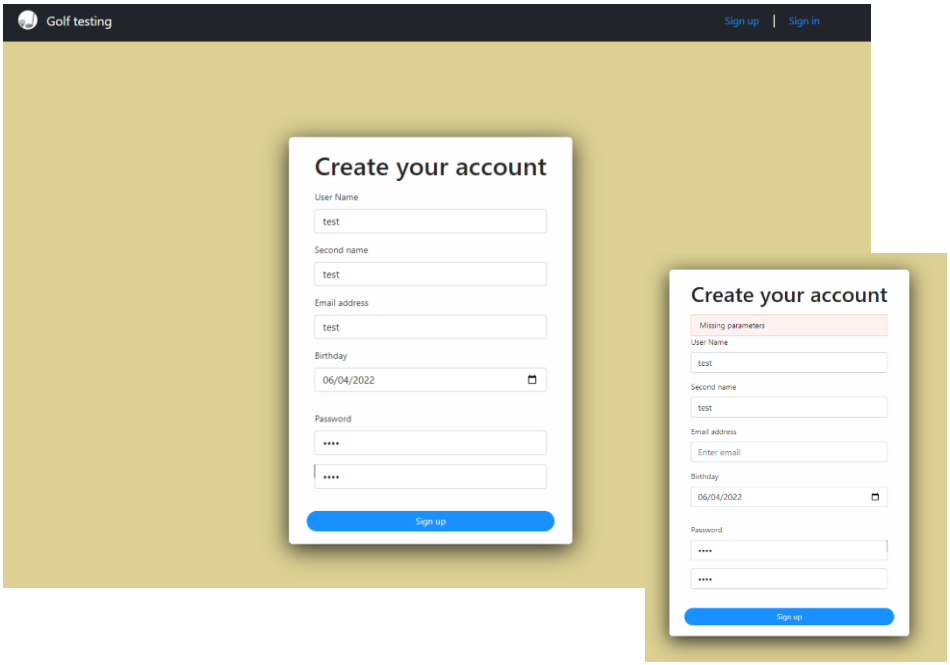

Fig. 7.10. Quarta iteració, pàgina crear compte. Font: pròpia, 2022.

En la següent imatge Fig.7.11, es mostra la pàgina per iniciar sessió. En cas d'introduir malament les dades, apareix un missatge d'error. En cas contrari, crea el compte i redirigeix a l'usuari a resoldre la kata.

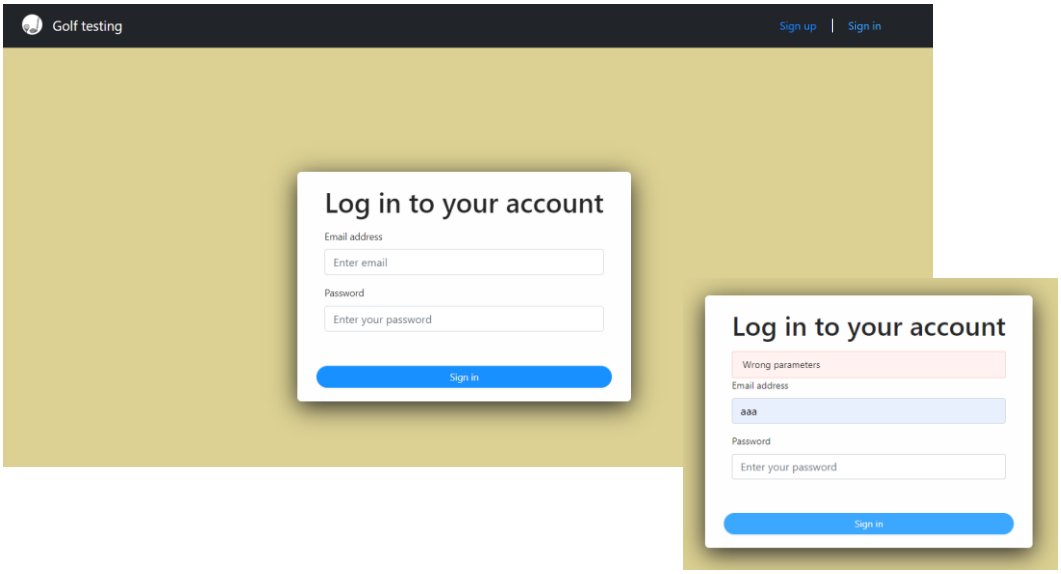

Fig.7.11. Quarta iteració, pàgina iniciar sessió. Font: pròpia, 2022

En la següent imatge *Fig.7.12*, es mostra la pàgina per resoldre la kata. S'ha canviat l'estil de la pàgina perquè sigui més minimalista i intuïtiu.

