Universitat de Lleida

Experiment d'Optimització de Sakai

Estudiant: **Raul Hidalgo Caballero**

Titulació: Grau en Enginyeria Informàtica

Títol de Treball Final de Grau: Experiment d'Optimització de Sakai

Director/a: Marta Oliva

Presentació

Mes: Setembre

Any: 2018

1. Índex:

2. Resum:

En aquest treball de final de grau s'ha intentat arreglar certs problemes relacionats amb el Campus Virtual que l'autor ha anat veient durant la seva estança a la Universitat de Lleida i de la seva intenció de crear una aplicació per a dispositius mòbils.

Aquests problemes encapsulen una gran quantitat de temàtiques, des de les més importants com la seguretat, fins a les més opcionals com poden ser menors temps de resposta. La majoria d'aquests problemes s'han detectat duran la creació de la mencionada aplicació mòbil.

Aquest document, està encara cap a la investigació de les causes d'aquests problemes. Tanmateix, s'ha intentat arreglar aquests problemes o almenys donar una explicació teòrica sobre com solucionar-los.

Aquest projecte va néixer de l'interès de l'autor d'aprendre Java i Android durant l'estiu de primer curs d'Enginyeria Informàtica, crear una aplicació per al Campus Virtual de la Universitat, ja que no existia. És important remarcar que el Campus Virtual de la UdL fa ús d'un sistema anomenat SAKAI de codi obert que bàsicament és un sistema gestor per a la intercomunicació entre alumnes i professors.

Aquest document, conte tota la investigació realitzada respecte el projecte SAKAI. Juntament amb explicacions de canvis realitzats al repositori de codi font del projecte.

Com a punt d'inici és important conèixer les següents URL:

- https://github.com/sakaiproject/sakai
  És el repositori de codi font principal del projecte Sakai. Conte l'històric de canvis al codi font, juntament amb informació encarada al desenvolupador.
- https://jira.sakaiproject.org/
  Instància de JIRA, on el projecte Sakai manté els problemes, peticions de canvis, etc...

3. Introducció:

1. Context:

Aquest projecte va néixer de l'interès de l'autor d'aprendre Java i Android durant l'estiu de primer curs d'Enginyeria Informàtica, crear una aplicació per al Campus Virtual de la Universitat, ja que no existia.

L'aplicació, és va crear satisfactòriament en dues meitats. Per un cantó, un client de Java per a l'API Direct de Sakai, l'API RESTful de Sakai. I per l'altre, una aplicació per a Android amb funcionalitats com sincronització i cache local de recursos, polling de qualificacions i avisos, etc...

Aquesta aplicació es va distribuir en format APK entre els companys de classe. D'aquesta forma podia aconseguir feedback molt útil. Una de les millores que s'hem suggerien era que no existia per a IOS.

Així doncs a l'estiu de 2º curs a 3º vaig procedir a portar l'aplicació al framework Xamarin, amb la qual cosa suportava qualsevol dispositiu mòbil.

Amb més experiència, i l'aplicació més optimitzada, hem vaig veure amb cor de llançar-la com a beta privada a la Play Store.

L'aplicació, estava en un estat bastant bo, però no finalitzada i preparada per a ser llançada de forma pública.

Durant l'estiu de 3º curs a 4º, vaig començar a fer els preparatius per a considerar-la el meu treball de fi de grau.

En discutir la implementació de l'aplicació, es van trobar certs errors en la seva implementació. L'error principal, era el mecanisme que s'utilitzava per autentificar l'usuari de l'aplicació. El mecanisme, feia ús de l'API, però requeria que l'usuari guardes les seves credencials dins l'aplicació. Encara que s'utilitzessin els mecanismes de seguretat del SO en qüestió, era necessari tenir accés a la contrasenya sense encriptar. Per la qual cosa, es va veure que clarament era necessari modificar el codi font de Sakai, ja que no suportava mètodes d'autentificació diferents d'usuari i contrasenya.

Ja que entràvem en l'obligació de modificar el codi font del projecte Sakai o utilitzar una implementació considerada insegura, es va decidir que el treball final de carrera és centres en realitzar les modificacions necessàries per a poder implementar una aplicació de forma eficient.

Aquestes modificacions, són funcionalitats o canvis que es consideren necessaris per a tenir una aplicació rapida, eficient i segura, entre els quals podem anomenar:

Un mecanisme per a poder autenticar i autoritzar aplicacions de tercers de forma segura i acceptada per la comunitat de desenvolupadors, a causa que les contrasenyes havien de ser guardades dins del dispositiu com a text pla.

Una millora en l'API que no facilites l'ús d'aquesta mentre al mateix temps no causes sobrecarrega en els servidors a causa d'una gran quantitat d'aplicacions instal·lades, ja

que la preocupació de fer un ús excessiu de l'API rellentis els servidors podia ser un punt en contra per al seu desplegament.

La idea de millorar la mateixa interfície web per tal que no fos necessària la creació d'aplicacions per a dispositius mòbils, ja que l'aplicació en el fons tan sols necessita fer ús de mecanismes proporcionats per navegadors moderns.

Millorar la facilitat de desplegament del sistema a l'igual que habilitat una més fàcil orquestració del sistema en general, per tal d'auto balancejar la carrega en casos una sobrecàrrega d'usuaris.

A més, altres sistemes a part de l'aplicació es poden aprofitar d'aquestes millores.

2. Objectius del Treball:

Els objectius del treball és investigar quins canvis requeriria el projecte Sakai per tal de poder estar preparat per a implementar qualsevol tipus d'aplicació per a dispositius mòbils. Així doncs es resumeixen de la següent manera:

Investigar un mecanisme de Single Sign On per al Sakai, que sigui segur i àmpliament utilitzant, per a implementar-lo en un futur si la comunitat decideix acceptar el canvi, aportant un major grau de seguretat i permeten un mecanisme per a que aplicacions de tercers es connectin a la instància de Sakai.

Investigar un mecanisme per a millora l'API de Sakai, que sigui rapida per tal de tenir velocitats de resposta més rapides, fàcilment ampliable per tal de facilitar la possible ampliació de l'API, i a ser possible que generi la menor quantitat de sobrecàrrega al sistema, ja que una major quantitat de dispositius accedint-hi no comporti problemes de rendiment.

Investigar la possible aplicació de nous mecanismes per evitar la creació d'una aplicació per a dispositius web. Noves tecnologies s'estan obrint camí en el punt mig entre aplicacions natives i webs. Estudiar quins beneficis i inconvenients comporten aquestes tecnologies poden ajudar a simplificar tot el projecte.

Un cop s'hagi realitzat tota la investigació pertinent, s'intentarà realitzar el màxim de canvis possibles relacionats amb aquestes temàtiques dins el repositori del projecte, sempre que la comunitat de Sakai estigui disposada a acceptar els canvis.

Dins d'aquests canvis, també s'integren tots aquells canvis que es considerin bàsics o de manteniment del projecte.

4. Estat de l'Art:

Al començament del projecte, es va realitzar un estudi de l'art. El propòsit d'aquest estudi, era obtindré tots els coneixements necessaris per poder desenvolupar aquest projecte. Aquest estudi es basa en l'estructura del mateix projecte Sakai, i de múltiples tecnologies necessàries per a comprendre'l completament.

3. Estudi sobre Sakai:

Sakai representa un enfocament fonamentalment diferent del sistema de gestió de l'aprenentatge. A diferència d'altres sistemes "oberts" disponibles avui, la direcció i el conjunt de característiques de Sakai provenen de l'ensenyament superior per abordar les necessitats dinàmiques d'una comunitat acadèmica global. La comunitat de codi obert de Sakai valora molt la participació dels seus col·laboradors, amb educadors i desenvolupadors de diverses institucions que treballen conjuntament per convertir les grans idees en realitat per a tota la comunitat de Sakai.

Sakai fa ús de les següents tecnologies de forma general:

- **Java:**

Java és un llenguatge de programació de propòsit general, concurrent, orientat a objectes, que va ser dissenyat específicament per tenir tan poques dependències d'implementació com fos possible.

Es basi en la idea de compilar una vegada, i executar a qualsevol arquitectura o sistema operatiu. Això es basa en la JVM, que bàsicament interpreta l'assemblat resultat intermedi i l'executa per a aquella arquitectura i sistema operatiu.

- **Maven:**

Maven és una eina de programari per a la gestió i construcció de projectes Java. És àmpliament utilitzat, ja que emmascara tota la complexitat de gestionar projectes, dependències, processos de compilació i creació d'assemblats.

- **JavaEE:**

Framework de desenvolupament de serveis de aplicacions empresarials. Bàsicament, es una extensió de l'API de Java que encapsula totes aquelles APIs relacionades amb el desenvolupament de aplicacions empresarials. Generalment, un servidor d'aplicacions compatible amb JavaEE es el encarregat de implementar aquesta API.

- **Spring:**

Spring és un framework per al desenvolupament d'aplicacions i contenidor d'inversió de control, de codi obert per a la plataforma Java. Bàsicament, és un framework per a la injecció de dependències, que és un patró que permet la creació de sistemes molt desacoblats. De forma més entenedora, Spring uneix diferents classes a través d'anotacions o d'acord amb uns fitxers XML. Aquests elements són injectats als elements que els requereixen.

