

Utredning om möjligheter för aktivering av tilläggsfunktioner för bokningssystem

Jan-Anders Björnvik

Examensarbete för ingenjör (YH)-examen

El - och automationsteknik

Vasa 2022

EXAMENSARBETE

| | |
|---------------------|----------------------------------|
| Författare: | Jan-Anders Björnvik |
| Utbildning och ort: | El - och automationsteknik, Vasa |
| Inriktning: | Informationsteknik |
| Handledare: | Jan Berglund |

Titel: Utredning om möjligheter för aktivering av tilläggsfunktioner för bokningssystem

Datum: 9.5.2022

Sidantal: 28

Abstrakt

Detta examensarbete gjordes på begäran av Hogia Ferry Systems, som utvecklar mjukvara för att hantera data om passagerare och frakt med färjetrafik. Syftet med detta examensarbete var att undersöka möjligheterna att förverkliga en portal där kunderna själva skulle kunna testa och köpa enklare tilläggsfunktioner som används i deras installation av bokningssystemet BOOKIT.

Lösningen som undersöktes var om man kan bygga vidare på en redan existerande kundportalplattform där man sedan listar enklare tilläggsfunktioner som kan aktiveras utan extra hjälp av en programmerare. Listan kan sedan uppdateras med lämpliga tilläggsfunktioner som visas på portalen. För att göra detta kan tilläggsfunktionerna läsas upp ur en XML-konfigurationsfil som kommer med BOOKIT-installationen och för att hitta rätt data om tilläggsfunktionerna i konfigurationsfilen använder man sig av LINQ to XML i programmeringsspråket C#. Datan skickas sedan till webbapplikationen där den visas ut i kundportalen. Eftersom de tilläggsfunktioner som kommer att listas på kundportalerna skall vara sådana funktioner som är lämpliga för en aktivering på portalen och som kunden kan använda, måste funktionerna vara kompatibla med kundens installation av BOOKIT samt kunna aktiveras genom att ändra parametervärdet från falsk till sann.

När det blir dags att implementera detta portaltillägg blir det antagligen ändrat så att data om tilläggsfunktionerna blir exporterade till en databas, som man sedan läser upp data ifrån, för att göra det framtidssäkert.

Språk: svenska

Nyckelord: Hogia, BOOKIT, tilläggsfunktioner, kundportal

BACHELOR'S THESIS

| | |
|-------------------|---------------------------------------|
| Author: | Jan-Anders Björnvik |
| Degree Programme: | Electrical Engineering and Automation |
| Specialisation: | Information Technology |
| Supervisor(s): | Jan Berglund |

Title: Investigation of possibilities for activating add-on functions for booking systems

Date: May 9, 2022

Number of pages: 28

Abstract

This thesis was done on behalf of Hogia Ferry Systems, which develops software for managing passenger and cargo data for ferry traffic. The aim of this thesis was to evaluate the possibilities of actualising a portal where customers themselves could test and buy simpler add-on functions that are used in their installation of the reservation system BOOKIT.

The evaluated solution was if it is possible to build on an already existing customer portal platform where simpler add-on functions are listed and can be activated without extra help from a programmer. The list containing the add-on functions can then be updated with suitable add-on functions that are displayed in the portal. To do this the add-on functions are read from an XML configuration file that comes with the BOOKIT installation. LINQ to XML in the programming language C# can be used to find the correct data in the configuration file and the data will then be sent to the web application where it is displayed in the customer portal. The functions listed in the portal must be compatible with the customer's installation of BOOKIT and be able to be activated simply by changing the parameter value from false to true.

When the time comes to implement this portal add-on, it will probably be made so that data about the add-on functions are exported to a database, from which the data can be read, to make it more future-proof.

Language: Swedish

Key words: Hogia, BOOKIT, add-on functions, customer portal

Förkortningar och definitioner

| | |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| BOOKIT | BOOKIT är ett bokningssystem som är inriktat på färjeresor och färjetransporter. Systemet hanterar hela bokningsprocessen från att fånga uppgifter om passagerare, fordon och gods till uppföljningsaktiviteter som fakturering och rapportering. |
| IIS | Internet Information Services är en serverprogramvara från Microsoft för internetbaserade tjänster. |
| XML | Extensible Markup Language, ett universellt och utbyggbart märkspråk och en förenklad efterträdare till SGML. |
| XSLT | Extensible Stylesheet Language Transformations är en standard för bearbetning av XML-data. |
| XSD | XML Schema Definition är en World Wide Web Consortium (W3C) rekommendation som anger hur man formellt beskriver elementen i ett XML-dokument. Denna beskrivning kan användas för att verifiera att varje innehållsobjekt i ett dokument följer beskrivningen av element där innehållet ska placeras. |
| LINQ to XML | LINQ to XML är ett LINQ-aktiverat, in-memory XML programmeringsgränssnitt som gör att man kan arbeta med XML inom .Net-programmeringsspråken. |
| DOM | Document Object Model är ett programmerings API för HTML- och XML-dokument. DOM definierar den logiska strukturen för dokument och hur ett dokument öppnas och manipuleras. |
| DTD | Document Type Definition beskriver ett sätt att beskriva strukturen på ett XML- eller SGML-dokument. |
| DoS | Denial of Service-attack är en attack som är avsedd att stänga av en maskin eller ett nätverk vilket gör det otillgängligt för dess användare. DoS-attacker uppnår detta genom att översvämna målet med trafik eller genom att skicka information som utlöser en krasch av systemet. |