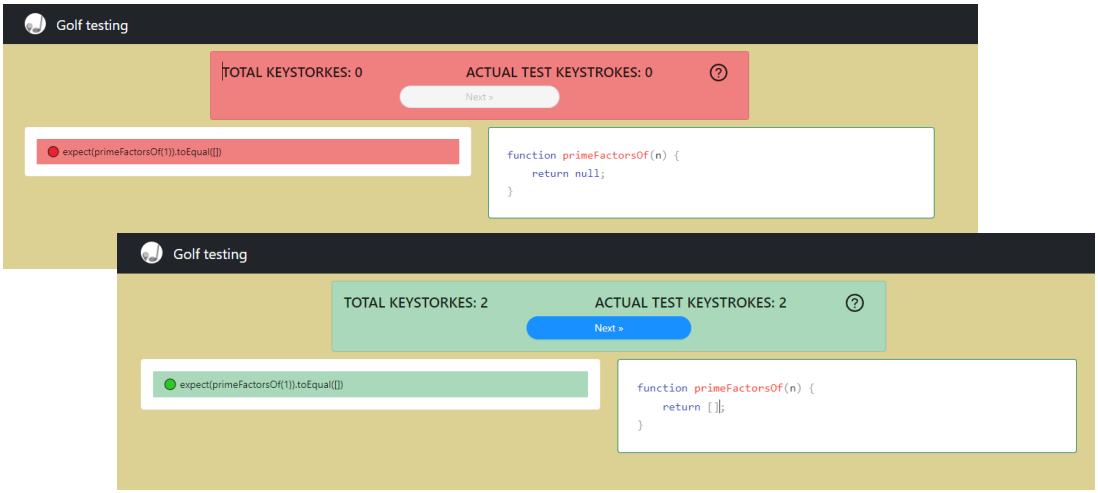

Fig.7.12. Quarta iteració, pàgina resoldre kata. Font: pròpia, 2022

En la següent imatge *Fig. 7.13*, es mostra la pàgina on es mostren l'estadística de la resolució de la kata. S'ha canviat l'estil de la pàgina perquè sigui més minimalista i intuïtiu.

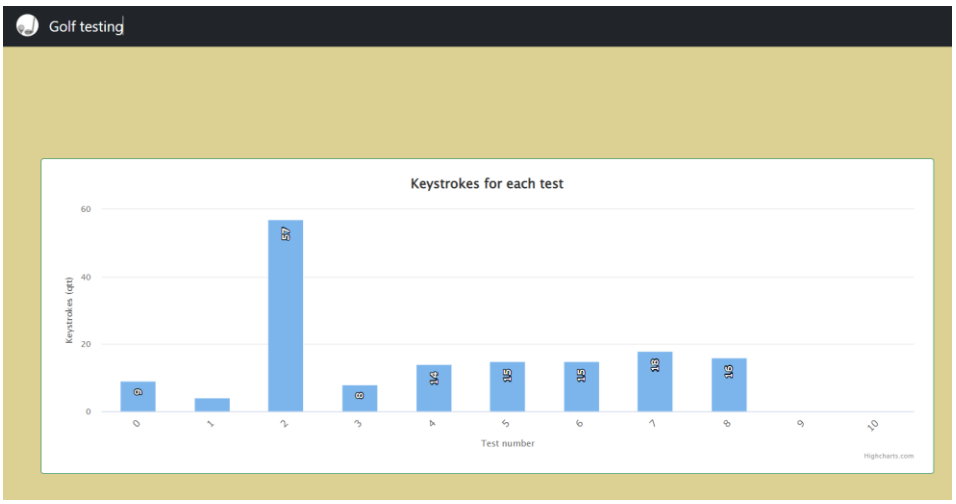

Fig. 7.13. Quarta iteració, pàgina estadística. Font: pròpia, 2022

7.5 Cinquena iteració

L'objectiu de la cinquena iteració és afegir la possibilitat d'escollir entre diferents kates. En finalitzar la iteració s'obté una nova pàgina.

En la següent imatge *Fig.7.14*, es mostra la pàgina on es mostren les diferents kates que es poden resoldre. L'usuari pot escollir entre diferents kates proposades, les quals estan etiquetades per nivell de dificultat i per llenguatge de programació.

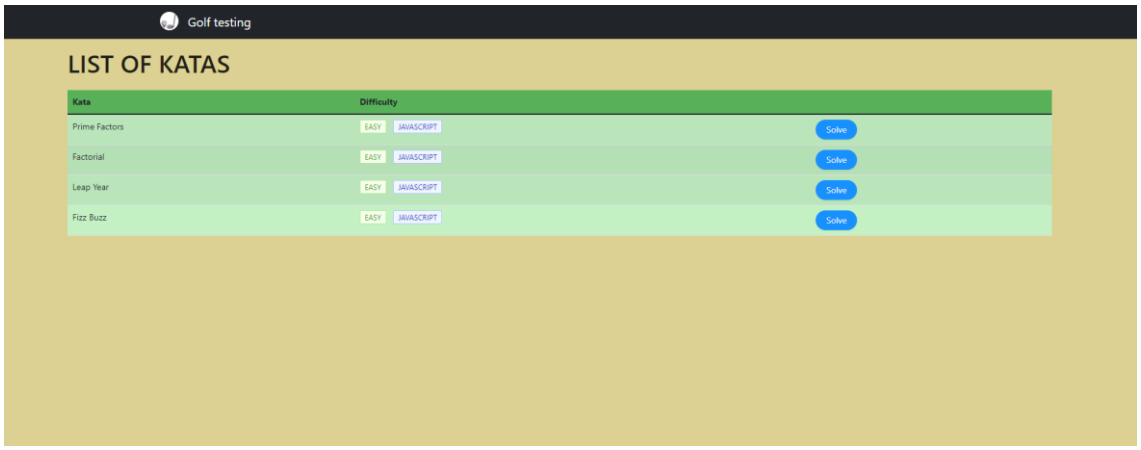

Fig.7.14 Cinquena iteració, pàgina llista kates. Font: pròpia, 2022

En la següent imatge *Fig.7.15*, es mostra pàgina per resoldre kates. Segons la kata que ha escollit l'usuari a resoldre canvien tant els tests, com la funció a resoldre.