Com a extensions de Spring, existeixen un conjunt de llibreries que faciliten la creació de webs, com MVC, Data Repositories, Security, Containers, etc.

- **Hibernate:**

Hibernate és una eina de mapeig objecte-relacional (ORM) per a la plataforma Java que facilita el mapeig d'atributs entre una base de dades relacional tradicional i el model d'objectes d'una aplicació, mitjançant arxius declaratius (XML) o anotacions en els beans de les entitats que permeten establir aquestes relacions.

- **Tomcat:**

És un servidor d'aplicacions web per a JavaEE. Dit d'una altra forma, Tomcat, és la implementació de l'API JavaEE juntament amb altres funcionalitats exteriors a JavaEE.

Conte un motor de Servlets, JSP Pages, Java Expression Language i WebSockets, que bàsicament són les parts bàsiques per implementar JavaEE juntament amb d'altres.

- **API REST:**

Estructura per exposar una API a través de HTTP. Entre els seus avantatges és que s'implementa sobre un protocol sense estats, les operacions més importants estan basades en el protocol de capa inferior, GET, POST; PUT i DELETE. L'API de Sakai fa ús d'XML i JSON, cosa que facilita la transferència entre diferents sistemes que no tenen per què estar programats en el mateix llenguatge.

- **SOAP:**

SOAP (Simple Object Access Protocol) és un protocol estàndard que defineix com dos objectes en diferents processos poden comunicar-se mitjançant l'intercanvi de dades XML. La gran complexitat del sistema SOAP ha causat que entri en desús, ja crear una implementació és molt costos de desenvolupar.

Estructura del projecte:

Com a forma general, el projecte s'estructura de la següent manera:

Dins el repositori, a la base de la qual ens referirem com a "/" existeixen els fitxers comuns bàsics d'un projecte Git, entre els quals podem nomenar els següents: .gitignore, README.md, LICENCE.txt, etc.

Annexat a la base existeix un fitxer "pom.xml" que anomenarem "base", aquest fitxer l'orquestrador del projecte, conte referències a tots els altres mòduls annexats a l'arrel del projecte, i és on es realitza la majoria de comandes que són transmeses a les seves referències.

Annexat a la base, existeixen carpetes de projecte, que anomenarem mòduls, aquests mòduls, representen funcionalitats o parts separades del projecte Sakai, la seva totalitat representa tot el codi bàsic de Sakai.

Un mòdul, generalment està format per tres submòduls i un fitxer "pom.xml" que representa el mòdul. Els tres submòduls són els següents:

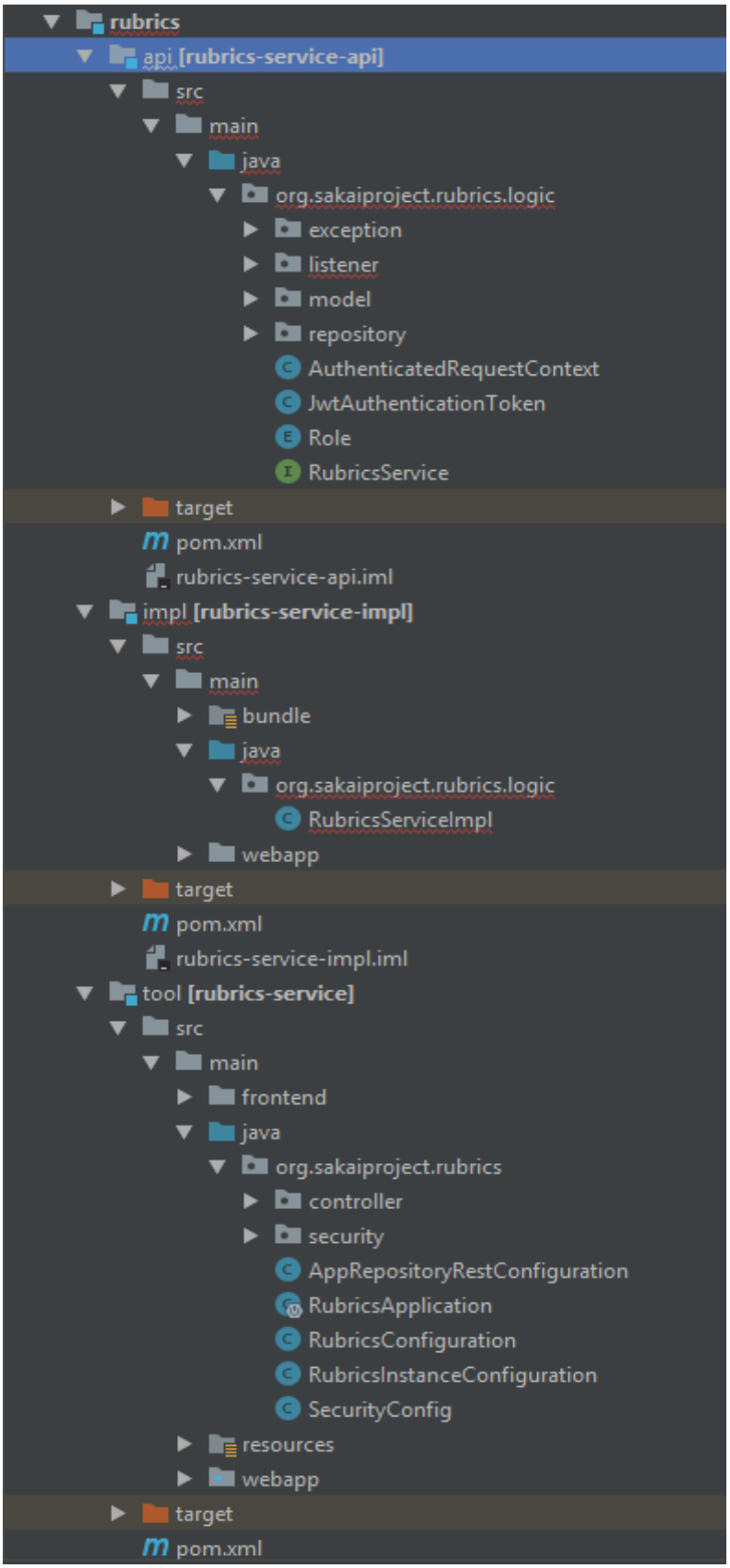
API: Conte el codi que no pot ser canviat i que no depèn de la implementació, juntament amb les interfaces per ser implementat. De forma més entenedora, és la "Application Programing Interface" d'un mòdul.

IMPL: La implementació per defecte de l'API. Implementa totes les interfaces referenciades al submòdul API i permet inferir una lògica especifica sense interferir en la signatura de l'API.

TOOL: Implementa la part d'un submòdul que s'exposa a l'usuari, generalment via HTTP, far les crides a l'API que indirectament executa codi del submòdul de tipus IMPL que estigui activat.

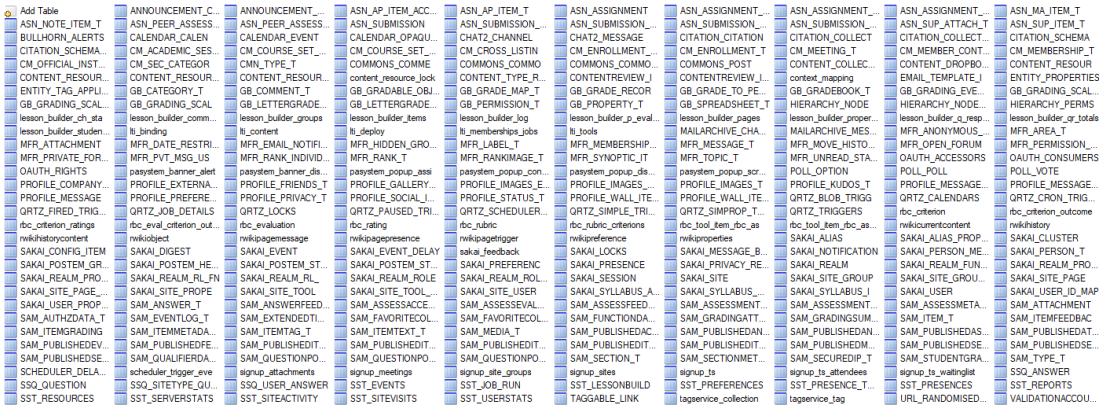
Existeix un mòdul especial, anomenat "master", dins la carpeta "/master" que conte tota la informació bàsica de qualsevol mòdul o submòdul. Conte informació sobre l'estructura de qualsevol projecte, llibreries per proveïdes, versions, configuracions de configuració, etc.

A continuació podem veure un exemple d'un mòdul anomenat "rubrics" on es veu l'estructura interna del mòdul i els seus submòduls.


rubrics
api [rubrics-service-api]
src
main
java
org.sakaiproject.rubrics.logic
exception
listener
model
repository
AuthenticatedRequestContext
JwtAuthenticationToken
Role
RubricsService
target
pom.xml
rubrics-service-api.iml
impl [rubrics-service-impl]
src
main
bundle
java
org.sakaiproject.rubrics.logic
RubricsServiceImpl
webapp
target
pom.xml
rubrics-service-impl.iml
tool [rubrics-service]
src
main
frontend
java
org.sakaiproject.rubrics
controller
security
AppRepositoryRestConfiguration
RubricsApplication
RubricsConfiguration
RubricsInstanceConfiguration
SecurityConfig
resources
webapp
target
pom.xml

Estructura de la base de dades:

A continuació es mostren totes les taules que una instància de Sakai autogenera. Això ens ajuda a entendre l'envergadura del projecte a més d'ajudar-nos a entendre les relacions entre entitats que existeixen en l'àmbit de dades.



