Sistema de reserva i lloguer de pianos

Roger Muñoz Bernaus

Resum– La principal motivació d'aquest projecte és incentivar la participació ciutadana en la cultura a través de la gran xarxa d'Internet. Aquesta estrambòtica idea recau a desenvolupar un sistema de lloguers i reserves de pianos. El sistema contempla que els pianos estan situats en un espai, aquests espais són administrats per administradors d'espai i els usuaris realitzen reserves sobre els pianos per tal de poder anar-los a tocar. L'arquitectura del sistema de reserves consta, d'una banda, d'un *webservice* i, d'altra banda, d'un *backend* i *frontend*. En aquest projecte es durà a terme un disseny complet dels diferents mòduls que permetin assolir els requeriments; es valoraran les diferents alternatives d'implementar el disseny proposat; es farà la implementació de l'alternativa escollida; i finalment, es provarà la implementació desenvolupada. A cada pas es justificarà la idoneïtat de les decisions preses.

Paraules clau– Reserva, lloguer, *webservice*, *backend*, *frontend*, *Codeigniter*, *Flask*, Python, PHP, API, MySQL, Internet de les coses

Abstract– The main motivation of this project is to encourage citizen participation in culture through the Internet. This strange idea will materialize with the development of a system of rents and bookings for pianos. In the system, pianos are located in spaces, and these spaces are managed by space administrators. Users can book pianos in order to be able to play them. The architecture of the system is made, on the one hand, of a *webservice* and, on the other hand, a *backend* and *frontend*. This project will consist on doing the design of the different modules (that meet the specified requirements); evaluating the different alternatives for implementing the proposed design; implementing the chosen alternative; and finally, testing the developed implementation. On each of those steps, a justification of the suitability of the decisions taken will be provided.

Keywords– Booking, rent, webservice, backend, frontend, *Codeigniter*, *Flask*, Python, PHP, API, MySQL, Internet of Things.

1 Introducció

En els darrers anys, s'ha popularitzat el terme *Internet Of Things*, és a dir, "Internet de les Coses". Aquest concepte serveix per definir el concepte que tots els objectes tinguin accés a la gran xarxa d'Internet. Per exemple amb la implementació d'aquest concepte, una nevera podria arribar a fer la compra de manera automatitzada.

Aquest projecte s'emmarca en el món de "les coses" que tenen Internet. Actualment hi ha molts dispositius electrònics com la nevera o el cotxe que ja gaudeixen d'estar connectats a Internet. Si anem un pas més enllà, per què no posar Internet a la cultura? La idea pot semblar ben estrambòtica, però tot i així el que es podria fer és intentar incentivar-la a través d'Internet. Una manera de fer-ho podria ser posar instruments a l'abast de tothom, en llocs distesos i d'oci per tal que la gent els toqués de manera espontània. Aquests instruments podrien estar connectats a Internet i així aprofitar el potencial dinamitzador d'aquesta gran xarxa per tal de fer-ne difusió i alhora poder gestionar-los. Evidentment, sempre que l'instrument ho permeti, és a dir, que poc o molt tingui una part electrònica que permeti la seva gestió.

Amb aquesta iniciativa no només s'aprofita la gran xarxa per promocionar la cultura, sinó que també permet gestionar un conjunt d'instruments de manera compartida, per tal que el cost d'aquests sigui menor. D'aquesta manera es facilita l'accés a instruments musicals que sovint són cars i, ara més que mai degut al context social de crisi en el qual vivim, no són a l'abast de qualsevol butxaca.

Aquest projecte cobreix la necessitat que té l'empresa Click Piano. Aquesta jove empresa requereix d'un sistema de reserves i lloguers per tal de poder gestionar els seus instruments, en aquest cas els pianos.

E-mail de contacte: roger.munoz@e-campus.uab.cat
Menció realitzada: Enginyeria de Tecnologies de la Informació
Treball tutoritzat per: Jordi Herrera Joancomartí (dEIC)

Aquest treball se centra en el desenvolupament d'un sistema que permeti connectar l'instrument del piano a Internet, per tal de poder-lo col·locar a un lloc on la gent hi pugui tenir accés i així poder-ne gestionar el seu ús.

Per tal de realitzar una gestió d'aquest escenari, s'ha desenvolupat un *software* que permet la reserva de pianos i la seva gestió. L'èxit d'aquest projecte recau a aconseguir que un piano, amb accés a Internet, pugui ser gestionat pel sistema de reserves i lloguers.

En el context tecnològic actual, on la societat utilitza tot tipus de dispositius per connectar-se a Internet, la interfície de l'aplicatiu a desenvolupar ha de ser compatible amb la majoria dels dispositius actuals, des de tauletes i ordinadors fins a telèfons intel·ligents. Amb això es vol evitar que la gent tingui problemes d'accés a la plataforma.

El marc d'aquest treball és acadèmic. És un projecte final de grau, tot i que ha estat desenvolupat a una empresa.

L'objectiu principal d'aquest treball és, per tant, implementar el sistema de reserves i lloguers que ens permeti gestionar els pianos i els usuaris. Per tal d'aconseguir aquest objectiu, caldrà assolir els següents subobjectius:

1. Dissenyar l'arquitectura del sistema de reserves i lloguers, identificant els diferents agents.
2. Dissenyar els protocols de comunicacions dels diferents agents.
3. Dissenyar i implementar el *webservice*.

Com a objectius addicionals que es realitzaran si hi ha prou temps hi ha:

1. Desenvolupar la part del *frontend* on els usuaris puguin interactuar des de les diferents plataformes.
2. Desenvolupar la part del *backend* on es pugui gestionar tot el sistema.

La metodologia emprada per tal d'aconseguir tots els objectius d'aquest projecte és la metodologia en cascada. Aquesta decisió ha vingut donada pel temps que hi ha per realitzar el projecte i els recursos disponibles.

En aquest document, primer de tot, es fa un repàs de l'estat actual del desenvolupament web a la Secció 2. Després es realitza una descripció del sistema a la Secció 3. Un cop finalitzada la descripció general, es passa a analitzar les diferents parts que han conformat el desenvolupament a la Secció 4 del sistema. Dins d'aquesta secció es comença per la base de dades a la Subsecció 4.1, el webservice a la Subsecció 4.2, el *backend* i el *frontend* a la Subsecció 4.3. Llavors s'explica el detall del mòdul de transaccions a la Subsecció 4.4, el detall del mòdul de reserves a la Subsecció 4.5 i el detall del mòdul de pagaments a la Subsecció 4.6. Un cop finalitzada la secció desenvolupament, s'explica el test i la documentació a la Secció 5. Llavors es comenten els resultats a la Secció 6. I ja per finalitzar s'inclouen les conclusions a la Secció 7.

2 ESTAT DE L'ART

Qualsevol aplicació web que avui en dia es desenvolupi de cara al públic en general cal que tingui en compte la variabilitat de dispositius que hi ha al mercat.

Una possible manera d'aconseguir aquesta adaptabilitat seria que la lògica de negoci i les diferents possibles interfícies gràfiques d'usuari estiguessin desacoblades. Així, per una banda es pot reutilitzar tot el codi de la lògica de negoci i, per l'altra banda, s'aconsegueix un conjunt d'interfícies d'usuari adaptades a cada dispositiu.

Llavors, el codi on contindrà la lògica de negoci se l'haurà de proveir d'una *Application Programming Interface* (API) [15] per tal que les diferents interfícies puguin comunicar-s'hi.

Un altre factor que cal tenir en compte és quantes interfícies gràfiques d'usuari es poden crear i mantenir. Cal aconseguir arribar al màxim de públic amb el màxim d'interfícies gràfiques, però això no pot suposar un esforç i un cost inassumible per l'empresa. Cal buscar doncs una solució de compromís.

Per arribar a aquesta solució de compromís, tenim, per una banda hi ha les tecnologies hibrides que aporten una part compartida entre les diferents interfícies i una part nativa. Aquestes tecnologies tot i compartir una part del codi comporten, en més o menys mesura, un manteniment específic de cada plataforma o interfície gràfica. Per l'altra banda, les tecnologies web ja han inclòs el concepte de *responsive*, o sigui una mateixa interfície gràfica serveix pels diferents dispositius. De les dues possibilitats s'ha triat la segona ja que a l'haver-hi menys codi és més fàcilment mantenible per l'empresa. També pels recursos disponibles i el temps és la solució més adequada. Aquesta solució és una aplicació web [19].