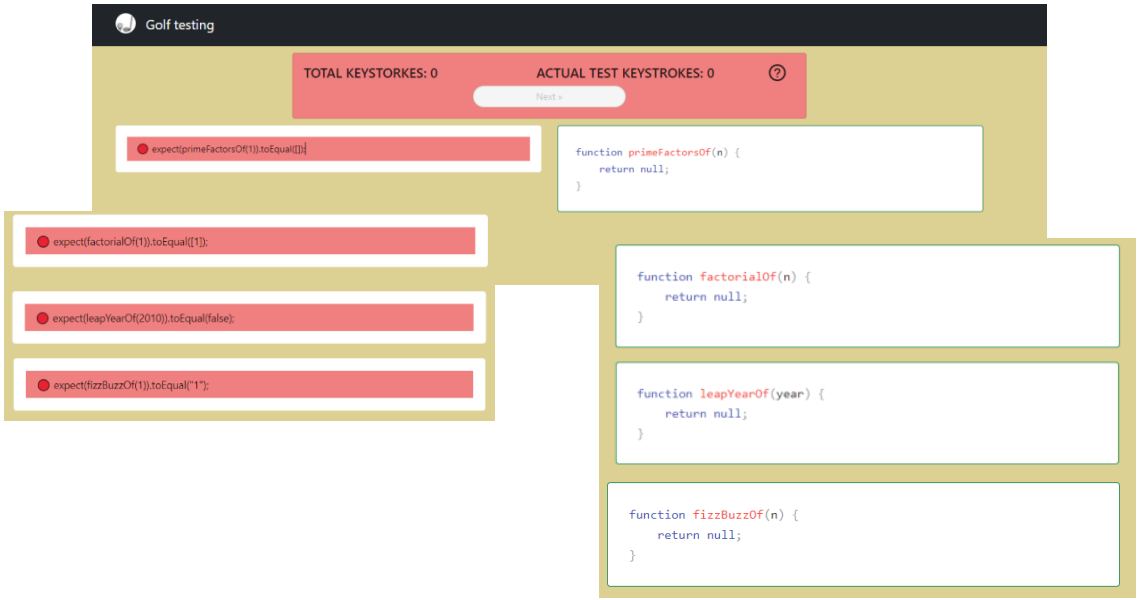

Fig.7.15 Cinquena iteració, pàgina arena. Font: pròpia: 2022

7.6 Sisena iteració

L'objectiu de la sisena iteració és desenvolupar la lògica per a crear un registre de cada usuari de les kates resoltes. D'altra banda, es fa un anàlisi dels possibles resultats de cada kata, per a posteriorment poder crear un criteri de puntuació.

En la següent imatge Fig. 7.16, es mostra la pàgina on es mostren les diferents kates que es poden resoldre. S'ha afegit un botó per a accedir als records anteriors de cada kata, és a dir, l'usuari pot consultar les seves antigues solucions.

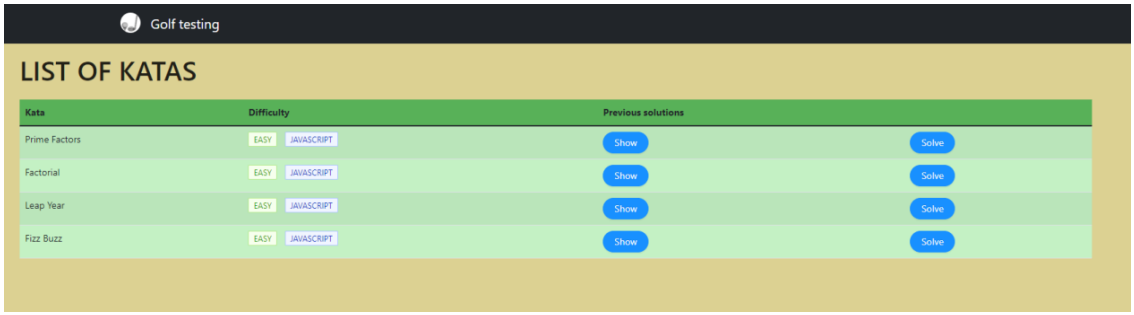

Fig. 7.16 Sisena iteració, pàgina llista kates. Font: pròpia, 2022

En la següent imatge Fig.7.17, es mostra la pàgina on es veuen els records anteriors d'una kata en concret. De cada kata, l'usuari pot veure la data de resolució i les estadístiques. D'aquesta manera, l'usuari pot veure la seva evolució.