Una instància neta genera exactament 300 taules.

Un estudi més profund de les entitats ha revelat que la base de dades s'ha normalitzat de forma molt granular, no es fa ús de tipus de dades específics del tipus de base de dades concret, i es fa un ús innecessari de claus primàries basades en caràcters.

5. Desenvolupament:

4. Seguretat – Investigació:

Estat Actual:

En aquest apartat, estudiarem l'estat actual de la seguretat del sistema Sakai específicament l'apartat de la seguretat que tracta l'autentificació. La intenció d'aquest apartat és recollir com més informació possible per veure com es comporta el sistema en la seva totalitat. A més, servirà per identificar diferents vectors d'atac que puguin existir, i veure en quins llocs és més probable que succeeixin, per tal de canviar aquests sistemes, augmentant la seva seguretat abans que puguin ser descoberts.

En l'apartat de "Features" de la pàgina web de Sakai es comenta que com a sistema d'autentificació utilitzen múltiples estàndards de Sign-On com CAS, Kerberos, SAML i Shibboleth. Tanmateix, la meva experiència personal tractant amb el projecte hem permet saber que també hi ha OAuth1.0, i LDAP.

Sobretot en l'apartat de seguretat no s'ha trobat gaire documentació, això, en certa forma és un error, ja que el desarrollador que acabi implementant la connexió amb altres sistemes pot cometre errors a causa de desconeixement. I aquest és l'error de seguretat més comuna i simple que existeix.

Seguidament, farem una explicació simple de què és cada estàndard mencionat:

- **OAuth1.0:**

És un estàndard que explica el flux d'autorització i autentificació. Depreciat l'any 2012 a favor de la versió 2.0. Actualment només es realitza manteniment de seguretat en les llibreries que l'implementen.

- **CAS:**

És un sistema d'autorització propi de l'empresa Apereo, desarolladora del projecte Sakai. El seu funcionament és de SSO i de Proxy a altres mecanismes d'autentificació.

- **Shibboleth:**

És un sistema de SSO basat en federació i dominis de seguretat, fa un alt ús de SAML.

- **SAML:**

Security Assertion Markup Language, és un sistema d'intercanvi d'informació relacionada amb l'autentificació i autorització, basat en XML.

- **Kerberos:**

Protocol de xarxa encarat a l'intercanvi d'informació de forma segura a través d'una xarxa considerada no segura.

- **LDAP:**

És un protocol de nivell d'aplicació a accedir un directori de serveis a través del protocol IP. Molt utilitzat en sistemes d'Intranet, ja que facilita la compartició de la informació de l'usuari a treves de la xarxa.

Seguidament, anem a veure una mica quins són els punts d'entrada principals al sistema, ja que ens ajudaran a entendre com funciona realment el sistema, en quins punts és més fort, i en quins punts és més dèbil.

- **/portal/xlogin**

És el punt d'entrada principal de la majoria d'usuaris. En cas d'intentar accedir a un punt que necessita autentificació, l'usuari és redirigit aquí. Identificació guardada a la sessió de JavaEE.

- **/dav/{nom-de-lloc}**

Punt d'entrada a l'accés del contingut a través de WebDAV, aquest punt fa ús del mecanisme d'autorització per defecte de HTML i HTTP, la contrasenya, és enviada codificada en BASE64 a través dels headers de HTTP.

- **/oauth-tool/authorize**

Punt d'entrada a través d'OAuth1.0. OAuth1.0 és un protocol considerat insegur, a causa que existeix un successor directe que millora tots els aspectes de la usabilitat i seguretat. No totes les implementacions de Sakai la tenen activada.

- **/direct/session/new?_username={usuari}&_password={contrasenya}**

Punt d'entrada a través de l'API REST, com es pot veure pel seu format, la informació d'usuari és enviada a través de paràmetres query de l'URL.

Com a forma general, podem concloure que entre l'usuari i el Sakai, l'única seguretat és el protocol SSL que hauria d'haver-hi entremig. Encara que no és obligatori per tal de poder fer funcionar Sakai.

Sobre OAuth, a part que s'utilitza la versió del protocol V1, també s'ha trobat que s'està utilitzant una implementació extremadament antiga. Mirant el fitxer "/oauth/pom.xml" es pot veure com s'està utilitzant un paquet anomenat "net.oauth.core", i la versió "20100527".

Fen una cerca, es pot veure com la seva publicació és del 2011, i admès, existeixen 2 actualitzacions publicades el 2016. Amb la qual cosa podem veure que fa molt temps que s'hauria d'haver fet la migració a un sistema millor, o com a mínim actualitzar la implementació.

Estat Proposat:

En aquest apartat s'explica la idea de l'autor sobre com millorar aspectes en la seguretat que es consideren imprescindibles. Juntament amb els motius i explicacions que es considerin necessaris.

Després d'haver tractat directament amb el sistema d'autentificació de Sakai a causa d'inicialment intentar realitzar el treball de fi de grau creant una aplicació mòbil s'ha arribat a la conclusió que Sakai, no està adaptat per a terceres persones interactuïn amb el sistema.

Després d'haver mirat diferents mecanismes per solucionar tots aquests problemes s'ha arribat a la conclusió de què difícilment hi ha un mecanisme perfecte, però si una millora que arreglaria gran part dels problemes. Aquesta solució és fer ús d'OAuth2.0 juntament amb JWT.

En la majoria de casos d'ús de Sakai, hi ha dos escenaris:

- La informació de l'usuari és administrada per Sakai
- La informació de l'usuari és delegada a un altre sistema. Ex: LDAP

Amb OAuth2.0 hi hauria tres escenaris:

- La informació de l'usuari és administrada per Sakai. OAuth2.0 integrat.
- La informació de l'usuari és delegada a un sistema OAuth2.0. OAuth2.0 no integrat.
- La informació de l'usuari és delegada a un altre sistema. OAuth2.0 integrat.

Cobrin aquests tres escenaris, es podria assegurar que des del punt de vista de l'autentificació i autorització sempre es faria ús d'OAuth2.0. Aquesta aproximació, faria que obligatòriament es fes us complet d'OAuth2.0 i no de forma opcional, simplificant la lògica i els costos de desenvolupament.

Així doncs en el primer cas, Sakai es converteix en un proveïdor de recursos, i a la vegada com a servidor d'OAuth. En el segon és simplement el proveïdor de recursos. I en el tercer és com el primer cas però delega responsabilitats a altres sistemes.

S'ha elegit OAuth2.0, perquè és considerat el sistema més utilitzat i que cobreix més casos d'ús dins la categoria de SSO.

Uns dels avantatges d'OAuth2.0 és que una implementació correcta i que compleixi els apartats opcionals, un client es pot configurar automàticament tan sols saben la seva pròpia informació. A més proveeix del sistema OpenID sense esforç causant que un usuari pugui usar la seva conta de Sakai com per exemple la UdL per a fer sign-in a sistemes que ni tan sols coneixen de l'existència de Sakai.

La gràcia d'OpenID, és que està pensat per a que qualsevol institució pugui convertir-se en proveïdora d'identitat.

Una de les altres avantatges d'OAuth2.0 és que ofereix nativament un sistema per a donar permisos a aplicacions de tercers. L'avantatja d'aquest fet és tenir control granular sobre les accions que poden fer les aplicacions de tercers. El sistema és capaç de

detectar quan una acció l'esta fen un usuari real i quan és una aplicació en nom d'un usuari.

Per la part de les JWT podem dir que és un sistema que identifica l'usuari dins de l'aplicació de tercers però ho fa de forma transparent a l'aplicació. Dit d'una altra forma, són tokens que tenen una càrrega útil que identifica l'usuari de forma pública, i una signatura que és validada contra l'emissor del token.

Una forma millor d'explicar perquè té gran utilitat és donar un cas real.

Imaginem que tenim una gran quantitat d'usuaris de Sakai, i tots volen entrar a la vegada. Per tal de permetre això, tenim deu instàncies de Sakai, aquests Sakais s'exposen a través d'un balancejador de càrrega.

Sense les JWT, el balancejador de càrrega té per a processar segons la cookie JSESSION o per IP a quina instància redirigir la petició, ja que la sessió és mantinguda a través d'un diccionari Clau->Usuari.

Amb les JWT, el balancejador de càrrega no ha de preocupar de a quina instància s'han redirigit anteriorment. Ja que les instàncies coneixen la clau d'encriptació i validen la informació continguda a les JWT, ajudant a un millor escalament horitzontal.

Dit d'una altra forma, amb les JWT els balancejadors de càrrega poden tenir menys intel·ligència i ser més simples sense haver d'afegir gran complexitat com per exemple guarda el diccionari Clau->Usuari a una base de dades de cache clau-valor.

Vulnerabilitats Trobades:

Contrasenyes de 8 caràcters fixes:

Aquesta vulnerabilitat només afecta la instància de Sakai de la UdL, però una seguretat més alta en el projecte Sakai hauria evitat que pugues ser explotada a través de Sakai.