Un altre punt de vista interessant a comentar és que avui en dia no hi ha cap aplicació que es programi des de zero sense fer ús de llibreries externes. L'ús d'aquestes llibreries facilita les tasques de desenvolupament. En l'àmbit de tecnologies web hi ha *frameworks* i llibreries ja existents al mercat.

Degut a la casuística del projecte, els recursos i temps disponibles, aquest treball cercarà i s'implementaran les solucions en aquests paradigmes i tecnologies.

3 DESCRIPCIÓ DEL SISTEMA

Dins d'aquest sistema hi ha diferents actors segons la funció de cada persona o agent. S'hi troben els perfils d'usuaris finals, els pianos, els administradors d'espais, els espais i els administradors.

En sistema de reserves i lloguers, el subjecte a reservar i llogar és un piano. El piano estarà situat en un espai i aquest espai pot estar administrat per algun propietari d'espai on està ubicat el piano. Per últim, hi haurà els administradors que gestionaran el sistema de reserves i lloguers. Per posar un exemple clarificador, un piano pot estar situat en un bar (espai) i aquest bar té un propietari.

Cada actor tindrà una relació diferent amb el sistema. Per un costat hi ha els usuaris finals, que realitzaran les reserves i lloguers dels pianos durant un interval de temps. Aquests actors necessiten poder realitzar les accions (consultar el balanç, fer recàrregues, reservar un piano, etc.) en el sistema a través de diferents dispositius, des de tauletes, mòbils i/o ordinadors. Sobretot, un usuari final just abans de tocar el piano, caldrà que des del mòbil activi el piano en qüestió. Per aquest motiu és molt important que el sistema funcioni

correctament amb el telèfon mòbil, o sigui a resolucions de pantalla molt petites.

Per un altre costat, hi ha els propietaris dels espais. Aquests actors caldrà que puguin administrar el seu espai, o sigui: administrar la fitxa de l'espai, pujar fotografies del mateix i administrar l'horari d'obertura de l'espai.

També existeix l'usuari administrador del sistema. Aquests actors són els que s'encarregaran de gestionar el sistema i fer tot el seguiment i control del mateix, com per exemple gestionar les tarifes per franges horàries dels diferents pianos, donar d'alta i baixa les promocions, etc. Aquest paper el duen a terme els treballadors de l'empresa.

De manera externa hi ha dos agents més no usuaris, que interactuen amb el sistema. Un és el piano i l'altre és el sistema de gestió de pagaments. En el cas del piano, serà el propi piano el que es connectarà al *webservice* i actualitzarà el propi estat. En el cas del sistema de gestió de pagaments, s'utilitzarà un sistema de pagaments ja existent al mercat anomenat *Braintree* [9]. Aquest sistema de pagament disposa d'un *Software Development Kit* (SDK), que s'utilitzarà a l'hora de fer la implementació.

4 DESENVOLUPAMENT

D'acord amb el que s'ha comentat en la Secció 2, el sistema proposat té una lògica de negoci separada de les diferents interfícies. Per tant, el sistema està composat d'un *webservice*, una base de dades i una interfície gràfica (*frontend* i *backend*). Aquest esquema es pot veure reflectit a la Figura 1. Cal comentar que tota la interacció del piano amb el sistema queda fora de l'àmbit d'aquest treball, tot i ser el piano és un agent del propi sistema.

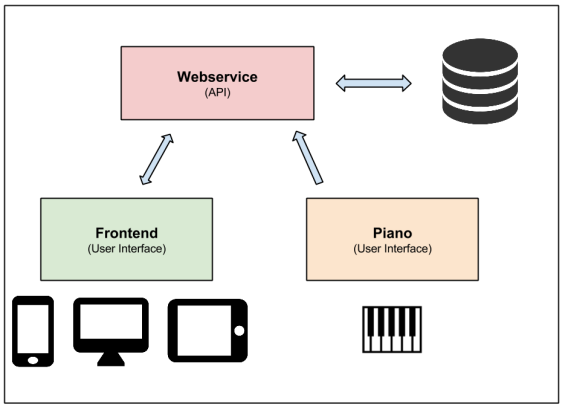

Figura 1: Resum del sistema

4.1 Base de dades

El sistema té una base de dades relacional centralitzada i controlada pel *webservice*. El control de l'accés a les dades resideix única i exclusivament en el *webservice*. D'aquesta manera s'aconsegueix la independència de les dades amb el subsistema de les interfícies d'usuari.

La decisió de fer la base de dades relacional és clara; tots els registres que intervenen en el sistema tenen sempre els mateixos camps, per tant implementar solucions no relacionals no aporta cap millora respecte els relacionals. D'altra banda s'ha escollit el MySQL [10] perquè de les alternatives gratuïtes és la que l'estudiant estava més familiaritzada.

L'eina utilitzada per tal de fer el modelatge de la base de dades ha estat el programa MySQL Workbench [3].

El resultat de la base de dades es pot consultar a l'apèndix A.2 a la Figura 6.

La base de dades està estructurada en set àrees. Aquestes set àrees corresponen als set grans conceptes: espais, usuaris, pianos, transaccions, horaris, reserves i imatges.

Els **espais** guarden tots els atributs referents a aquests i també engloben els serveis que els espais ofereixen als seus clients. Aquests serveis són tractats de manera global per tal de facilitar les cerques. Exemple de serveis poden ser la disponibilitat de xarxa sense fils o terrassa per part de l'espai.

Els **usuaris**, a part de guardar totes les dades referents a aquesta entitat, també engloben la taula de permisos i rols. A l'etapa d'anàlisi no es va plantejar aquesta àrea amb una taula permisos, sinó que els permisos es volien deixar programats a cada funció del *webservice*. Evidentment però, aquesta decisió va ser rectificada ràpidament, ja que a l'etapa de disseny es va veure la necessitat d'una taula on poder gestionar de manera centralitzada i ordenada l'accés dels diferents usuaris i rols a la base de dades. Va ser llavors quan es va implementar aquesta funcionalitat.

La taula de permisos està composta de dues parts: "a qui"es dóna permís i "a què"es dóna exactament permís. A la primera part, hi ha tres maneres d'especificar a qui es dóna el permís: al propietari (*owner*), a un grup d'usuaris que tenen un rol concret i a un usuari concret. La decisió de tenir el propietari dins d'aquesta taula ve inspirada en la manera que Linux implementa l'accés al seu sistema de fitxers. Per altra banda, no s'ha fet ús dels grups de Linux, sinó que s'ha adoptat la forma de rols que s'implementen en les *Role-based access control* (RBAC) [14], o sigui l'usuari pot tenir un o varis rols assignats, i és en aquests rols on es concedeixen els permisos. L'última manera d'especificar el permís, o sigui a un usuari concret, tot i que actualment no es farà servir, s'ha implementat per si mai es dóna un cas no contemplat en el propi sistema de propietaris i rols.

Els rols que existeixen al sistema són els següents: *anonymous*, *user*, piano i *admin*. Els rols són assignats automàticament, o sigui quan un usuari es registra, se li assigna el rol d'usuari. O bé, quan un piano és donat d'alta en el sistema se li assigna el rol de piano.

Per últim destacar que s'ha deixat el sistema preparat per tal d'implementar la identificació d'usuaris mitjançant les xarxes socials de *Facebook* i *Twitter*. Amb aquesta funció el que s'aconsegueix són dues funcionalitats en primer lloc facilitar el registre d'usuaris d'aquestes xarxes i en segon lloc es poden utilitzar aquestes credencials per tal de tenir presència a la xarxa social (sempre i quan l'usuari ho autoritzi).

El **piano** s'ha modelat com a ítem dins del sistema de lloguers i reserves. El sistema de reserves es va pensar en genèric, o sigui, que es pogués llogar qualsevol element. A part de ser un ítem, al piano se li ha assignat un usuari, o sigui que quan es dóna d'alta un piano, a part de guardar la informació referent a aquest, es genera un usuari específic amb unes credencials d'usuari per cada piano. D'aquesta manera el piano es pot validar al *webservice*, i així pot executar els mètodes per tal d'aportar informació al sistema de

reserves sobre l'estat del mateix. A la fitxa del piano, s'ha inclòs la possibilitat d'assignar-li imatges. Així quan un usuari està reservant un piano pot veure exactament el piano que és.

Una altra àrea és la de les **transaccions**. Aquestes són les encarregades d'aportar la informació de l'estat econòmic del sistema, o sigui són les que recolliran, entre altres coses, els pagaments que es realitzin cap al sistema. Les transaccions s'han modelat de manera centralitzada en una taula. El que diferencia els tipus de transacció són dues variables de la mateixa: l'origen i el destí. Per exemple, un pagament cap al sistema o una sortida cap a fora el sistema, és l'origen i el destí de la mateixa. Aquest punt serà explicat més en detall a la Secció 4.4.