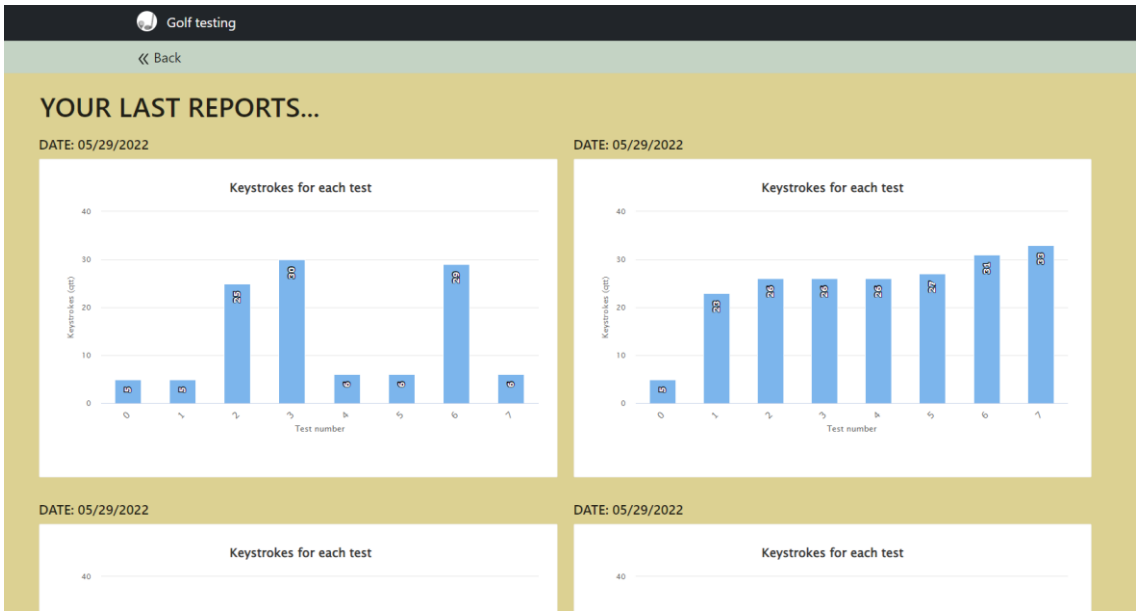

Fig.7.17 Sisena iteració, pàgina estadistiques. Font: pròpia, 2022

7.7 Setena iteració

L'objectiu de la setena iteració és desenvolupar la lògica per a puntuar als usuaris un cop finalitzin la kata. La puntuació es fa a partir del anàlisi fet en l'anterior iteració. D'altra banda, es desenvolupa la lògica perquè els usuaris puguin veure rankings. La puntuació es basa en la suma de la màxima puntuació obtinguda de cadascuna de les kates resoltes. Es mostra el correu electrònic de cada usuari, ja que la finalitat és que els usuaris es puguin promocionar.

En la següent imatge *Fig. 7.2018*, es mostra la pàgina amb l'estadística un cop finalitzada la kata. S'ha afegit una qualificació segons la quantitat de caràcters. D'aquesta manera s'incentiva a l'usuari a fer un correcte ús del TDD.

Fig. 7.18 Sisena iteració, pàgina estadìstica. Font: pròpia, 2022

En la següent imatge *Fig. 7.2019*, es mostra la pàgina amb els registres d'una kata. S'ha actualitzat i s'ha afegit la puntuació obtinguda en cada kata. D'aquesta manera l'usuari veu més clarament l'evolució i quines són les millors solucions fetes.

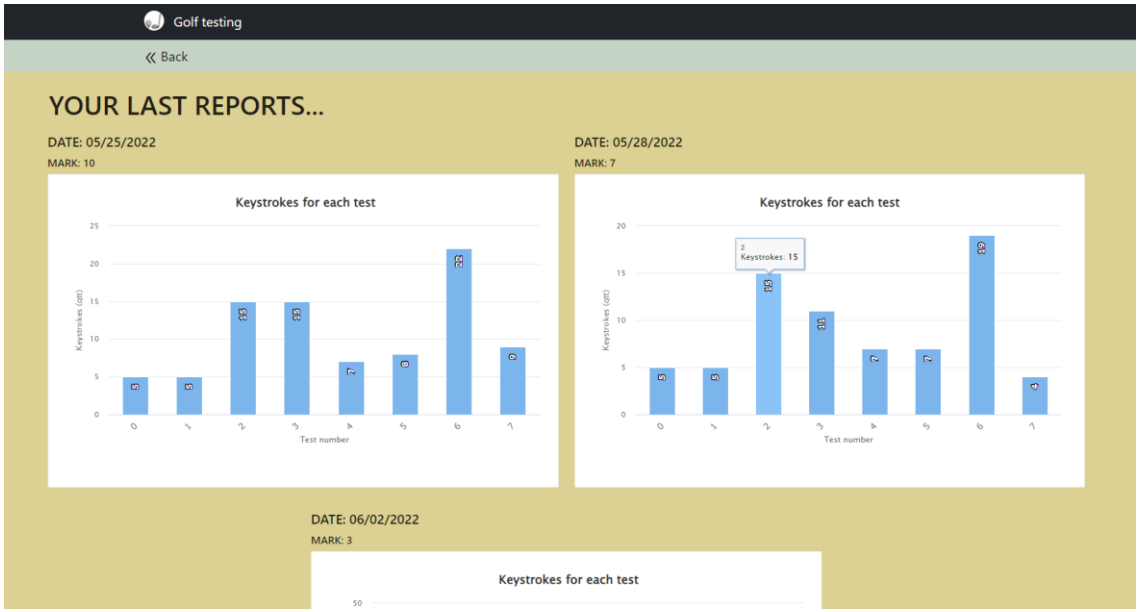

Fig. 7.19 Sisena iteració, pàgina records. Font: pròpia 2022

En la següent imatge *Fig. 7.20*, es mostra la pàgina on es mostren les diferents kates que es poden resoldre. S'ha afegit un botó en forma de copa per a poder accedir al ranking d'usuaris.

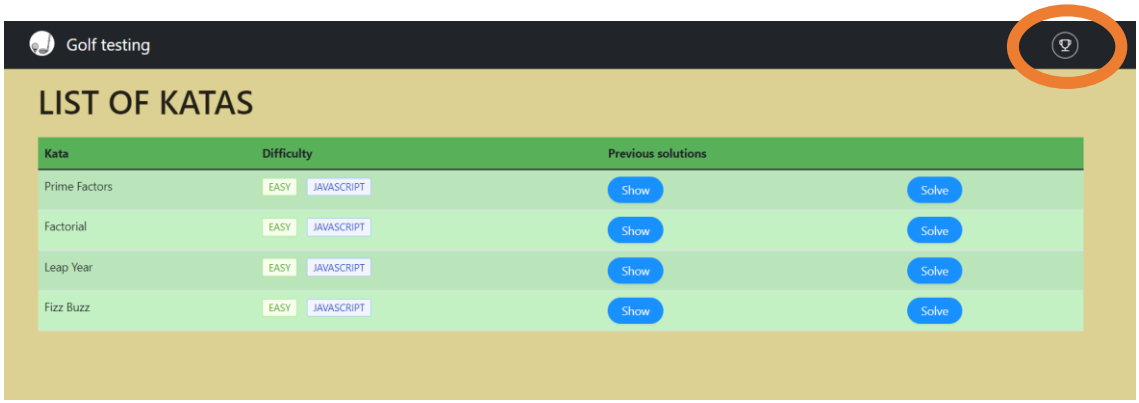

Fig. 7.20 Setena iteració, pàgina llista kates. Font: pròpia, 2022

En la següent imatge Fig. 7.21, es mostra la pàgina del ranking. De cada usuari es mostra la posició global en el ranking, l'email i els punts totals.