Tal com indica el títol, la instància de Sakai de la UdL es basa en una contrasenya de < 8 caràcters.

De forma simple, el sistema només considera els 8 primers caràcters. Significant que la contrasenya "LaMevaContrasenyaSuperSegura" només és efectiva la part "LaMevaCont".

Aquest fet en si mateix no implica un error greu, sinó el fet que tampoc té mecanismes per evitar atacs per força bruta.

La fortalesa d'una contrasenya, es basa en la indeterminació i l'aleatorietat. Com més informació coneguem respecte a la seva forma, més combinacions podem eliminar.

Tècnicament, aquesta vulnerabilitat causa que hi hagi (255^8) + (255^7) + (255^6) + (255^5) + (255^4) + (255^3) + (255^2) + (255^1) = 17948489581465697280 combinacions de contrasenyes possibles. Però, és realment així? Un usuari ficarà una tabulació enmig de la seva contrasenya? I un caràcter null?

Els usuaris, no són ordinadors, que aleatoritzen les seves contrasenyes. Utilitzen certes regles. I les utilitzarem per reduir de forma notable les combinacions.

- Regla Núm. 1: Els usuaris fan anar contrasenyes de 8 o més caràcters. Gràcies als mecanismes que obliguen que la gent tingui contrasenyes de 8 o més caràcters, els usuaris per defecte creen les seves contrasenyes amb una longitud més gran o igual a 8 caràcters. D'aquesta forma, podem inferir que la contrasenya efectiva serà d'exactament 8 caràcters. (255^8)= 17878103347812890625. Això és una reducció de 70386233652806655 combinacions.
- Regla Núm. 2: Els usuaris només fan ús de caràcters que estan en un teclat normal. Això són (10 + 26 +26)^8 = 218340105584896. Això és una reducció de 17877885007707305729 combinacions, cada vegada més.

A partir d'aquí, acaba la secció d'assercions, i comença l'apartat on es dóna prioritats.

- Regla Núm. 3: Es prioritza el ús de lletres minúscules.
- Regla Núm. 4: És té costum a què si existeix una majúscula és el primer caràcter.
- Regla Núm. 5: Els números, tenen tendència a sortir al final de les contrasenyes.
- Regla Núm. 6: Les vocals es prioritzen a les consonants.
- Regla Núm. 7: Certs caràcters són poc comuns, com per exemple "zxyw"
- Etc.

Totes aquestes regles redueixen de forma considerable les verdaderes combinacions possibles.

Existeixen dues formes de protegir el sistema, les dues s'haurien d'aplicar al mateix temps, ja que no són mútuament exclusives.

Implementar un sistema de captcha segur eliminaria la facilitat d'enviar múltiples peticions i que fossin transmeses a la següent capa de la instància. Actualment, captchas com "recaptchaV2" aconsegueixen ser transparents per a l'usuari, però en cas de tenir dubtes, mostren un captcha per assegurar que és realment un usuari. Aquest mecanisme seria perfecte per les pàgines on als usuaris se'ls sol·licita una contrasenya.

D'altra banda, no tots els accessos seran realitzats a través de les pàgines destinades als usuaris. L'API i WebDAV, són dos casos. Tot i que pel cas de l'API es recomana fer ús d'OAuth2.0 existeix un mecanisme per evitar atacs de força bruta.

El millor mecanisme és la creació d'un filtre global per IP que només permetés cert nombre de peticions concurrents al mateix temps.

Existeix la creença de què crear un delay en una petició fallida de sign-in és una mesura efectiva. En realitat és una mesura contraproduent, ja que només és efectiva si les peticions són obligatòriament síncrones. Si es realitza un bloqueig per IP durant un període de temps en superar un límit de peticions per unitat de temps, s'invalida la possibilitat efectiva de realitzar atacs de força brut efectius. Per exemple, si se superen les 50 peticions per segon des d'una mateixa IP, es considera que s'està sofrint un atac i es bloqueja la IP durant 1 minut. Això provoca que cada 50 peticions es bloquegi el sistema provocant que la velocitat màxima de validació sigui de 50 combinacions per minut, una velocitat extremadament baixa que causa que no sigui efectiu realitzar atacs d'aquest calibre.

LDAP Overflow:

Aquesta vulnerabilitat s'ha trobat després d'executar l'exploit de la vulnerabilitat anomenada "Contrasenyes de 8 caràcters fixes".

Aquesta vulnerabilitat és causada per una sobrecàrrega en el sistema LDAP, causant que tot el sistema de la UdL ningú pugui fer sign-in. Això afecta tant al Sakai, Correu, com altres sistemes de sign-in que requereixen el sistema LDAP.

A causa que la vulnerabilitat no pot investigar-se amb més profunditat sense causar greus problemes a tota la Universitat de Lleida, no s'investigarà més fen proves amb l'exploit. Ja que en el moment de la investigació és període de matriculacions i aquesta vulnerabilitat deshabilite que els usuaris pugin fer el que es coneix com a "Automatrícula".

A contínua s'expliquen mecanismes per securitzar el sistema i evitar aquest Overflow.

En primer lloc, la forma més fàcil d'evitar una sobrecàrrega és ser capaç de gestionar més peticions. Segurament, s'hauria d'actualitzar el software i el servidor que l'executa, ja que hauria de ser capaç de gestionar més dels 10Mbps necessaris per sobrecarregar-lo.

En segon lloc, realitzar un filtre per usuari en què mes de X peticions no es puguin fer al mateix temps. Explicat d'una altra forma en sobrepassar més de 10 peticions d'un mateix usuari per segon, denegar-les totes durant els següent minut. Això evitaria que fos

possible invalidar qualsevol exploit de contrasenyes d'aquest tipus. Ja que l'atacant sabria que seria possible que envies la contrasenya correcta però que fos marcada com a incorrecta.

5. Seguretat – Experimentació:

Contrasenyes de 8 caràcters fixes i LDAP Overflow:

En aquest apartat tractarem l'apartat experimental sobre seguretat relacionat amb la vulnerabilitat dels 8 caràcters i l'Overflow de LDAP.

La idea original era escriure un motor per a generar diccionaris que generessin contrasenyes amb intel·ligència inferida. Seguint prioritats establertes comuns a la majoria de contrasenyes.

Un motor per a realitzar peticions HTTP de forma asíncrona a la màxima velocitat que oferís la xarxa, per tal de maximitzar el nombre de comprovacions de contrasenya per unitat de temps.

Un sistema d'orquestració entre diferents instàncies d'aquest programa, repartint-se les tasques que havien de comprovar per tal de poder escalar horitzontalment el sistema. Podent fàcilment afegir VMs amb grans capacitats de gestió de xarxes.

El desenvolupament d'aquest exploit ha sigut pausat de manera indefinida en el punt en què es generaven contrasenyes mitjanament simples i s'estava programant el motor de peticions HTTP, ja que a aproximadament un atac a 10Mbps l'atac causava una sobrecàrrega del sistema LDAP, causant que cap usuari pugues entrar en cap sistema a causa de no poder fer sign-in.

Per la qual cosa s'ha decidit que ja és té suficient informació sobre la vulnerabilitat i que es coneixen suficients mecanismes per a poder arreglar la vulnerabilitat.

6. Optimització – Investigació:

Estat Actual:

En aquest apartat, estudiarem l'estat actual del rendiment del sistema Sakai. La intenció d'aquest apartat és recollir com més informació possible per veure com es comporta el sistema en la seva totalitat.

El nostre punt d'estudi, serà l'API, ja que és el punt que més ens interessa des del punt de vista de la creació d'una aplicació per a dispositius mòbils.

L'API del projecte Sakai és l'URL "/direct", aquesta és una API RESTful.

Està dividida en múltiples entitats, que representen els recursos que exposa el sistema.

Generalment, compleix amb totes les convencions que especifica RESTful.

Una de les millors formes de trobar problemes, és donar un cas d'ús, i veure com de bé o malament és capaç de treballar el sistema. El cas d'ús, ha de ser un cas perfectament plausible.

Donat que estem autenticats i autoritzats, volem saber el nom i l'URL de tots els fitxers de l'assignatura ALGEBRA.

Suposant que no tinguéssim al client cap tipus de cache, hauríem d'anar primer a "/direct/site.json" per poder llistar l'identificador de tots els recursos. Ja que no podem sol·licitar directament un amb el nom "ALGEBRA".

El primer que veiem, és que tot i que esperem una llista d'objectes SITE, en realitat el que rebem és un Objecte, format per "entityPrefix" i "site_colection", això, és completament innecessari, ja que en recuperar el llistat de Sites, ja sabem que són sites, i junta nom del recurs mes collection, requereix un treball una mica més complex. Ficats a ser estrictes, són 38 caràcters completament innecessaris que tant sòl augmenten la complexitat sense aportar cap valor.

Pel nostre cas, tant sòl necessitem l'identificador i el nom del Site. Ja que filtrarem pel nom, i ens quedarem amb l'identificador del qual encaixi.

Tot i que la nostra acció és simple, hem sigut bombardejats amb tot tipus d'informació del Site, per a cada un dels Sites en què tenim algun tipus d'accés. En el nostre cas, hem rebut 383KB dels quals segurament necessitem menys de 500 bytes.