Per tal de poder modelar quan els espais estan disponibles per poder tocar el piano ha calgut incloure també el concepte d'**horari**. L'horari és doncs un interval de temps en el qual el piano està disponible per ser tocat. En aquest sistema el concepte d'horari s'ha modelat en dues parts. Per un costat, es modela l'horari d'obertura normal (positiu) indicant el dia de la setmana, els horaris d'obertura i els horaris de tancament d'aquell dia. Per exemple dilluns de 10:00 a 14:00 i de 17:00 a 21:00. S'ha implementat per dies de la setmana i horari d'obertura i tancament perquè habitualment els horaris comercials de cada setmana són iguals. Però aquesta implementació sense res més és incompleta, ja que si un dia concret es volen variar els horaris, el sistema no seria capaç d'acceptar aquest cas. Per això, també s'ha modelat la restricció horària (negatiu), és a dir, l'especificació de dates concretes en les quals el piano no estarà disponible. Per exemple dilluns 15 de juny de 2015 de 9:00 a 21:00. Això permet modelar dies festius, tancaments d'espais per esdeveniments privats, vacances, etc. Així doncs, per tal d'obtenir l'horari d'obertura del piano, cal tenir en compte els horaris d'obertura (positiu) i les restriccions introduïdes (negatiu). D'aquesta manera, es pot saber si el piano estarà disponible per tocar o no. Evidentment, com ja es comentarà més endavant, caldrà comprovar altres factors, com per exemple que no hi ha cap altra reserva activa en aquell horari.

L'àrea de **reserves** conté diversos conceptes. Per una banda, el concepte de tarifa. S'han modelat diferents tarifes (preu/hora) per tal que aquestes siguin aplicables segons el piano, el dia de la setmana i l'hora. Un apunt important és que la tarifa es crea i es destrueix, no és modificable. Aquesta restricció permet evitar incoherències dins de la pròpia base de dades, com ara en el cas que es tingui una reserva confirmada d'un usuari amb una tarifa per d'aquí dos dies i ara mateix es realitzi un canvi de tarifa, fet que fa variar el preu de la reserva. Evidentment amb l'usuari s'ha de mantenir el preu pactat (de quan l'usuari va realitzar la reserva), per tant si es permetia el canvi de tarifes s'havia de tractar aquest cas. Una possible solució era realitzar un bonus en el cas que la diferència fos a favor de l'usuari, i en cas contrari si el cost era superior l'empresa l'havia d'assumir. Tot plegat resultava feixuc, per aquest motiu es va decidir que les tarifes no eren modificables. Això sí, un piano sí que admet un canvi de tarifa.

L'àrea de reserves també incorpora els conceptes de targeta regal, targetes amb un codi promocional i recàrregues. Quan un usuari vol regalar diners a un altre usuari utilitza una targeta targeta regal. D'altra banda, quan els administradors vulguin promocionar el sistema de lloguer de pianos el que voldran és regalar diners a usuaris no registrats per tal que es registrin i provin els pianos via els codis promocionals. Finalment quan un usuari vulgui incorporar diners a dins del seu perfil per tal de poder després realitzar reserves el que necessitarà és realitzar una recàrrega. Tots aquests conceptes són modelats en un de sol: el bonus. El bonus, per tal de diferenciar cada cas anterior, té un tipus. També té un valor d'entrada i un de sortida. Aquests dos valors serveixen per modelar casos com per exemple: si recarregues 10 euros te'n donem 15. Per altra banda fent zero el camp de valor d'entrada i qualsevol valor al camp de sortida s'obtén el cas de targetes regal o codis promocionals.

Finalment l'àrea de reserves conté les reserves dels diferents usuaris. Una reserva tindrà associada una o vàries tarifes i un piano. La reserva (igual que en el cas de la tarifa) no és modificable, sols es pot crear i destruir. Aquesta restricció serveix per mantenir la coherència de les dades dins de la base de dades.

L'última àrea de dades és la d'**imatges**. Es va decidir modelar la gestió de les imatges de manera centralitzada perquè en una de les reunions de seguiment amb l'empresa va sorgir la possibilitat de realitzar una xarxa social associada a aquest sistema. És per aquest motiu que es va decidir centralitzar el sistema de fitxers d'imatges, per tal que si algun dia sorgeix la necessitat de tractar i gestionar grans volums d'imatges, es pugui ampliar aquest subsistema de forma senzilla.

Com a últim comentari respecte la base de dades cal tenir en compte que al nostre país, quan una empresa tracta amb dades, en aquest cas, de clients, cal que aquestes estiguin protegides per tal de complir amb la llei de protecció de dades [1]. En aquest cas al tractar-se de dades de nivell 1, no cal que estiguin xifrades a la base de dades, però sí que s'ha d'assegurar-ne la confidencialitat. Per tant, el que sí que és requisit indispensable és que les dades viatgin xifrades des del servidor fins als diferents terminals clients que s'hi connectin. En aquest context s'ha optat per instal·lar un certificat amb la tecnologia OpenSSL [18] en el *webservice*, per tal que les dades siguin xifrades al sortir del servidor. El certificat en sí no garanteix el xifratge de les dades, però la utilització del protocol HTTPS sí que n'assegura la protecció. El certificat s'utilitza per autenticar el servidor, i un cop autenticat es negocia una clau. És amb aquesta clau que es xifra el contingut de la comunicació.

4.2 Webservice

El *webservice* del sistema de reserva ha estat desenvolupat amb la tecnologia de Python Flask [2]. Flask és una llibreria que facilita la creació de servidors web amb el llenguatge Python. El concepte bàsic que utilitza és el decorador de Python. Es pot entendre el decorador [4] de Python com una interrupció al flux propi del programa. Un exemple clarificador seria el del Fragment 1: la idea és que abans d'executar la funció *foo()* s'executa la funció *decorator()*.

Fragment 1: Exemple de decorador

```
@decorator
def foo():
        #something pass inside function
        print "foo"
        ...
```

Aprofitant la potencialitat dels decoradors, Flask proveeix funcions per tal de facilitar l'encaminament, o sigui, des que arriba una petició fins que s'encamina a la funció que correspon. També amb els decoradors s'ha realitzat tota l'autenticació i autorització dels usuaris (que més endavant se n'explicarà el seu funcionament).

El codi del *webservice* s'ha estructurat, igual que la base de dades, per àrees: usuaris, pianos, espais, transaccions i reserves. Com es pot veure, les àrees no són ben bé les mateixes ja que s'han unificat els horaris amb les reserves per tenir una estructura més compacte.

També existeix una àrea comuna que és on hi ha allotjat el codi de les llibreries comunes: enllaç amb la base de dades, constants, estandardització dels missatges de retorn, gestió de fitxers d'imatges i l'autenticació i autorització d'usuaris.

El flux de treball que segueix el *webservice* queda recollit a la Figura 2.

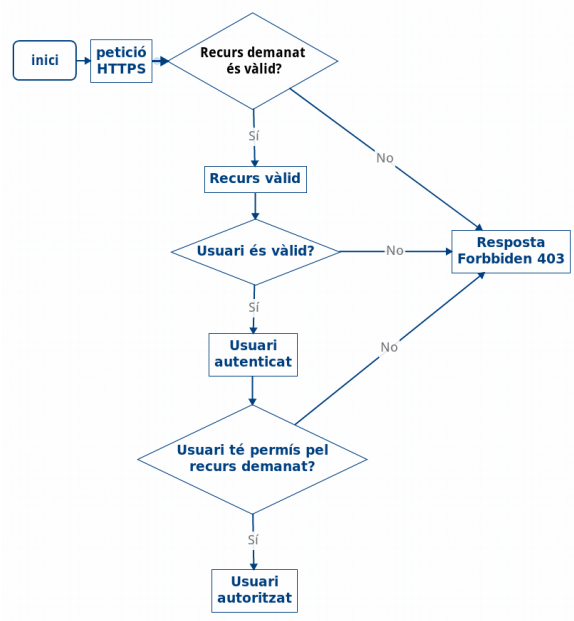

Figura 2: Flux del *webservice*

Com es pot comprovar, tot comença quan una petició HTTPS arriba al *webservice*. Si la petició és HTTP, el webservice no en fa cas, ja que sols està escoltant peticions segures (TLS) al port 5000.