Es mostra l'email, ja que la finalitat del ranking és que els usuaris es puguin promocionar. És a dir, a partir del ranking els usuaris poden contactar entre ells.

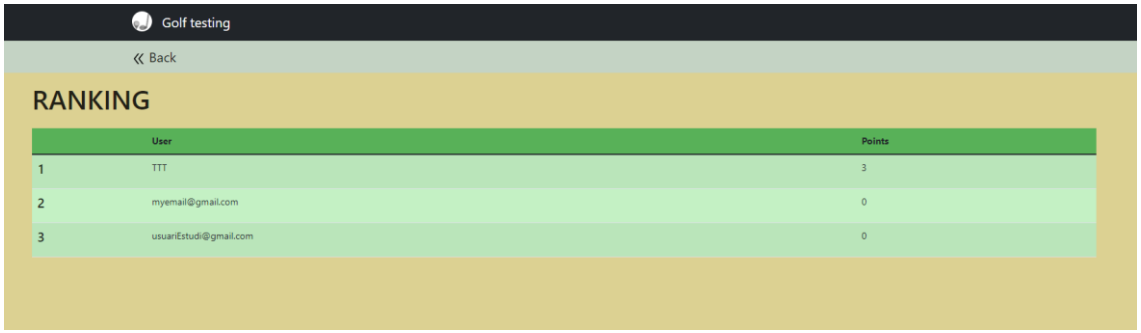

Fig. 7.21 Setena iteració, pàgina ranking. Font: pròpia: 2022

7.8 Demo

Es pot consultar una demo de l'aplicació on es veu el resultat final, en *Annex XXIV. Demo de l'aplicació.*

8. Arquitectura

En aquest capítol es mostra l'arquitectura obtinguda un cop finalitzades les iteracions.

El projecte utilitza el patró MVC (Model Vista Controlador), que és un patró de disseny utilitzat per implementar interfícies d'usuari, dades i lògica de control. Aquest patró està caracteritzat per separar la lògica de negoci i les interfícies d'usuari. D'aquesta manera, s'aconsegueix una millor divisió de treball i facilita el manteniment del projecte. El patró MVC compost de tres parts:

- Model: Gestiona les dades i la lògica de negoci.
- Vista: Encarregat del disseny i la presentació a l'usuari.
- Controlador: Intermediari entre el model i la vista.

En la següent imatge es mostra el flux de comunicació entre aplicacions:

Fig. 8.1 Flux de comunicació entre aplicacions. Font: pròpia, 2022

8.1 Backend

L'arquitectura de paquets utilitzada en l'aplicació de backend segueix el patró capes. El patró capes és un model de desenvolupament que ens permet separar les parts que componen una aplicació. El patró capes està dividit en les següents capes:

- **Presentació**: és el responsable de gestionar les peticions del client.
  Aquestes classes utilitzen la notació `@RestController` ja que contenen les peticions de l'API.

En aquest cas s'han creat tres classes controladores; KataRestController, ReportRestController i UserRestController, amb els següents endpoints:

- GET: /katas
- GET: /katas/{id}
- GET: /users?email={email}&&password={password}
- GET: /reports?email={email}&&kataId={kataId}
- GET: /reports/ranking
- POST: /users (el body de la petició HTTP conté el user)
- POST: /reports (el body de la petició HTTP conté el report)

- **Aplicació**: Serveix per coordinar, és a dir, redirigeix els objectes del domini i d'infraestructura (persistència, etc.), ja que són els que internament han de resoldre els problemes.

  Aquestes classes utilitzen la notació `@Service.`

  En aquest cas s'han creat tres classes de servei; KataServiceIml, ReportServiceIml i UserServiceIml.

- **Domini**: Classes que contenen la lògica de domini, és a dir, les regles de negoci.

  Aquestes classes utilitzen la llibreria Spring Data MongoDB i es declaren amb la notació `@Document(collection = "<nom de la col·lecció>"`.

  En aquest cas s'han creat quatre classes de domini; Kata, Test, Report i User.

- **Persistència**: Intermediari entre el controlador i la base de dades.

  Són interfícies que utilitzen la llibreria Spring Data MongoDB, és per això que estenen de la interfície MongoRepository.

  En aquest cas s'han creat tres interfícies; KataRepository, ReportRepository i UserRepository.

- **Útils**: Van totes les classes que van a qualsevol capa; constants, excepcions genèriques...

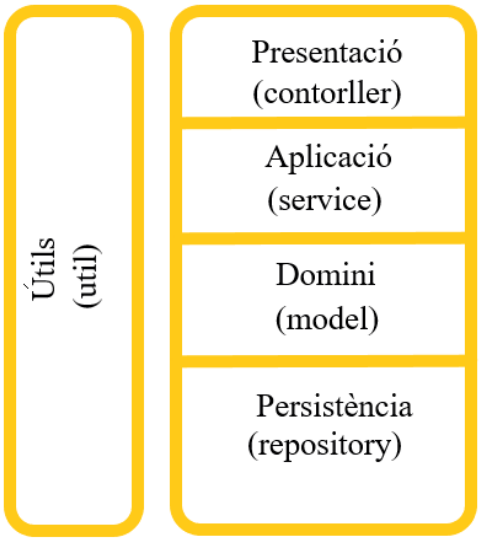

Fig. 8.2 Patró capes. Font: pròpia, 2022.