Bé, ara ja tenim l'identificador del Site anomenat ALGEBRA, és hora d'intentar accedir al seu contingut, per això, anem a "/direct/content/site/{site-id}.json".

Tanmateix, ens trobem amb el mateix problema, esperem una llista d'objectes, però rebem un objecte que encapsula la llista de forma innecessària, a més, el nom de la col·lecció ha canviat a "content_collection".

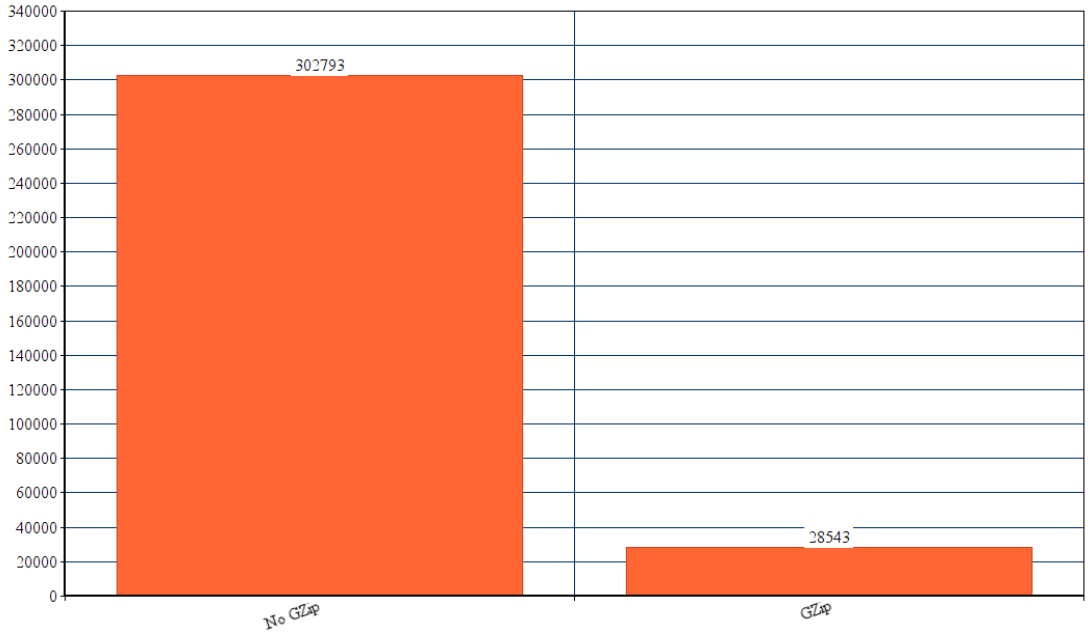
Igual que en l'anterior cas, som bombardejats amb informació que realment no ens interessa.

Després de 2 peticions HTTP, uns 400KB d'informació, i unes quantes peticions a la base de dades, ja tenim el que necessitàvem.

Cal mencionar en favor del sistema Sakai, que aquesta fen un molt bon ús sobre els sistemes de cache per a fitxers estàtics però el no es fa cap tipus d'ús de cache en l'API, ni de cap tipus d'encoding per tal de minificar el consum d'amplada de banda.

Com a exemple, el resultat de "/direct/site.json" genera 302793 bytes, minificat amb el sistema GZip que tots els navegadors entenen i envien per defecte, és minificar a tan sols 28543 bytes. Això significa una reducció d'un 1061% respecte a la mida original.

En la imatge següent es mostra la comparativa per a una petició de "/direct/site.json" sense GZip i una amb GZip.



Amb el projecte, s'entrega també dues llibreries clients per a la mencionada API REST, una per a Java, i una per a .NET:

https://github.com/deinok/sakai-api-java

https://github.com/Sakzilla/Sakai.ApiClient

Estat Proposat:

En aquest apartat s'explica la idea de l'autor sobre com millorar aspectes relacionats amb el rendiment que es consideren una millora. Juntament amb els motius i explicacions que es considerin necessaris.

Aquest apartat estudia en profunditat el cas de l'API, ja que una aplicació per a dispositius mòbils farà un ús intensiu d'aquesta part del projecte Sakai i una bona API pot ajudar en la simplificació del treball tant en el que es coneix com a backend com frontend, ja que és el punt d'unió entre les dues meitats.

Després d'estudiar l'API en profunditat, es proposa fer una nova versió de l'API. Aquesta nova API, faria ús del protocol GraphQL, una forma d'exposar l'API a través de HTTP que ofereix una gran quantitat de millores per damunt de REST i RESTful.

Seguidament, s'explicarà els punts on es millora respecte a una API RESTful.

GraphQL té un únic entrypoint, això significa que totes les peticions es fan a una sola URL, simplificant la complexitat de tindre múltiples URLs per cada acció i recurs. L'entrypoint és normalment "/api/graphql" o "/graphql".

GraphQL té un sistema d'introspecció, això, significa que pot realitzar-se una petició per tal de conèixer de forma estàndard com està formada l'API i poder generar models per aquest. El sistema podria equivaldre al WSDL de SOAP.

GraphQL està encarat a modelitzar les coses com a grafs, això és una forma totalment diferent de l'API RESTful, però molt més propera a la realitat del que interessa a l'usuari. A continuació veiem un exemple més entenedor d'una petició.

```
{
  hero {
    name
    friends {
      name
    }
  }
}
```

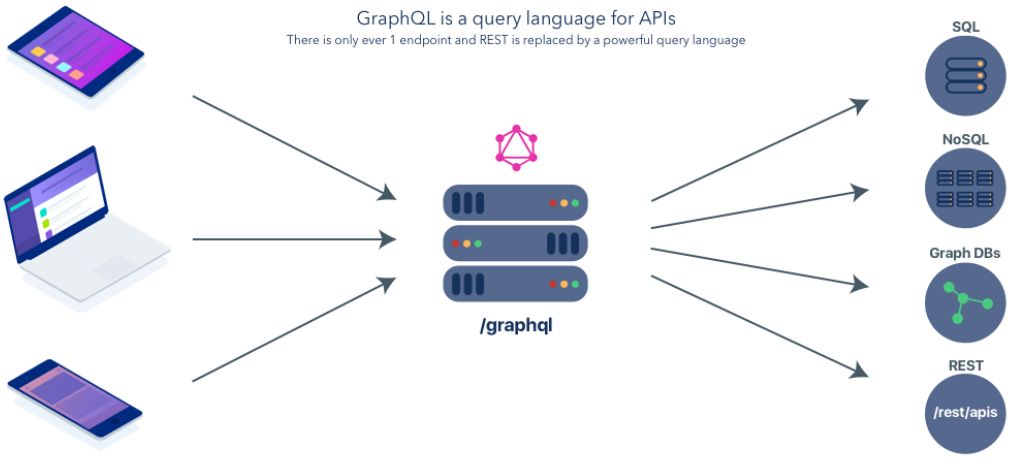
```
{
  "data": {
    "hero": {
      "name": "R2-D2",
      "friends": [
        {
          "name": "Luke Skywalker"
        },
        {
          "name": "Han Solo"
```

GraphQL tan sols retorna el que l'usuari ha demanat, ni mes, ni menys. Això, és el punt més fort de GraphQL, ja que el servidor de GraphQL sap exactament el que vol el client, i pot recuperar tan sols la informació que interessa. Aquesta optimització, permet que un servidor GraphQL ben implementat, és capaç de convertir una consulta de GraphQL a una consulta SQL millorant el rendiment de la base de dades, del servidor i optimitzant l'amplada de banda entre client i servidor.

GraphQL separa el concepte de peticions de lectura a les d'escriptura. Aquests, són denominats "querys" per a la lectura, i "mutations" per a les escriptures. Això permet que un programador que desconeix la implementació interna pugui causa efectes secundaris sense saber-ho.

GraphQL permet un mecanisme anomenat "batching" que de forma simple, permet multiplexar múltiples peticions en una sola petició HTTP, millorant els temps de resposta.

GraphQL permet ser programat fàcilment com un Gateway, el que permet la creació d'APIs GraphQL fàcilment utilitzant diferents tipus d'APIs exposant-los a través d'una sola API. A continuació veiem una infografia representativa d'aquest concepte.



GraphQL permet mecanismes per a prioritzar certes parts d'una petició i que la informació més prioritària arribi abans al client que la que és costosa i pot tindre latències notables.

GraphQL permet obtindré informació en temps real. Un client pot subscriure's a una informació i d'aquesta forma ser notificat de qualsevol canvi en temps real. Aquest mecanisme és anomenat "subscriptions" i s'implementa a través de WebSockets.

Com a exemple del potencial que té aquest protocol, compararem com es faria una petició com l'anterior, demanar tot el contingut d'una assignatura.