Un cop arriba la petició el primer que es comprova és que la petició és vàlida i pot ser enrutada, o sigui, que segueix l'estructura `base_url/entity/action` i que, si incorpora dades, les porti amb el mètode *POST*. Evidentment també es comprova que la parella entitat acció sigui vàlida. Qualsevol altre petició de recurs que no compleixi aquesta norma serà denegada directament. Per exemple, una petició que tingui una entitat però no una acció o les dades vénen amb el mètode GET aquesta petició és contestada amb un codi HTTP 403, o sigui *forbbiden*.

Si la petició és vàlida, el següent pas que es comprova és que l'usuari que l'està fent és un usuari vàlid dins del sistema. Aquest procés es pot realitzar amb dos conjunts de paràmetres d'entrada: per *token* o per usuari i contrasenya. Encara que es pugui autenticar l'usuari amb dos grup de dades, sols se'n processarà un dels dos, prioritzant l'*access_token* per davant de l'usuari i contrasenya. Tant si l'*access_token*, com la parella de paràmetres usuari i contrasenya són vàlids l'usuari serà validat i es passarà al següent nivell: l'autorització. En cas contrari, si algun paràmetre és erroni es denegarà i es respondrà de nou de manera taxativa amb el codi d'error HTTP 403.

Un cop la petició és correcta, i l'usuari ha estat validat, el següent pas és autoritzar l'usuari pel recurs que ha demanat. Aquest procés fa una unió dels recursos que l'usuari té autoritzats a la taula *permissions*, ja explicada a la Secció 4.1, segons la seva identitat, el seu rol i si és o no propietari del recurs sol·licitat. També es processaran els camps que demana l'usuari. Això vol dir que pot donar-se el cas que un usuari demani per veure un camp del qual no té accés. Llavors de la mateixa manera es retornarà el codi d'error HTTP 403. En cas que l'usuari tingui accés al recurs i als camps que demana a la petició, serà redirigit a la funció que processa aquell recurs.

Aquest flux que s'acaba de descriure és el normal de les peticions del *webservice*, però també existeixen peticions que no requereixen d'un usuari registrat per fer-se, com per exemple, la petició de registrar un nou usuari (`user/create`). En aquest tipus de peticions, o sigui les que no cal autenticar ni autoritzar l'usuari, el flux de treball se simplifica: si la petició és vàlida directament s'executa la funció requerida.

Val la pena destacar els *tokens* d'usuari, ja que la seva utilització permet fer més ràpida la validació; el *token* quan es guarda a la base de dades sols es fa un *hash* sense *salt*, en canvi l'usuari i la contrasenya s'utilitza *hash* amb *salt* per la poca naturalitat d'aleatorietat que tenen els usuaris a l'hora d'escollir la seva contrasenya. També un altre motiu, més que raonable, és que en l'aplicació client (sigui quina sigui) no cal que guardi l'usuari i la contrasenya ni en sessió ni *cookies* ni en un lloc temporal. I l'altre motiu és que el *webservice* està fet que no guardi l'estat per tal que sigui més escalable al llarg del temps, per tant l'aplicació client ha de guardar alguna dada de l'usuari per validar-lo, i tal i com s'ha comentat anteriorment, l'usuari i la contrasenya no és una bona solució.

Ja per finalitzar aquesta secció indica que tots els missatges de retorn del *webservice* han estat estandarditzats per tal de facilitar les diferents aplicacions de client que es puguin connectar. Tots els missatges positius contenen un objecte *ok*, un objecte *access_token* i un objecte *data*. El contingut d'aquests objectes depèn del recurs demanat. En canvi, els missatges d'error contenen sempre un objecte *error* i, si passa el procés d'autenticació d'usuari també contindrà l'objecte *access_token*. Tots els missatges que retorna el *webservice* estan codificats amb el format estàndard JSON[13].

4.3 Backend i Frontend

En un inici del projecte, el *backend* i el *frontend* eren dues interfícies gràfiques d'usuari web separades, una pels administradors del sistema i una que servia a l'usuari final. Però degut a la compartició de les seves funcions es van fusionar en una sola, per tal que els administradors de l'empresa sols tinguessin una sola plataforma on gestionar tot el sistema. Segons el rol que tingui l'usuari assignat es mostren unes opcions o unes altres.

La tecnologia emprada per desenvolupar aquesta part del

projecte és *Codeigniter* [6]. *Codeigniter* és un *framework* escrit en PHP, que serveix per ajudar al desenvolupador web a realitzar les tasques més habituals de les aplicacions en PHP, proporcionant un conjunt de llibreries que faciliten el desenvolupament. Aquest *framework* es caracteritza per ser una eina molt poc pesada (sols ocupa 2 MB), modular i extensible. A part d'aquest *framework* s'han utilitzat les llibreries *Bootstrap* [5] i *jQuery* [7]. *Bootstrap* s'ha fet servir per facilitar la compatibilitat entre dispositius des de mòbils a ordinadors, ja que el *backend* i *frontend* s'ha desenvolupat de forma *responsive*. *jQuery* s'ha utilitzat per tal de fer més amigable la programació amb javascript.

L'estructura que s'ha seguit per desenvolupar aquesta part ha estat el model vista controlador (MVC) que ja incorpora de manera innata el *framework* de PHP. També s'ha creat una llibreria anomenada *l_webservice* que és la que gestiona les comunicacions amb el webservice de manera centralitzada. Per tal de fer les peticions HTTPS mitjançant PHP, s'ha utilitzat el mòdul *cUrl* de PHP [8].

Pel que fa a la relació entre el *backend* i el *frontend* i la base de dades cal indicar que totes les funcions que requereixen accés a les dades, evidentment no apunten directament a la base de dades MySQL sinó que fan una crida al corresponent mètode del *webservice* per tal d'assegurar la independència comentada a la Secció 2.

Per últim, per tal d'aconseguir el compliment de la llei de protecció de dades o sigui d'assegurar el xifratge de les dades en trànsit entre el *backend* i *frontend* i el client, caldrà que faci ús del protocol HTTPS. A més, per tal d'assegurar l'autenticitat del *backend* i *frontend* es farà ús d'un certificat, igual que s'ha utilitzat en el *webservice*. Això és així perquè les dades des del *webservice* fins al *backend* i *frontend* ja viatgen xifrades, però evidentment si no es fa ús de la tecnologia TLS des del *backend* i *frontend* fins on estigui el client final les dades viatjarien en clar, essent susceptibles de ser vistes per tercers.

4.4 Detall del mòdul de transaccions

El sistema per gestionar els pagaments que s'ha implementat parteix del fet que l'usuari té un balanç associat. Aquest balanç és el saldo que l'usuari té per gastar en el sistema, de la mateixa manera que en el sistema bancari tradicional. Tots els moviments del balanç de l'usuari, tant si fa reserves, com si fa recàrregues, etc. es gestionen a través de les transaccions. La idea d'aquesta modelització ve inspirada amb una de les idees que hi ha darrere de la *blockchain* del sistema monetari digital *Bitcoin* [17].

La transacció és la unitat bàsica del sistema de reserves que indica que hi ha hagut un moviment monetari d'una institució a una altra. Per representar aquest moviment la transacció conté un ens d'origen i un de destí. Existeixen quatre tipus d'institucions diferents dins del subsistema de transaccions: sistema (*sys*), entitat financera, bonus i balanç (*Balance*). Els moviments entre aquestes quatre institucions permeten modelar tota la casuística de moviments de diners dins del sistema. A la Figura 3 es veuen clarament les institucions, així com també les diferents accions entre elles, representades per les transaccions.

La institució **entitat financera** representa l'entrada de diners externs al sistema per part dels usuaris finals. Tot i que aquesta institució té el nom genèric d'entitat financera els

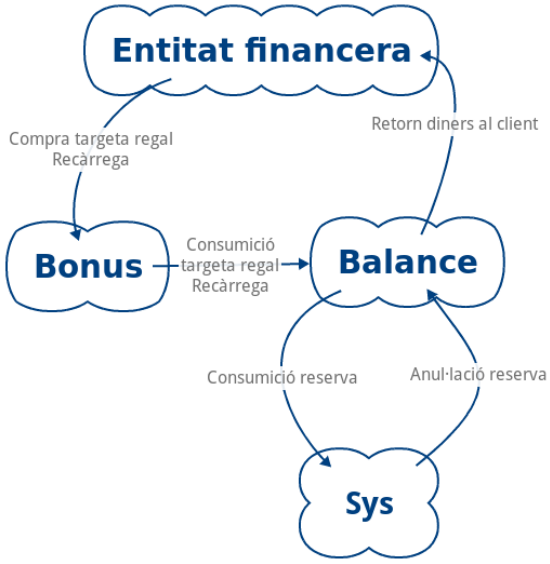

Figura 3: Flux de treball de les diferents institucions de les transaccions

pagaments es realitzaran a través de la plataforma de *Braintree* de *PayPal*. Aquesta plataforma permet tot tipus de pagaments des de targetes, girs bancaris, comptes *PayPal*, etc.