L'arquitectura de paquets utilitzada en l'aplicació de backend és la següent:

```
src
└── main
|   └── java/server/tcm/tfg
|   |   └── contorller
|   |   |   └── frontendException
|   |   └── dto
|   |   └── model
|   |   └── repository
|   |   └── service
|   |   |   └── exceptions
|   |   |   └── impl
|   |   └── util
|   └── resources
└── test
    └── java/server/tcm/tfg
        |   └── steps
        |   |   └── utils
        └── resources/features
```

Es pot consultar l'arquitectura incloent totes les classes, en *Annex XXII. Arquitectura backend completa.*

8.2 Frontend

Redux és una llibreria JavaScript de codi obert per la gestió de l'estat de les aplicacions. Està compost per:

- **Action**: Les accions són objectes JSON simples amb un camp amb un valor de cadena anomenat tipus.
  Les accions s'envien al *store* a partir de `store.dispatch()`.
- **Reducer**: els reducers són funcions responsables de crear el següent *state* donat l'estat actual i una acció.
- **Store**: el *store* és un objecte que reuneix les accions i els reducers; conté l'estat actual de l'aplicació.

React és un framework de JavaScript que permet crear components. Aquests components són funcions que contenen codi HTML reutilitzable.

React es comunica amb les diferents funcionalitats de Redux per gestionar l'estat. Els diferents components de React es comuniquen amb el Store de Redux a partir d'accions de l'usuari. Quan l'usuari duu a terme una acció, fent ús de `store.dispatch()` s'actualitza l'estat del store.

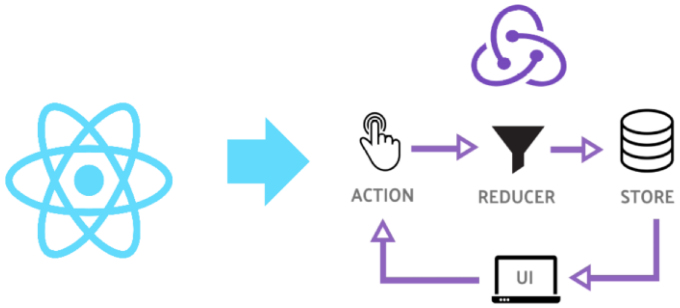

Fig. 8.3 Comunicació React Redux. Font: pròpia: 2022

L'arquitectura de paquets utilitzada en l'aplicació de frontend és la següent:

```
src
└── __test__
    └── features
    └── mocks
    └── steps
└── assets/images
└── components
└── constants
```

```
└── store
|   └── code
|   └── kata
|   └── report
|   └── step
|   └── test
|   └── user
└── test
```

Es pot consultar l'arquitectura incloent tots els fitxers, en *Annex XXIII. Arquitectura frontend completa.*

8.3 Base de dades

Com a base de dades, s'ha utilitzat MongoDB, que és un sistema de bases de dades no relacionals, orientat a documents i open source.

Per a la persistència de dades, s'ha necessitat crear tres documents:

- **Users**: per a guardar tota la informació relacionada amb cadascun dels usuaris.
- **Katas**: per a guardar tota la informació relacionada amb cadascuna de les kates.
- **Reports**: per a guardar tota la informació relacionada amb la solució d'un usuari a una kata.

Users

En la següent imatge es pot observar el document d'un usuari. Cada usuari te com a camps un ObjectId com a identificador, un nom, un cognom, un email i una contrasenya.

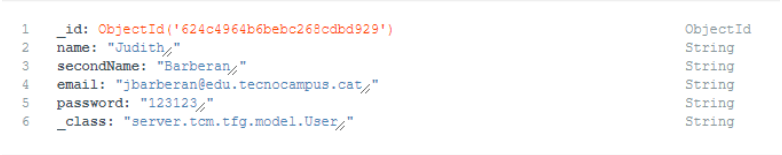

Fig. 8.4 Document Users. Font: pròpia, 2022

Katas

En la següent imatge es pot observar el document d'una kata. Cada kata té com a camps un ObjectId com a identificador, un array d'objectes amb cadascun dels tests i la seva

solució, el nivell de dificultat, el llenguatge de programació i el codi inicial que s'ofereix a l'usuari.

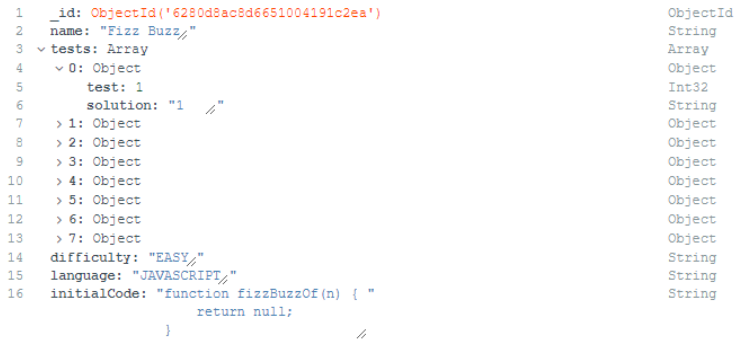

Fig. 8.5 Document Katas. Font: pròpia, 2022

Reports

En la següent imatge es pot observar el document d'un report. Cada report té com a camps un ObjectId com a identificador, l'email de l'usuari, l'identificador de la kata, la data de solució, un array amb la solució proposada i la nota obtinguda.