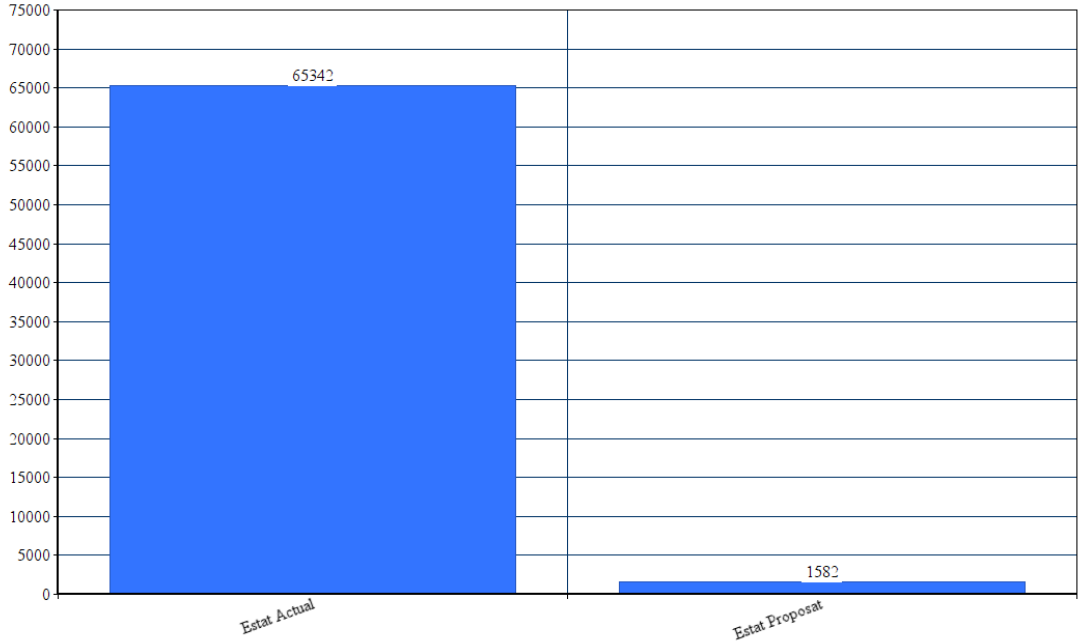
Es podria realitzar una petició com la següent:

```
query{
  sites (name: "ALGEBRA"){
    content{
      title,
      url
    }
  }
}
```

Aquesta petició, retornaria un llistat d'objectes "Site", que únicament tindria el camp content en forma de llista d'objectes "Content", i que cada objecte content, contindria els camps "title" i "url".

Si a més, el resultat és retornes amb un encoding com GZip, el resultat de la petició seria de pocs bytes, es faria amb una sola petició HTTP i l'ORM del servidor podria haver simplificat l'accés a base de dades, resultant en una millora en rendiment tant en el client, servidor i base de dades.

En la següent imatge es veu una comparativa de la petició en la qual s'ha basat l'exemple, comparant els dos estats, l'actual i el proposat. En aquesta comparativa no es veuen reflectits les millores entre el sistema Sakai i la seva base de dades.


75000
70000
65000
60000
55000
50000
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
65342
1582
Estat Actual
Estat Proposat

7. Optimització – Experimentació:

En aquest apartat, estudiarem el treball realitzat relacionat amb el rendiment del sistema Sakai. La intenció d'aquest realitzar un petit experiment per tal de mostrar els avantatges d'una API GraphQL i donar un punt d'entrada a GraphQL dins el sistema Sakai.

Com a referència, podem observar el repositori on viu el codi font:

https://github.com/deinok/sakai/tree/graphql

En primer lloc, s'ha creat un nou mòdul per a Sakai, anomenat "graphiql", aquest mòdul és l'entorn de desenvolupament per a GraphQL amb interfaç gràfica. Permet fàcilment inspeccionar l'esquema de GraphQL i realitzar tot tipus de peticions al sistema de GraphQL.

El mòdul, escolta a l'endpoint "/graphiql" i bàsicament és un servlet HTTP que redirigeix a un fitxer "index.html". Aquest fitxer index.html conte tota la informació necessària per a arrancar l'entorn de programació GraphiQL, ja que està escrit en React i el ús de mecanismes d'introspecció permet que s'autoconfiguri automàticament.

També s'ha creat un segon mòdul anomenat "graphql", aquest mòdul, és la capa de GraphQL, el mòdul conte tota la informació relacionada amb GraphQL.

El mòdul està format pels següents elements:

Un Servlet HTTP que converteix les peticions GraphQL - HTTP en un format que Java sigui capaç d'entendre. Això implica que és capaç d'entendre peticions GET i POST en totes les convencions possibles de GraphQL. En el mètode GET, recupera la informació dels paràmetres de la query, mentre que amb el POST recupera la informació a partir del body de la petició HTTP. Un cop formatat correctament per a Java, una instància d'una petició GraphQL és transmesa al motor de GraphQL.

Un arxiu anomenat "schema.graphqls" que conte l'esquema de GraphQL, aquest conte l'API de GraphQL per a Sakai, però és transparent del llenguatge de programació, això ens permet una fàcil portabilitat de l'API a altres llenguatges.

Un paquet anomenat "models" que modelitza els objectes que s'exposen a Sakai com a classes POJO, això ens permet definir fàcilment el format de què s'exposa sense modificar altres parts de Sakai.

Un paquet anomenat "repositories" que conte la capa de lògica de GraphQL, administra com es recuperen els objectes en ser sol·licitats pel motor de GraphQL, i a més transforma d'objectes interns de Sakai a objectes preparats per ser exposats a través de GraphQL.

Un paquet anomenat "resolvers" que són el punt d'unió entre la lògica de GraphQL i l'especifica de Java. Aquesta és la que utilitza el motor de GraphQL, i permet d'acord amb informació de la petició realitzar millors peticions a la part especifica de Java.

8. Renderització – Investigació:

Estat Actual:

En aquest apartat, estudiarem l'estat actual del mecanisme de renderització del sistema Sakai. La intenció d'aquest apartat és recollir com més informació possible per veure com es comporta el sistema en la seva totalitat. La idea és poder comprendre com respon el sistema davant d'un usuari normal i corrent per tal de fer sentir a l'usuari que el sistema és d'alta qualitat.

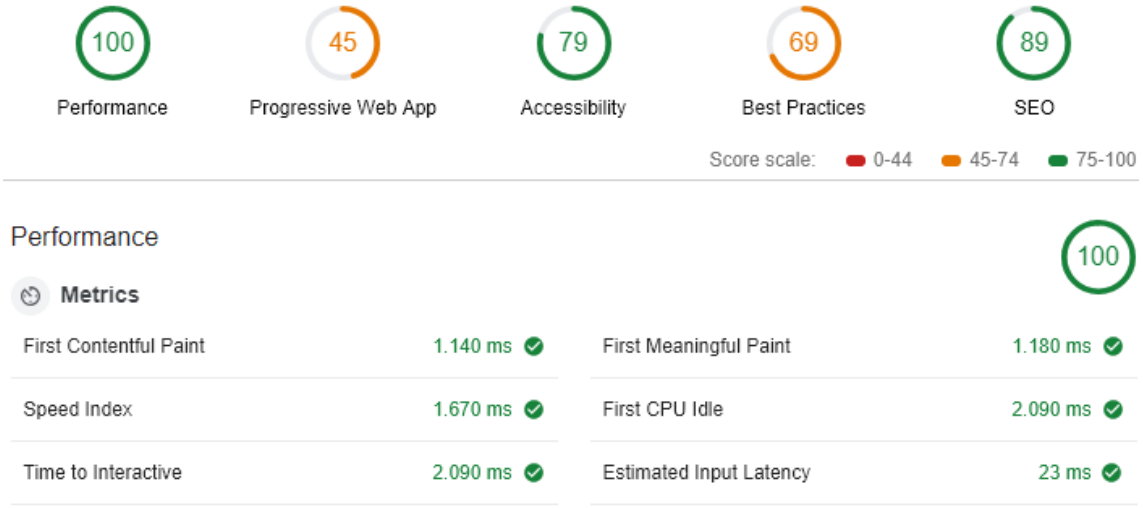
Actualment, la interfaç gràfica del projecte Sakai és un mix entre diferents llenguatges de UI el que fa evident, que no hi ha cap estàndard de cap tipus a l'hora d'escriure nous elements visuals.

Entre els diferents mecanismes que ens hem trobat per crear elements visuals existeixen:

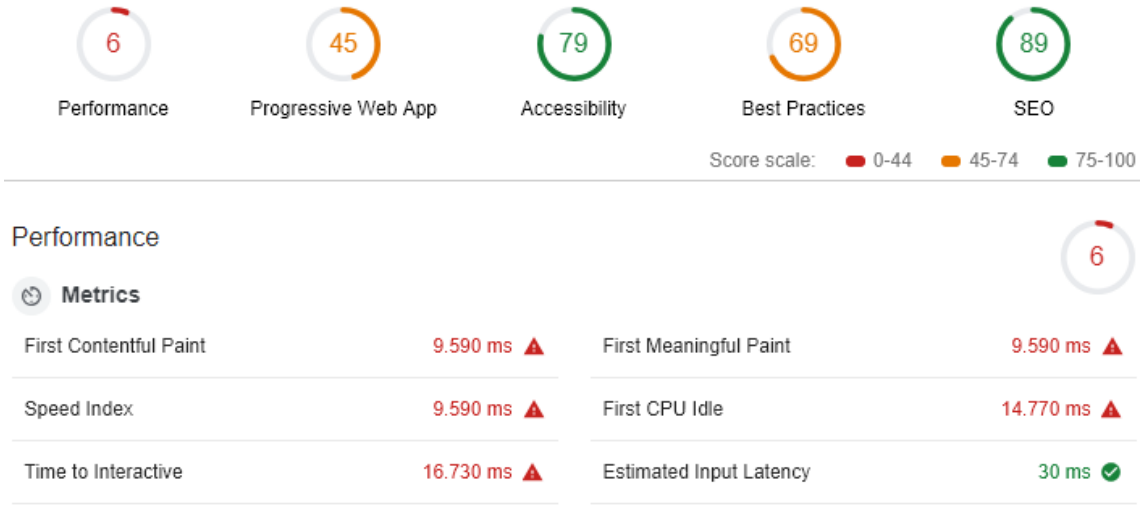
- JSP
- VM
- HTML – Wicket
- HTML
- TLD

Després de fer una auditoria amb un programa anomenat "Lighthouse" que puntua el nivell d'usabilitat de les pàgines web s'ha detectat la següent informació:

En dispositius de sobretaula, la web, es comporta millor de l'esperat, situant-se en menys d'1,5 s abans de poder interactuar amb la pàgina web. Però sí que es detecta que en canviar d'una pàgina a una altra existeix el mateix retard que aproximadament la primera càrrega de la web. Situant-se en aproximadament 1,2 s.