Des de la institució d'entitat financera es pot arribar a la institució de **bonus**. Els bonus bàsicament són de tres tipus: targetes de regal, recàrregues i promocions. Les targetes regal són diners que un usuari regala a un altre per poder després fer reserves. Les targetes regal tenen un codi d'utilització únic. Les recàrregues en el sistema també estan modelades com a bonus, ja que una recàrrega és un conjunt de diners que entren al sistema i que són traspassats a l'ens balanç. La diferència amb una targeta regal justament és aquest traspàs immediat cap al balanç d'un usuari. Les promocions són diners que el sistema regala en forma de bonus, per exemple pot realitzar campanyes de màrqueting amb un codi de bonus que regali 5 euros a cada usuari. Per tal que els usuaris no facin un abús d'aquest tipus de promocions, hi ha un camp a la taula de bonus que controla justament el número de vegades que pot ser utilitzat un determinat bonus per l'usuari. Val a dir que totes les transaccions que tenen a veure amb un bonus tenen el camp *bonus_code_id* omplert per tal de poder fer la traçabilitat més tard. És a dir, amb el sistema dissenyat, a partir d'una transacció es poden recuperar totes les dades que es van tenir en compte en el moment d'escriure la transacció al sistema.

La decisió d'ajuntar el concepte de les targetes regal, recàrregues i promocions com a bonus d'un ús, o de varis usos, ve donada perquè amb una sola entitat es podia gestionar tot, i simplificava el model tant de transaccions com de bonus, així com també la seva gestió.

La institució **balanç** és la que s'encarrega de representar els balanços de tots els usuaris. Li poden arribar transaccions des de l'ens bonus, en forma de targeta regal, recàrrega o promoció, i des del la institució sistema (anul·lació de reserva). Des d'aquesta institució poden sortir transaccions cap a l'entitat financera (retorn de diners al client) o cap a l'ens sistema (quan es realitza una reserva el saldo de l'usuari passa al sistema).

L'última institució que queda per explicar és la del **siste-**

ma (*sys*). Aquest ens és l'encarregat de gestionar els diners de les reserves. Aquesta institució és la representació de la pròpia empresa que gestiona el sistema de reserves i lloguers. Quan un usuari realitza una reserva l'ens sistema rep el valor d'aquesta reserva. De manera contrària, quan l'usuari anul·la una reserva els diners són retornats a la institució balanç. Un apunt sobre l'anul·lació de reserva: l'usuari sempre podrà anul·lar la reserva si aquesta no ha començat a executar-se (tocar el piano) i el temps que falta fins l'execució de la reserva en qüestió és major que un paràmetre configurable del sistema. Aquest paràmetre es troba a la taula *config* de la base de dades i té com a *key* el valor de *booking_cancel_allowed_hours* en segons.

El fet de modelar les transaccions d'aquesta manera fa que se'n pugui extreure un conjunt d'informació addicional sobre l'estat econòmic del sistema.

Per tal de saber quants diners ha ingressat l'empresa del sistema es pot utilitzar l'Equació 1, que representa la suma de les transaccions que tenen com origen entitat financera menys la suma de les transaccions que tenen com a destí entitat financera (que representen els retorns de diners al client).

$$\begin{aligned}\text{Ingressos} = &\sum_{\forall T_x \mid T_x.\text{From=Entitat financera}} T_x.\text{Quantity} \\ - &\sum_{\forall T_x \mid T_x.\text{To=Entitat financera}} T_x.\text{Quantity}\end{aligned} \tag{1}$$

El sistema també pot comprovar si els balanços dels diferents usuaris són correctes. Per fer-ho, cal aplicar l'Equació 2, que representa els diners que ha recarregat l'usuari menys els diners que ja ha consumit. La següent equació indica el balanç d'un usuari i:

$$\begin{aligned}\text{Balance User}_i = &\sum_{\forall T_x \mid T_x.\text{From=Balance } \& T_x.\text{UserId=i}} T_x.\text{Quantity} \\ - &\sum_{\forall T_x \mid T_x.\text{To=Balance } \& T_x.\text{UserId=i}} T_x.\text{Quantity}\end{aligned} \tag{2}$$

D'una manera similar, també es pot saber quina quantitat de diners han estat consumits en reserves, aplicant l'Equació 3.

$$\begin{aligned}\text{Diners reserves} = &\sum_{\forall T_x \mid T_x.\text{From=Sys}} T_x.\text{Quantity} \\ - &\sum_{\forall T_x \mid T_x.\text{To=Sys}} T_x.\text{Quantity}\end{aligned} \tag{3}$$

4.5 Detall del mòdul reserves

En aquesta secció es realitza una visió de com funciona internament el mòdul de reserves. També s'inclou tot el sistema de pagament *online*, ja que una reserva no es pot realitzar si l'usuari no té una quantitat (en el seu balanç) igual o superior al cost de la reserva en qüestió.

El flux d'execució que segueix el mòdul de reserves és el que es reflecteix a la Figura 4.

El procés comença amb un usuari identificat. Això vol dir un usuari que ha estat registrat a l'aplicació, s'ha comprovat

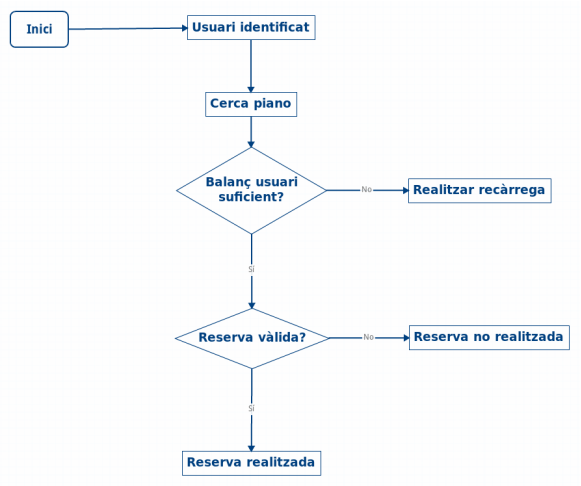

Figura 4: Flux d'un usuari realitzant una reserva

el seu correu electrònic i que aquest usuari té credencials vàlides per fer accions amb el sistema.

Aquest usuari en qüestió realitza una cerca sobre el conjunt de pianos per tal de seleccionar-ne un. Aquesta cerca, tot i que el que es vol aconseguir és el piano, realment es realitza sobre el conjunt d'espais. Aquesta decisió es va prendre per facilitar l'usabilitat del sistema, ja que els usuaris coneixen espais (llocs), però en canvi normalment no coneixen models o marques de piano. És a dir, el criteri més probable de l'usuari a l'hora de triar un piano és seva localització geogràfica. També aquesta decisió va quedar reforçada quan es va incorporar la utilització d'un mapa per tal de situar els espais i els pianos.

Un cop l'usuari ha seleccionat el piano desitjat, cal indicar el dia i hora de començar i acabar a tocar. De manera opcional, si el piano ho permet, els sistema pregunta a l'usuari si vol tocar amb els auriculars posats (o sigui, només per ell) o si vol tocar en obert (tothom que està a l'espai pot sentir-lo).

Un cop introduïdes totes aquestes dades el sistema de reserves realitza un conjunt de comprovacions que es detallen tot seguit.

Primer de tot es comproven els horaris. Per una banda, es realitzen les comprovacions de si l'espai on està ubicat el piano està obert, és a dir, les comprovacions d'horari positives que es comentaven a la Secció 4.1.

Un cop validat l'horari positiu, es comprova que no hi hagi cap restricció i/o incidència durant l'interval que l'usuari està demanant per fer la reserva. Aquesta comprovació és la d'horari en negatiu o restricció horària que es comentava a la Secció 4.1.

Acte seguit, si les anteriors comprovacions han estat exitoses, es prossegueix a comprovar que no hi hagi cap altra reserva que col·lisioni directament amb la que proposa l'usuari.