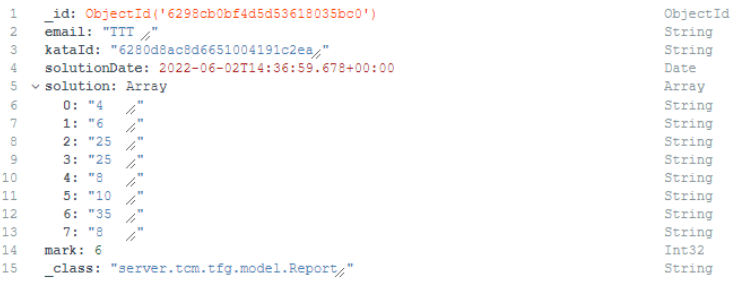

Fig. 8.6 Document Reports. Font: pròpia, 2022

9. Conclusions

En aquest apartat es valorarà el treball fet al llarg del desenvolupament del projecte.

Estat final del desenvolupament del projecte

L'estat final del desenvolupament del projecte és satisfactori tenint en compte tots els problemes que han anat sorgint durant aquest temps. Tot i només haver complert els objectius principals i no haver desenvolupat els secundaris, s'han complert cadascuna de les tasques plantejades seguint el mètode espiral i s'ha realitzat el desenvolupament seguint la metodologia BDD.

Valoració eines utilitzades

Heroku

El host que ofereix de manera gratuïta Heroku és senzill d'utilitzar en aplicacions d'una tecnologia. En el moment en què s'afageix més d'una tecnologia la versió gratuïta de Heroku no és el host més adequat, ja que obliga a adaptar l'arquitectura de paquets del projecte. Un altre inconvenient és que, la versió de gratuïta de Heroku s'apaga després de 30 minuts d'inactivitat, això és impensable per una aplicació en producció.

Github Actions

Les Github Actions han aportat un gran valor al projecte. Han permès saber en tot moment si els tests del projecte passaven i si el percentatge de coverage estava al 100%. D'aquesta manera s'ha evitat que es desplegués versions amb errors.

BDD

La metodologia BDD, ha facilitat el desenvolupament de l'aplicació. Plantejar els diferents escenaris per a posteriorment desenvolupar-los, ajuda a tenir una visió clara del comportament que ha de tenir l'aplicació. Això permet orientar els tests per aconseguir desenvolupar el codi necessari sense per a la funcionalitat a desenvolupar.

TDD

L'ús de TDD minimitza la quantitat del codi a escriure, ja que en fer proves, s'evita la possibilitat d'escriure codi que no s'utilitzarà. Permet un desenvolupament més ràpid i optimitzant el manteniment i l'evolució del codi. Tot i això, quan es comença a utilitzar TDD per primera vegada, escriure els tests en un ordre incorrecte pot comportar a grans complicacions en el moment de desenvolupar.

MongoDB

Al ser MongoDB una base de dades no relacional, ofereix flexibilitat en les dades. MongoDB s'ha adaptat a l'evolució de l'estructura de dades del projecte.

Valoració personal

El principal objectiu personal d'aquest projecte, era analitzar, aprendre i aplicar DBB i TDD correctament. S'ha complert les expectatives proposades tot i haver sigut més complex de l'inicialment plantejat.

D'altra banda, haver de resoldre tots els problemes sense l'ajuda d'un equip, ha complicat l'evolució de l'aplicació. Disposar d'un equip, facilita l'aprenentatge i la resolució de problemes, ja que entre els membres de l'equip es complementen.

10. Possibles ampliacions

En aquest capítol s'analitzen possibles millores i ampliacions del projecte de cara al futur. Cal destacar que, no s'han desenvolupat els objectius secundaris proposats per falta de temps. De manera que, aquests objectius també són possibles ampliacions.

Rols

Actualment, només existeix el rol d'usuari. S'ha de poder diferenciar entre quatre rols: organització, treballador d'una organització, autònom i administrador.

- **Organització/empresa**: Entitat interessada en què els seus treballadors aprenguin TDD per a poder treure benefici econòmic d'aquest coneixement.
  Les organitzacions han de poder consultar els resultats de tots els seus empleats, consultar rankings i fer anàlisis de l'evolució dels seus empleats.
- **Treballador** (usuari associat a una organització/empresa): Persona interessada en aprendre TDD per a poder aplicar els coneixements posteriorment en l'empresa.
  Els treballadors han de poder resoldre kates, consultar les seves solucions i veure rankings amb els altres usuaris que pertanyin a la mateixa organització.
- **Autònom**: Persona independent interessada en aprendre TDD per motius aliens a empreses.
  Els autònoms han de poder resoldre kates, consultar les seves solucions i veure els rankings amb tots aquells usuaris que no pertanyin a cap organització.
- **Administrador**: Persona que gestiona el contingut de l'aplicació.
  Ha de poder gestionar a les organitzacions, treballadors i autònoms. D'altra banda, ha de poder gestionar les kates, és a dir, ha de poder afegir-ne de noves i modificar i eliminar les existents.

Encriptació de contrasenyes

Actualment les contrasenyes dels usuaris es guarden a la base de dades sense encriptar. Aquest fet és una clara vulnerabilitat en l'aplicació, ja que en cas que hi hagi un atac, l'atacant tindria accés a totes les contrasenyes dels usuaris.

D'aquesta manera, l'encriptació de les contrasenyes en el moment de guardar-les a la base de dades, és una millora d'alta prioritat.

Creació de més kates i elavoració d'un algoritme més precís per avaluar als usuaris.

Actualment, les puntuacions de les resolucions de les kates, es fan de manera individual. És a dir, segons la kata que ha solucionat l'usuari, s'avalua d'una manera o un altre. Obtenint un major volum de dades de diferents usuaris, es podria fer un anàlisi més clar del patró que segueix TDD. D'aquesta manera, es podria desenvolupar un algoritme global.

Possibilitat de resoldre kates en diferents llenguatges de programació.