En dispositius mòbils amb velocitats de 3G sí que es nota un retard major del qual seria possible. Però, això és un fet inevitable, ja que la latència d'aquest tipus de xarxes implica un cost en l'experiència d'usuari.



Cal mencionar que el disseny visual ha tingut una millora excel·lent des de la versió 10.X la qual es veia completament antiga. La versió 11.X va suposar una millora descomunal respecte la versió anterior.

A més, la versió de la Universitat de Lleida, ha dissenyat la interfaç gràfica d'una forma encara millor del que la versió bàsica de Sakai implementa com a base.

Des del punt de vista de l'experiència mòbil, ara ja es digna de ser vista des d'un mòbil, adaptant-se en temps real a la mida de la pantalla.

En general, el resum és que des del punt de vista de l'usuari, gairebé no hi ha cap error en l'àmbit d'usabilitat en general i la interfaç és molt intuïtiva.

Estat Proposat:

En aquest apartat s'explica la idea de l'autor sobre com millorar aspectes relacionats amb la interfaç gràfica que es consideren una millora. Juntament amb els motius i explicacions que es considerin necessaris.

Un dels punts on s'han vist més problemes d'interacció amb l'usuari ha sigut quan s'accedia al sistema a través de dispositius mòbils.

La idea principal és reduir els temps de resposta en els dispositius fins al punt que sigui possible que un usuari pugi navega per la web sense connexió a internet.

La majoria de frontends de webs poden ser partits en dues meitats. L'estructura i disseny visual, i les dades que es mostren.

Així doncs la idea general és per a l'estructura i el disseny visual fer ús del cache per a fitxers estàtic que els navegadors utilitzen àmpliament i un Service Worker per a les dades.

De forma simplificada, és aconseguir una Progressive Web App, aconseguir que el projecte Sakai fos una PWA valida estalviaria en gran part la necessitat de crear una aplicació per a diferents dispositius, ja que les PWA poden ser instal·lades com una aplicació web.

Per a l'apartat de la interfaç gràfica, es recomanen dues alternatives.

Web Components, són un conjunt d'API de la plataforma web que permeten crear noves etiquetes HTML personalitzades, reutilitzables i encapsulades per utilitzar-les en pàgines web i aplicacions web. Els components i widgets personalitzats que es basen en els estàndards del component web competeixen amb navegadors moderns i es poden utilitzar amb qualsevol biblioteca o framework de JavaScript que funcioni amb HTML.

Bàsicament, fen ús de les especificacions "Custom Elements", "Shadow DOM", "HTML Imports" i "HTML Templates" és possible la creació de nous element HTML reutilitzables i altament portables. El problema és que són complexes de crear i utilitzar en comparació a frameworks de UI per a HTML

L'altra alternativa, és utilitzar React, un framework per a crear UI amb JavaScript, els seus avantatges són que els elements són reutilitzables, i contenen un conjunt de propietats i un estat, amb la qual cosa, aquests elements d'HTML reaccionen als canvis de forma que actualitzacions a les seves dades no requereixen recarregar tot l'arbre DOM, facilitant el desacoblament de diferents parts de l'interface gràfica.

Per a l'apartat de les dades, es recomana fer ús de Service Workers. Els Service Workers, tenen dos punts forts:

Per un cantó, fer ús d'un Proxy per a recuperar les dades del servidor, permet un implementar un mecanisme de cache transparent a la capa de lògica, fen ús del cache per defecte del navegador. Això, permet que gairebé no es facin peticions HTTP, i que la informació requerida ja estigui descarregada inclús abans de sol·licitar-la, millorant el temps de reacció de la web, millorant la satisfacció de l'usuari, i millorant el nivell de sobrecàrrega del sistema Sakai.

Per l'altre canto, el codi executat en un Service Worker s'executa en un thread diferent del de UI, permeten que el thread de UI se centri únicament a realitzar tasques relacionades amb la UI mentre paral·lelament, el thread del Service Worker realitza les tasques de I/O i intensives de CPU i notifica a la UI els canvis que siguin necessaris.

La unió de les dues parts mencionades, són els elements bàsics per a la creació d'una PWA que de forma general milloraria la sensació de l'usuari respecte el sistema i simplificaria el desenvolupament de la part de UI de Sakai.

9. Altres Millores – Investigació:

En aquest apartat, estudiarem l'estat actual d'altres aspectes del sistema Sakai que no entren en cap dels altres apartats. La intenció d'aquest apartat és recollir com més informació possible per veure com es comporta el sistema en la seva totalitat.

JDK:

El sistema Sakai és únicament capaç de funcionar amb la versió de Java 1.8. Tot i que existeixen 3 versions superiors, i la última versió és LTS (Long Term Support), Sakai és incapaç de funcionar amb aquestes noves versions a causa de problemes de compatibilitat amb JDK9. A continuació explicarem els problemes per als quals no es pot actualitzar a nous JDKs.

Amb la introducció del JDK9 es va introduir un nou sistema d'empaquetament, anomenat "Project JigSaw", aquest projecte es va idear aproximadament per a la versió JDK7 i ha sigut el causant de molts retards al JDK7, JDK8 i finalment JDK9.

Això, i l'eliminació de moltíssimes APIs i mecanismes ideats per a ser interns però que realment eren exposats a través de l'API i ABI han causat que la transició de JDK8 ->;; JDK9 sigui un problema difícil de solucionar.

Com a norma general, noves versions de software són millors en tots els aspectes que anteriors versions. Per la qual cosa té molt sentit intentar preparar el sistema per a noves versions de Java.

Generalment, el problema més gran per fer la migració, és el ús de llibreries desactualitzades i incompatibles amb versions superiors a JDK8. Les més importants i difícils d'actualitzar són Hibernate i Spring, ja que Sakai fa molt ús d'aquests frameworks.

Llibreries:

S'ha detectat que el projecte Sakai fa un ús generalitzat de llibreries completament desactualitzades, i també un ús excessiu de llibreries que solucionen problemes molt petits. Això és un problema ja per un cantó es poden provocar incompatibilitats, i per altre Sakai requereix de llibreries tan petites que algunes ja no són desenvolupades en cap sentit.

HTTP/2.0:

Una de les millores que es podria fer és habilitar l'ús d'HTTP/2.0.

L'ús d'aquest protocol milloraria el rendiment general del sistema, ja que algunes de les millores que ofereix son temps de latència menors gràcies a coses com la multiplexació de peticions a través d'un mateix socket TCP i mecanismes de PUSH des del servidor.

Docker:

Sakai, per si mateix no ofereix una imatge de Docker per tal de simplificar el procés de Deployment de les instàncies.

Amb suport per a Docker, permetria als implementadors de Sakai poder fer ús de l'anomenat Docker Swarm per tal d'administrar múltiples instàncies de Sakai de forma automatitzada.

Això permetria uns menors costos de Deployment i de manteniment de servidors, a més de permetre optimitzar els recursos computacionals depenen de la càrrega necessària en temps real.

Temps Compilació – Deployment - Startup:

S'ha detectat que els temps de compilació, deploy i Startup són exageradament alts arribant als 20 minuts simplement per aplicar un canvi en el codi font.

Aquest fet alenteix la velocitat de desenvolupament d'una forma exagerada. Fins al punt que es converteix en un problema real que tira enrere a nous desenvolupadors.

Codi font gran i complex:

S'ha detectat que el codi font ocupa una quantitat estranya d'emmagatzematge, aproximadament 150MB sense compilar i aproximadament 1GB després de compilar.

Això és una quantitat estranyament gran per les funcions que està realitzant Sakai.

A part, s'ha detectat que el codi font segueix certs patrons que causen una verbositat innecessària.

Continuous Integration:

S'ha detectat que el mecanisme de Continuous Integration de Sakai ofusca certes accions de forma automàtica. Això pot provocar problemes d'environament.

Issues i Pull Requests:

L'organització de Sakai obliga que cada Pull Request creat estigui linkejat a una issue a un sistema JIRA extern. Això dificulta el ràpid desenvolupament extremadament.