En el cas que no hi hagi cap col·lisió, es comprova l'ús d'auriculars de la reserva. Per una banda el piano ho ha de permetre, i per l'altra banda, l'usuari ho ha de demanar. Aquesta comprovació no seria necessària en un ús normal de l'aplicació, ja que la interfície gràfica no oferiria l'alternativa de tocar en obert si aquesta no està disponible. Però, al ser un *webservice* accessible públicament, pot rebre peticions d'altres agents no sols del *backend* i *frontend*. Per

tant, s'ha de comprovar que l'agent que fa la petició no està intentant fer alguna cosa no permesa; en aquest cas fer una reserva amb auriculars quan el piano no ho permet en la seva configuració.

Un cop finalitzades aquestes comprovacions referents a la reserva en sí, se segueix amb el càlcul del preu de la reserva. Aquest càlcul es realitza mitjançant el conjunt de tarifes que té associades el piano en l'interval que l'usuari està demanant la reserva.

Un cop calculat el preu de la reserva es prossegueix a mirar si l'usuari té el saldo corresponent a la quantitat de la reserva que està fent. En cas de no tenir suficient saldo en el compte, es retornarà un error referent que no hi ha prou saldo i es convidarà a l'usuari a realitzar una recàrrega o una consumició de targeta de regal, en cas que l'usuari en tingui. Més endavant es parlarà de com funciona el sistema de pagament.

Comprovat el saldo de l'usuari, el següent pas és crear la transacció amb origen *balance* i destí *sys* amb usuari que se li passa com a paràmetre a la petició de reserva.

Realitzades totes aquestes accions sols queda inserir la reserva com a tal a la taula de reserves.

4.6 Detall del mòdul pagaments

El mòdul que controla tot el sistema i la gestió dels pagaments en aquest projecte ha estat realitzat mitjançant la plataforma Braintree [9]. La utilització de la plataforma de pagament ha estat realment molt útil, ja que si s'hagués hagut d'implementar des de zero, aquest mòdul sol ja suposa un volum de treball equivalent a un projecte final de grau.

Tal com s'ha comentat a la Secció 4.5, quan un usuari realitza una reserva, necessita prèviament tenir un balanç positiu en el seu compte. Aquest balanç s'aconsegueix a partir del bescanvi de bonus, o sigui recàrregues directes de l'usuari o consumició de targeta regal que un altre usuari li ha donat. Segons la Figura 3, per tal d'aconseguir bonus cal una transacció de la institució entitat financera a bonus. Aquesta transacció s'aconsegueix quan un pagament ha estat acceptat i validat per part de la plataforma de *Braintree*.

El sistema de pagament de *Braintree* funciona segons la Figura 5.

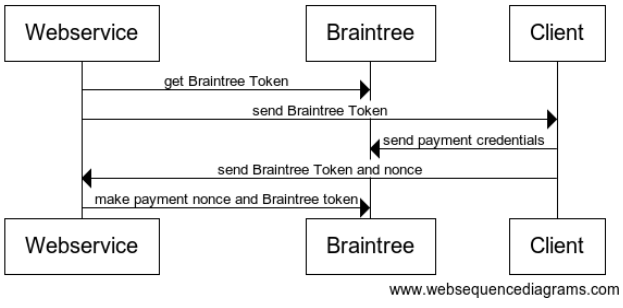

Figura 5: Resum del funcionament del sistema de pagament *Braintree*.

Primer de tot el que cal és tenir un *token* vàlid del servidor de *Braintree*. Cal tenir en compte que aquest *token* és diferent de l'*access_token* que utilitza el *webservice* per tal d'identificar els usuaris del sistema.

Un cop s'ha aconseguit aquest *token*, s'envia al client (navegador del client final que tindrà oberta la web de *backend* i *frontend*). Llavors el client autoritza amb *Braintree* un pagament cap a nosaltres, en aquest projecte mitjançant les formes de pagament *PayPal* o targeta de crèdit o dèbit. El servidor de *Braintree* retorna al client un *nonce*. Acte seguit, el client envia al *webservice* el *token* inicial i el *nonce* rebut del servidor de *Braintree*.

El *webservice* un cop rep la petició HTTPS amb els camps de *token* i *nonce* intenta realitzar una transacció (compra) per l'import de la reserva. Si el servidor de *Braintree* retorna que la transacció (compra) és vàlida, com que el sistema confia amb l'entitat *Braintree* s'accepta aquesta resposta com a vàlida, i es genera la transacció en el sistema de reserves amb origen entitat financera i destí bonus.

Si la petició que es rep és de recàrrega, a més a més, es genera una altra transacció amb origen bonus i destí balanç. En aquest cas també s'actualitza el balanç de l'usuari.

5 TESTS I DOCUMENTACIÓ

Un cop realitzada la fase de desenvolupament, cal comprovar la correcció de les funcionalitats del sistema. En el projecte s'han dedicat tots els esforços dels tests a la part del *webservice*, ja que és la part més crítica del projecte i és on recau la lògica de l'aplicació.

Per tal de dur a terme aquesta fase del projecte, s'ha fet ús dels tests unitaris. La idea bàsica dels tests unitaris és realitzar proves a les diferents parts del codi per separat. D'aquesta manera es pot generar un conjunt de tests que comproven que la funció o funcions que s'avaluen tenen un correcte funcionament. Són molt útils quan es realitzen canvis en el codi que ja funciona, ja que amb una sola execució de tots els tests es comprova que el canvi efectuat en el codi ha tingut el comportament esperat i que no s'ha realitzat cap altra acció no controlada en el mateix projecte.

Val a dir que mentre es realitza l'etapa de desenvolupament del *webservice* les crides HTTPS al propi *webservice* no s'han realitzat amb els tests unitaris, sinó que s'ha utilitzat l'eina *Postman* [11]. Aquesta eina permet realitzar consultes HTTPS de manera molt senzilla i dinàmica. A més a més permet guardar les consultes de manera que la realització de proves se simplifica.

La llibreria *Flask* de Python amb la qual s'ha desenvolupat el projecte conté una sèrie de funcions que permeten generar aquestes funcions de test. Ha estat una tasca relativament senzilla, ja que amb una sola línia de codi es poden realitzar les peticions HTTPS al *webservice* amb els paràmetres POST necessaris per cada crida. De totes maneres, la tasca de realitzar els tests unitaris té la seva complicació ja que s'han hagut de provar totes les casuístiques de cada funció.

Un cop finalitzats els tests unitaris a tota la part del *webservice*, el següent pas ha estat repassar la documentació de totes i cadascuna de les funcions del *webservice*, per tal que si mai s'ha d'editar alguna d'aquestes funcions l'equip que realitzi aquest canvi tingui una visió global i ràpida del propi projecte i codi. Així doncs, totes les funcions del *webservice* inclouen un comentari amb el format adient, tot indicant una petita descripció de què fa la funció, els paràmetres d'entrada, si són opcionals o no, i el retorn de la funció. Això ha permès, posteriorment, fer ús de l'eina de generació automàtica de documentació *Sphinx* [12]. D'aquesta

senzilla manera, s'ha executat el procés automàtic de generació de documentació i la documentació del codi ha estat realitzada.

6 RESULTATS

El resultat principal d'aquest projecte és el propi sistema de reserves compost per un *webservice* i un *backend* i *frontend*, tal com mostra la Figura 1.

Pel que fa al *webservice*, s'ha implementat una API totalment funcional que compleix amb tots els requeriments especificats a l'inici del projecte. Aquesta API és utilitzada tant per les interfícies gràfiques (que serveixen de punt d'accés pels usuaris finals de l'aplicació) com pels pianos (element que no s'ha explicat en detall en aquest document ja que la seva implementació quedava fora de l'abast del projecte, però que són una peça indispensable del sistema).

En relació a les interfícies d'usuari, el resultat final ha estat la unificació del *backend* (utilitzat pels administradors del sistema) i *frontend* (utilitzat pels clients finals) en una sola web. Aquesta web mostra més o menys opcions depenent del perfil d'usuari identificat. A més, la web és completament *responsive* i, per tant, s'adapta a una gran quantitat de dispositius, des de dispositius de pantalla petita com ara mòbils fins a ordinadors de sobretaula. Així, s'ha aconseguit assolir el requeriment d'arribar al màxim de dispositius diferents amb una sola plataforma. D'aquesta manera, es redueix notablement el cost de manteniment del codi, disminuint alhora els costos per l'empresa.

Respecte a les proves de funcionament de l'aplicació, s'han generat testos unitaris de la part més important del sistema, el *webservice*. Aquests testos comproven que les funcions implementades retornen els resultats esperats per tot un conjunt de casos de prova.