Actualment, totes les kates estan plantejades per ser resoltes en Javascript. Aprofitat que ja es mostra l'etiqueta de llenguatge, es podria donar la possibilitat de resoldre les kates en diferents llenguatges com: Java, Php, python, C++...

Web en diferents idiomes

Actualment, l'aplicació està en angles, ja que és l'idioma internacional. Una possible millora seria oferir als usuaris la possibilitat de canviar d'idioma.

Fòrum de preguntes

En el moment de finalitzar la kata a l'usuari li poden surgir dubtes com; com puc fer per millorar la meva solució? Hi ha alguna altra manera de resoldre la kata? He fet bé el plantejament?

Es podria crear un fòrum de preguntes perquè els usuaris de l'aplicació interactuessin i es complementessin coneixement.

Host

Actualment, s'utilitza com a host la versió gratuïta de Heroku. Aquest host té un clar desavantatge i és que s'apaga després de 30 minuts d'inactivitat. D'aquesta manera, quan un usuari es connecta i el host estava apagat, la web triga cert temps en carregar. Això pot fer crear desconfiança a l'usuari i decidir no entrar.

11. Bibliografia

[1] A. Fullen, K. Chennaian i S. Balasubramaniyan, “World Quality Report 2021-22” Capgemini, Sofeti i Micro Focus, 2021.

[2] A. Bertolino, “Software Testing Research: Achievements, Challenges, Dreams”, Istituto di Scienza e Tecnologie dell’Informazione, Pisa, Itàlia, Maig 2007.

[3] D. Rodenas, *The real reason to do Testing*, Gener 2022. [En línia] https://drpicox.medium.com/the-real-reason-to-do-testing-6f12b410dde3 [Últim accés: 03 Gener 2022]

[4] A. Tarlinder, *Developer testing: Building quality into software*, 2016, primera edició, ISBN: 0134291069

[5] W. Mwaura, *End-to-End Web Testing with Cypress: Explore techniques for automated frontend web testing with Cypress and JavaScript*, 2021, primera edició, pp. 93-93.

[6] D. North, *DAN NORTH & ASSOCIATES LTD*, Stickyminds, vol. 2006, núm. 03, 2006. [En línia] https://dannorth.net/introducing-bdd/ [Últim accés: 09 Febrer 2022].

[7] K. Beck *et al*, “Manifesto for Agile Software Development”, Ward Cunningham, 2001. [En línia] https://agilemanifesto.org/ [Últim accés: 22 Gener 2022]

[8] J. F. Smart, “Building software that makes difference” a *BDD in Action: Behavior-driven development for the whole software lifecycle*, 2015, pp. 3-31. ISBN: 161729165.

[9] Base de datos SQL vs. NoSQL: diferencias, ventajas y mitos, Ilimit [En línia] https://www.ilimit.com/blog/base-de-datos-sql-nosql/ [Últim accés: 16 Gener 2021]

[10] R. F. Córdova i B. E. Cuzco, “Análisis comparativo entre bases de datos relacionales con bases de datos no relacionales”, tesis universitària, Universitat Politècnica de Cuenca, 2013. [En línia] https://dspace.ups.edu.ec/bitstream/123456789/6977/1/UPS-CT003639.pdf [Últim accés: 09 Febrer 2022]

[11] G. Fariño, “Modelo Espiral de un proyecto de desarrollo de software”, Universitat Estatal de Milagro, Milagro, 2011. [En línia]

https://www.ojovisual.net/galofarino/modeloespiral.pdf [Últim accés: 09 Febrer 2022]

[12] "2021 Java Developer Productivity Report", JRebel, 2021.

[13] "Spring.io", 2021. [En línia] https://spring.io/ [Últim accés: 09 Febrer 2022]

[14] "Spring boot", 2021. [En línia] https://spring.io/projects/spring-boot [Últim accés: 08 Febrer 2022]

[15] "Oleksandra, Dmitry i Eugene, "Top JavaScript (JS) Trends to Watch in 2022", Març 2021 [En línia] https://www.codica.com/blog/top-javascript-trends/ [Últim accés: 09 Febrer 2022]

[16] "2021 state of the cloud report" ,Flexera, 2021

[17] "Redux", Febrer 2022 [En línia] https://es.redux.js.org/ [Últim accés: 10 Febrer 2022]

[18] "React Redux", "Why Use React Redux?" Febrer 2022 [En línia] https://.js.org/introduction/why-use-react-redux [Últim accés: 10 Febrer 2022]

[19] "2021 state of the cloud report" ,Flexera, 2021.

[20] "Heroku", 2021. [En línia] https://www.heroku.com/pricing [Últim accés: 10 Febrer 2022]

[21] Robert C. Martin Series, *Clean Craftsmanship: Disciplines, Standards, and Ethics,* 2021, primera edició, ISBN: 9780136915713

[22] Robert C. Martin, "Clean code 19 – Advanced TDD I" [En línia] https://cleancoders.com/episode/clean-code-episode-19-p1 [Últim accés: 15 Febrer 2022]

[23] Generalitat de Catalunya, "Gestió de riscos", 2020.

[24] "Github", Github Actions" Febrer 2022, [En línia] https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions [Últim accés: 15 Febrer 2022]

[25] "Cucumber", "Guia Cucumber" Febrer 2022, [En línia] https://cucumber.io/docs/guides/ [Últim accés: 17 Febrer 2022]

[26] "MongoDB", "Mongodb", Abril 2022, [En línia] https://www.mongodb.com/es [Últim accés: 01 Abril 2022]

[27] "Spring", "RestTemplete", Abril 2022, https://docs.spring.io/spring-framework/docs/current/javadoc-

api/org/springframework/web/client/RestTemplate.html [Últim accés: 01 Abril 2022]