Code Style:

El codi font de Sakai és un complet mix de múltiples estils de codi. Dificultant la comprensió del codi.

Anàlisis de codi:

Un anàlisi del codi amb eines com per exemple l'analitzador de Java d'IntelliJ IDEA ha revelat més de 60000 warnings dels quals aproximadament uns 20000 són warnings verdaders que indiquen micró optimitzacions, o canvis que simplificarien el codi fent-lo més simple de llegir.

10. Altres Millores – Experimentació:

En aquest apartat, explicarem tots els canvis que s'han realitzat al projecte Sakai.

Aquests canvis estan ja inclosos dins el projecte Sakai en el repositori principal del projecte. En la seva major part, són canvis bàsics de manteniment per tal de tindre una bona base de treball.

Com a referència, tots aquests canvis, poden ser observats al repositori de codi font principal amb usuari @Deinok:

https://github.com/sakaiproject/sakai

Juntament amb les issues creades a la instància de JIRA amb usuari @Deinok:

https://jira.sakaiproject.org/projects/SAK/issues

JDK:

S'han realitzat canvis per tal que el projecte compili per a JDK9 però malauradament encara no passa tots els tests. Actualment, aproximadament un 25% dels mòduls ja passen els tests amb el JDK9.

Per als JDK 10 i 11, aproximadament un 20% dels mòduls ja compilen satisfactòriament.

Llibreries:

S'han actualitzat una gran quantitat de llibreries. Totes les llibreries d'APIs com "javax.*" han sigut actualitzades de tot el projecte, permeten l'ús de noves APIs de última generació que milloren el rendiment general de tot el projecte. També s'han actualitzat una gran quantitat de llibreries de la família "org.apache.commons.*".

Una de les altres millores fetes, és que s'ha simplifica unificat les versions de moltes llibreries, simplificant el procés de gestió d'aquestes i millorant la interoperabilitat dels mòduls i els seus temps de compilació.

HTTP/2.0:

Aquesta millora forma part de les millores a nous JDK, ja que el servidor d'aplicacions Tomcat només és capaç de funcionar amb HTTP/2.0 amb versions superiors a Tomcat 9.0 i JDK 9.

Docker:

S'ha creat un fitxer de Docker genèric que és capaç de compilar el Sakai i de servir-lo en la mateixa imatge mentre es fa ús de la tècnica multi-stage per tal que la imatge tan sols extengui de la imatge de Tomcat i únicament tingui el deploy realitzat, minificant el temps de pull i simplificant els costos d'extensió d'aquesta imatge.

Com a referència, en aquest repositori viu la imatge de Docker creada:

https://github.com/Sakzilla/sakai-docker

Temps Compilació – Deployment - Startup:

S'ha modificat alguns plugins de Maven per tal que el temps de compilació resulti en aproximadament 18 minuts en comparació als 20 minuts abans de la modificació.

Continuous Integration:

S'ha realitzat un port al CI CircleCI que permet executar els tests en un entorn basat en Docker.

Aquest canvi, millora el temps de testing, indica millor quin és l'entorn de testing, i ens permet testejar per a múltiples JDKs.

Code Style:

S'ha realitzat una petició formal per a que el projecte utilitzi un fitxer EditorConfig, que permetria reformatarà tot el projecte a un única guia d'estil i eliminaria el problema de múltiples estils de codificació.

Anàlisis de codi:

S'han anat arreglant alguns del warnings mòdul per mòdul de forma progressiva, ja que un sol Pull Request que modifiques la meitat del projecte seria denegat de forma inevitable.

6. Conclusions:

Després de realitzar aquest treball s'ha arribat a una conclusió molt important.

Tots els desenvolupadors de Sakai volen noves features, però ningú està disposat a mantenir-les.

Tothom realitza el treball que li interessa, sense arribar a una convenció. Un exemple és el mix de frameworks, alguns fan anar Spring, altres Spring Boot, altres simplement JavaEE, etc.

Els anomenats mòduls són partits per funcionalitats i no per capes lògiques, provocant que per a buscar cert tros de codi has de realitzar una cerca que tarda molt temps. Tot aquest esquema tindria cert sentit si els mòduls fossin plugins, però en el fons són parts centrals del sistema Sakai.

Els mòduls no segueixen la convenció de projectes Maven. És més, si crees un mòdul que segueixi la convenció, aquest no funciona, no se sap perquè.

Em sorprèn enormement que en una CPU amb 16 threads a 4GHz cadascun tardi 20 minuts a compilar, això relentitza enormement el temps per fer cada canvi més mínim.

El procés per aplicar un canvi és extremadament lent, fins a punt que alguns projectes basats a aplicar arxius .patch a través d'enviar-se correus són més ràpids. Em va sorprendre enormement, el dia que vaig arreglar un error que causava que el master no compiles perquè van afegir un mòdul creat el 2005 que referenciava un arxiu fora del projecte, a més van demanar-me que creés una issue a un sistema de gestió de projectes que a part que no aporta res, simplement complicar el fet d'entendre els canvis aplicats, els hi vaig comentar que no tenia sentit crea issues d'aquest tipus quan el canvi era eliminar tres simples caràcters. Realment, tot aquest procés va tardar 4 dies.

També és interessant el tema de la llibreria Lombok, tenen un plugin especial per tal d'estalviar-se una simple línia de codi a cada classe que necessitin fer algun tipus de loggin, l'ús d'aquest mecanisme causa que tècnicament, els arxius de codi font siguin detectats per IDEs com a què existeixen errors.

Una de les coses més estranyes que he vist, són els increïbles 1GB com a resultat de compilar el projecte. No em dona la sensació que existeixin suficients funcionalitats per a generar tal quantitat de codi optimitzat.

En definitiva, sincerament crec que hi hauria un encarregat encara que sigui temporalment d'arreglar les bases del sistema Sakai i realitzar un mínim de manteniment, ja que m'ha fet la impressió que he passat més temps arreglant coses que considero bàsiques, que fen les tasques que havia pensat en un principi.

7. Treball Futur:

Queden bastants tasques pendents, ja que tal com s'ha comentat, s'ha gastat moltíssim temps en fer tasques de manteniment per tal d'evitar futurs problemes en el projecte.

Per ordre de prioritat es llisten les següents tasques de forma simplificada:

1. Realitzar una neteja de les APIs de les llibreries depreciades.
2. Treure el ús de Lombok.
3. Arreglar els errors automàtics que es detecten al fer un anàlisis del codi.
4. Portar Hibernate a última versió.
5. Portar Spring a última versió.
6. Portar Spring a Spring Boot.
7. Habilitar JDK9.
8. Habilitar JDK10.
9. Habilitar JDK11.
10. Deshabilitar JDK8, 9, 10.
11. Portar tota l'API Direct a GraphQL.
12. Implementar OAuth2.0.
13. Deshabilitar on sigui possible els altres mecanismes de sign-in.
14. Unificar la interfaç gràfica amb un sol sistema.
15. Simplificar el codi en la mesura de el possible.
16. Eliminar llibreries relativament innecessàries.
17. Etc.

8. Web grafia:

- https://github.com/sakaiproject/sakai
- https://jira.sakaiproject.org/
- https://www.sakaiproject.org/
- https://confluence.sakaiproject.org/display/BOOT/Install+Tomcat+8
- http://nightly2.sakaiproject.org/
- https://es.wikipedia.org/
- https://visualstudio.microsoft.com/es/xamarin/
- https://cordova.apache.org/
- https://www.android.com/
- https://maven.apache.org/
- https://www.oracle.com/technetwork/java/javaee/overview/index.html
- https://spring.io/
- http://hibernate.org/
- https://www.mysql.com/products/workbench/
- http://tomcat.apache.org/
- https://oauth.net/core/1.0/
- https://www.apereo.org/projects/cas
- https://www.shibboleth.net/
- https://developers.onelogin.com/saml
- https://web.mit.edu/kerberos/
- http://web.mit.edu/rhel-doc/4/RH-DOCS/rhel-rg-es-4/ch-ldap.html
- https://oauth.net/
- https://jwt.io/
- https://redis.io/
- https://www.cvedetails.com/
- https://es.wikipedia.org/wiki/Copy-on-write
- https://dirtycow.ninja/
- http://boards.4chan.org/g/
- https://cv.udl.cat/portal
- https://www.gzip.org/
- https://github.com/graphql-dotnet/graphql-client
- https://graphql.org/
- http://facebook.github.io/graphql/June2018/
- https://github.com/graphql-java
- https://www.apollographql.com/
- https://msdn.microsoft.com/es-es/library/system.linq.iqueryable(v=vs.110).aspx
- https://chrome.google.com/webstore/detail/lighthouse/blipmdconlkpinefehnmjammmfjpmpbjk?hl=es
- https://www.oracle.com/technetwork/java/javaee/jsp/index.html
- https://developers.google.com/web/progressive-web-apps/
- https://www.webcomponents.org/
- https://reactjs.org/

- https://angular.io/
- https://developer.mozilla.org/es/docs/Web/API/Service_Worker_API
- https://mvnrepository.com/
- https://developers.google.com/web/fundamentals/performance/http2/?hl=es
- https://www.docker.com/
- https://hub.docker.com/
- https://circleci.com/
- https://travis-ci.org/
- https://editorconfig.org/