Finalment, s'ha generat documentació automàtica del codi a partir dels propis comentaris en *docstring*[16] que s'han inclòs en les definicions de les diferents funcions.

Així doncs, observant els objectius proposats a l'inici d'aquest treball final de grau, es pot observar que s'han aconseguit tots els objectius, fins i tot els secundaris.

7 CONCLUSIONS

El projecte proposat de poder fer arribar Internet als pianos ha estat realitzat amb èxit. La necessitat de l'empresa del sistema de reserves i lloguers per tal de poder gestionar els pianos ha estat coberta.

Aquest sistema està compost d'un *webservice* i un *backend* i *frontend*. El requeriment que l'aplicació s'adaptés a la gran majoria de dispositius ha estat assolit. També el requeriment de fer el codi mantenible per l'empresa ha estat complert, ja que amb un sol codi pel *backend* i *frontend* es cobreixen la majoria de dispositius, el codi està degudament comentat, és modular i s'han inclòs els tests unitaris que permeten comprovar que els canvis que es poden anar introduint no produeixen conflictes amb el codi existent, o si els produeixen aquests són detectats.

Cal remarcar que el disseny plantejat inicialment en el projecte, dins l'etapa de disseny, ha resistit a tots els canvis i totes les peticions demanades per l'empresa. S'hi han realitzat canvis, però aquests han estat amb mínim impacte. A més, durant la implementació s'ha pogut veure com les decisions preses a l'etapa de disseny han estat encertades, ja que s'ha pogut desenvolupar tota l'aplicació amb canvis menors.

La idea de tenir separada la lògica de negoci de les possibles interfícies gràfiques ha estat molt positiva, ja que el gruix de feina ha estat desenvolupar el *webservice* i la implementació de la interfície gràfica (*backend* i *frontend*) i la seva integració amb el *webservice* han estat fàcils.

La tria de les tecnologies també ha resultat ser encertada. La decisió d'utilitzar *Codeigniter* com a *framework* per desenvolupar la part d'interfície d'usuari ha facilitat molt la tasca de desenvolupament. Amb el *framework* en qüestió s'ha pogut treballar de manera molt àgil i flexible, i l'aprenentatge no ha estat massa complicat. De la mateixa manera, la llibreria Flask ha resultat ideal per la implementació del *webservice*, doncs ha aportat totes les avantatges d'un llenguatge d'alt nivell com Python al desenvolupament web.

De la planificació feta inicialment, cal destacar que tot i les petites desviacions temporals que es van anar acumulant al tram final d'aquest projecte, s'han pogut solucionar i arribar a l'entrega final.

Aquest projecte és només la base d'una idea potencialment molt comercialitzable i que es pot estendre enormement. Per exemple, el piano podria tenir una pantalla connectada per tal de poder tenir un sistema d'ensenyar a tocar-lo. Una altra idea és la implementació d'una xarxa social com a tal, amb professionals i estudiants. Podria posar en contacte diferents perfils. També podria tenir una graella de músics per tal de fer concerts, on podria haver-hi un sistema de reputació de músics. Dins d'aquest context, també es podrien dinamitzar petites gires dins dels cercles d'espais per ciutats o països.

Aquest projecte, si acaba tenint una massa crítica d'usuaris pot seguir creixent ja que té un potencial de creixement molt gran.

A part de les possibles ampliacions de funcions, dins del sistema es podria realitzar una purgació de dades de la taula transaccions. Com ja es pot preveure, caldrà en algun moment descarregar de dades la taula en qüestió, bàsicament per fer més àgils les accions sobre aquesta. El model proposat permet fer aquesta neteja i deixar sols les transaccions necessàries per tal que el sistema segueixi sent consistent i la informació coherent.

També un altre punt seria acabar la implementació de la identificació mitjançant les xarxes socials de *Facebook* i *Twitter*.

Finalment, la valoració del projecte és molt positiva, ja que s'han assolit totes les tasques i el sistema de reserves ja és una realitat.

AGRAÏMENTS

Voldria agrair a Jordi Herrera Joancomartí l'ajut, la paciència i els consells que m'ha donat al llarg d'aquest projecte. M'agradaria agrair a l'empresa Click Piano per oferirme la possibilitat de dur a terme aquest projecte. També m'agradaria agrair els comentaris sempre constructius del meu entorn.

REFERÈNCIES

[1] Ley Orgánica 15-1999 de 13 de diciembre de Protección de Datos. Boletín Oficial del Estado, 1999.

[2] Documentació Flask. http://flask.pocoo.org/docs/0.10/, Últim accés 13-03-2015.

[3] Documentació Mysql workbench. https://dev.mysql.com/doc/workbench/en/, Últim accés 13-06-2015.

[4] Python Decorator. https://wiki.python.org/moin/PythonDecorators#What_is_a_Decorator, Últim accés 13-06-2015.

[5] Bootstrap. http://getbootstrap.com/components/, Últim accés 14-03-2015.

[6] Documentació Codeigniter. http://www.codeigniter.com/userguide3/, Últim accés 14-03-2015.

[7] jQuery. http://api.jquery.com/, Últim accés 14-03-2015.

[8] PHP cUrl. http://php.net/manual/es/book.curl.php, Últim accés 14-03-2015.

[9] Documentació Braintree. https://developers.braintreepayments.com, Últim accés 18-06-2015.

[10] Mysql. https://www.mysql.com/, Últim accés 21-06-2015.

[11] Postman. https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop, Últim accés 21-06-2015.

[12] Sphinx python documentation generator. http://sphinx-doc.org/, Últim accés 21-06-2015.

[13] The JavaScript Object Notation (JSON) Data Interchange Format. https://tools.ietf.org/html/rfc7159, Últim accés 28-06-2015.

[14] David Ferraiolo and Richard Kuhn. Role-based access control. In *In 15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[15] Paul Fremantle, Jacek Kopecky, and Benjamin Yowell Yousif Aziz. Web api management meets the internet of things. In *SALAD2015:services and applications over linked APIs and data.*, pages 213–220, 2015.

[16] David Goodger. PEP 0257 – Docstring Conventions. https://www.python.org/dev/peps/pep-0257/#what-is-a-docstring, Últim accés 21-06-2015.

[17] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf, Últim accés 20-06-2015.

[18] John Viega, Matt Messier, and Pravir Chandra. *Network Security with OpenSSL: Cryptography for Secure Communications*. O'Reilly Media, illustrated edition edition, June 2002.

[19] Spyros Xanthopoulos and Stelios Xinogalos. A comparative analysis of cross-platform development approaches for mobile applications. In *Proceedings of the 6th Balkan Conference in Informatics*, BCI '13, pages 213–220, New York, NY, USA, 2013. ACM.

APÈNDIX

A.1 Exemple de resposta del *webservice*


```
{
  "access_token": "ac7428b115285be17d57",
  "data": [
    {
      "capacity": 1593,
      "description": null,
      "name": "casa Rog",
      "medias": [
        {
          "id": 1,
          "name": "image"
        }
      ],
      "pianos": [
        {
          "id": 1,
          "model": "model1",
          "brand": "marca1",
          "name": "piano1",
          "description": "piano de prova"
        }
      ],
      "dimension": "1",
      "geoloc": "41.541473,2.095789",
      "address": "fake street",
      "services": [
        {
          "name": "wifi"
        },
        {
          "name": "wc"
        },
        {
          "name": "live_music"
        }
      ],
      "id": 1,
      "space.id": 1
    },
    {
      "capacity": 200,
      "description": "description new",
      "name": "ETSE",
      "medias": [
        {
          "id": 2,
          "name": "image"
        }
      ],
      "pianos": [
        {
          "id": 2,
          "model": "RTR6000",
          "brand": "Yamaha",
          "name": "super piano 2",
          "description": "aquest piano mola mes que el
              piano 1"
        }
      ],
      "dimension": "2",
      "geoloc": "41.550266,2.099019",
      "address": "UAB",
      "services": [
        {
          "name": "wc"
        }
      ],
      "id": 2,
      "space.id": 2
    }
  ],
  "ok": {
    "message": "OK_READ",
    "code": 200
  }
}
```


A.2 Estructura sencera base de dades

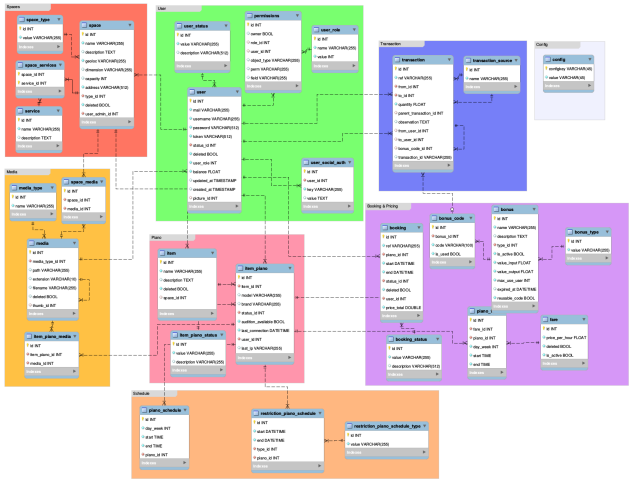

Figura 6: Disseny definitiu de la base de dades

A.3 Captures de pantalla del *backend* i *frontend*

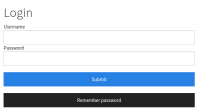

Figura 7: Finestra d'identificació.

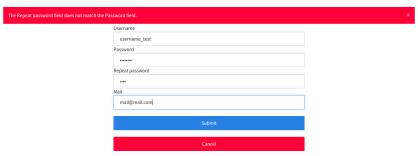

Figura 8: Finestra de registre d'usuari, amb error de contrasenya incorrecta.

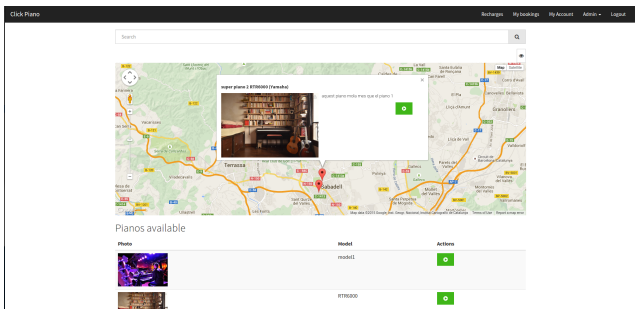

Figura 9: Finestra inicial, un cop realitzada la identificació.

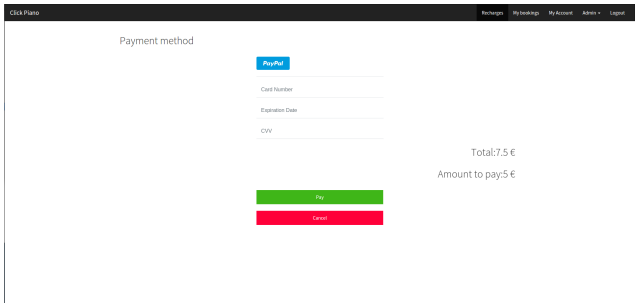

Figura 10: Finestra de pagament (al fer una recàrrega).

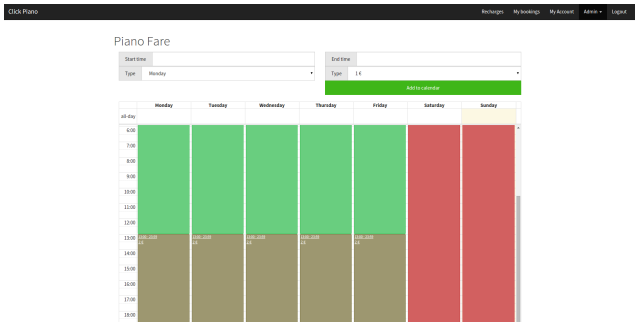

Figura 11: Finestra de gestió de tarifes segons horari per piano.

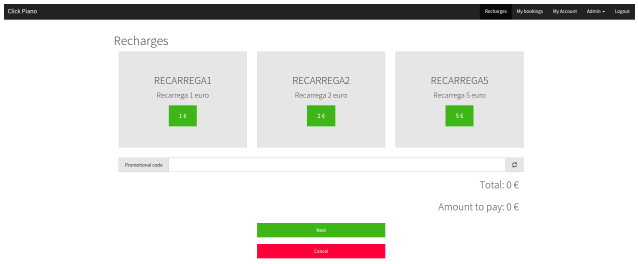

Figura 12: Finestra per escollir quina recàrrega realitzar.

A.4 Captures de pantalla de la documentació del *webservice*

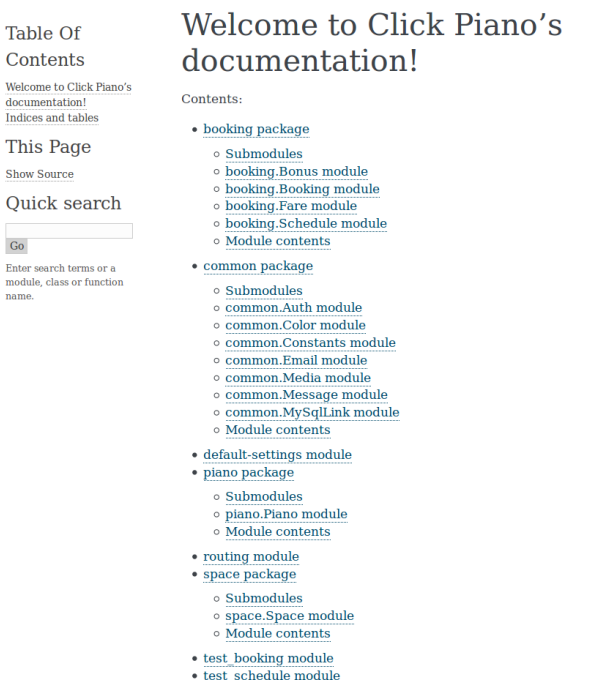

Figura 13: Índex de la documentació generada del *webservice*.

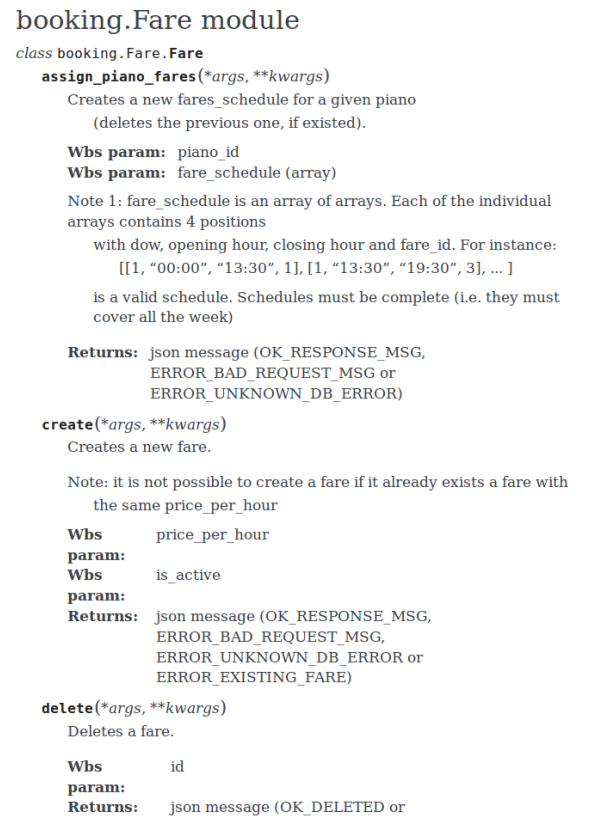

Figura 14: Exemple de la documentació generada del *webservice*.

A.5 Llistat de mètodes disponibles del *web-service*

Entitat	Acció
bonus	get_recharges
bonus	check_bonus_code
bonus	create
bonus	toggle
bonus	list
booking	create
booking	delete
booking	get_booking_price
booking	get_bookings
booking	play
booking	show_play_button
fare	create
fare	delete
fare	assign_piano_fares
fare	list_fares
fare	get_piano_fare_schedule
schedule	get_piano_schedule
schedule	get_piano_restrictions
schedule	get_restriction_types
piano	create
piano	delete
piano	read
piano	update
piano	alive
piano	search
space	create
space	delete
space	search
space	update
space	get_services
space	get_space_types
space	create_piano_schedule
space	update_piano_schedule
space	create_piano_restriction
space	delete_piano_restriction
space	get_list_spaces
transaction	get_transaction_token
transaction	make_payment
transaction	list_transactions
user	create
user	read
user	delete
user	search
user	update
user	get_bookings
user	login
user	validate_email
user	remember_password
user	change_password
user	get_my_pianos
user	get_my_spaces

Taula 1: Crides que es poden realitzar al *webservice*