Innehållsförteckning

| | | |
|-------|----------------------------------------|----|
| 1 | Inledning..... | 1 |
| 2 | Bakgrund | 1 |
| 2.1 | Uppdraget..... | 1 |
| 2.2 | Uppdragsgivare..... | 3 |
| 2.2.1 | Hogia Ferry Systems..... | 3 |
| 2.2.2 | Hogia..... | 4 |
| 3 | Teknik..... | 5 |
| 3.1 | .Net Framework | 5 |
| 3.2 | C# .Net | 7 |
| 3.3 | LINQ to XML | 9 |
| 3.3.1 | Säkerhetsaspekter med LINQ to XML..... | 10 |
| 4 | Utförande..... | 14 |
| 4.1 | Allmänna strukturen..... | 14 |
| 4.2 | XML-hantering..... | 18 |
| 4.3 | Implementation..... | 20 |
| 5 | Resultat och diskussion | 26 |
| | Referenser | 28 |

1 Inledning

Den utredning som är grunden för detta arbete har gjorts på uppdrag av Hogia Ferry Systems (HFS). Syftet med detta examensarbete var att undersöka möjligheterna att förverkliga en portal där kunderna själva skulle kunna testa och köpa enklare tilläggfunktioner som används i deras installation av bokningssystemet BOOKIT.

Det som i huvudsak kommer att presenteras i detta arbete är en teori där de tekniker som varit mest intressanta för denna utredning tas upp och ett utförande där lösningen på problemet presenteras.

2 Bakgrund

I detta kapitel kommer bakgrunden till uppdraget samt uppdragsgivaren att presenteras mera ingående för att ge läsaren en bakgrund till varför detta arbete gjordes. Det system som tilläggfunktionerna skall aktiveras för är bokningssystemet BOOKIT som HFS utvecklar.

2.1 Uppdraget

Bakgrunden till uppdraget var att undersöka hur man skulle kunna förverkliga en portal där färjeföretagen själva ska kunna testa enklare tilläggfunktioner, eng. features, till BOOKIT. Idag är processen att aktivera en tilläggfunktion tidskrävande och kräver ett samarbete

mellan ett flertal personer även vid små förändringar och enklare funktionalitet. Uppdraget i denna avhandling är att undersöka möjligheten att utveckla en portal där färjeföretagen själva ska kunna testa dessa enklare tilläggsfunktioner. Därmed kan man eliminera en utvecklare som mellanhand och effektivisera processen vid aktiveringen av tilläggsfunktioner.

Processen för aktivering av tilläggsfunktioner ser ut som följande. Slutkunden meddelar HFS att de skulle vara intresserad av att testa en ny tilläggsfunktion. Ett jobbärendes sätts upp för detta i ett kundhanteringssystem och ärendet tilldelas en utvecklare som har tid att utföra ärendet. Utvecklaren kontrollerar sedan vilken tilläggsfunktion som skall aktiveras. Beroende på hur komplext tillägget är ändras antingen en sann/falsk, eng. true/false, parameter i en XML-fil eller så tillsätts den kodlogik som fattas, ifall det är ett tillägg som kräver ändringar i användargränssnittet.

För att effektivisera detta så skulle man i stället bygga en portal där man sätter upp de tilläggsfunktioner som enkelt kan aktiveras av kunden. I portalen kan kunden sedan se vilka tilläggsfunktioner som är tillgängliga för att kunna testa eller köpa dem direkt.

I en första iteration av portalen är det tänkt att endast de tilläggsfunktioner som har en sann/falsk parameter ska vara tillgängliga för aktivering. Målet längre fram är dock att de flesta tilläggsfunktioner ska vara tillgängliga i portalen.

2.2 Uppdragsgivare

Här presenteras HFS, som är uppdragsgivare för detta arbete, Hogia-gruppen som HFS hör till samt den programvara som HFS utvecklar och säljer.

2.2.1 Hogia Ferry Systems

Uppdragsgivare för detta arbete var HFS i Vasa. HFS hör till en större koncern vid namn Hogia-gruppen som har sitt moderbolag i Stenungssund i Sverige.

Hogia Ferry Systems, som är ett av bolagen i Hogia-gruppen, grundades 1981 då med namnet Consy, men namnet ändrades till HFS när företaget gick med i Hogia-gruppen. Idag har HFS cirka 30 anställda med sin verksamhetspunkt i Vasa, Finland och är verksamma inom färjetrafiken. HFS utvecklar och säljer bokningssystemet BOOKIT sedan 1987.

BOOKIT som system består av många olika applikationer för att hantera färjebokningar men all information i BOOKIT lagras i en gemensam databas. Informationen är sedan tillgänglig för alla applikationer att använda sig av. (Hogia BOOKIT - standard applications, u.d.).

De applikationer som ingår i BOOKIT som system är följande:

- Booking
- Check-in
- Account Management
- Finance
- Quicksales
- Reporting

2.2.2 Hogia

År 1980 grundades ett programvaruföretag vid namn Hogia av Bert-Inge Hogsved i Sverige. Från och med år 2020 består Hogia-gruppen av 30 olika företag utspridda över hela Norden och i Storbritannien med sammanlagt över 700 anställda som alla sysslar med någon form av mjukvara. De olika systemen, som alla är utvecklade av något företag inom Hogia-gruppen, finns utspridda inom tre områden: finans- och affärssystem, HR-system och transportsystem. Företagen inom Hogia-gruppen ansvarar alla för utveckling, kundservice och försäljning inom sitt eget verksamhetsområde. (Vår historia, u.d.).

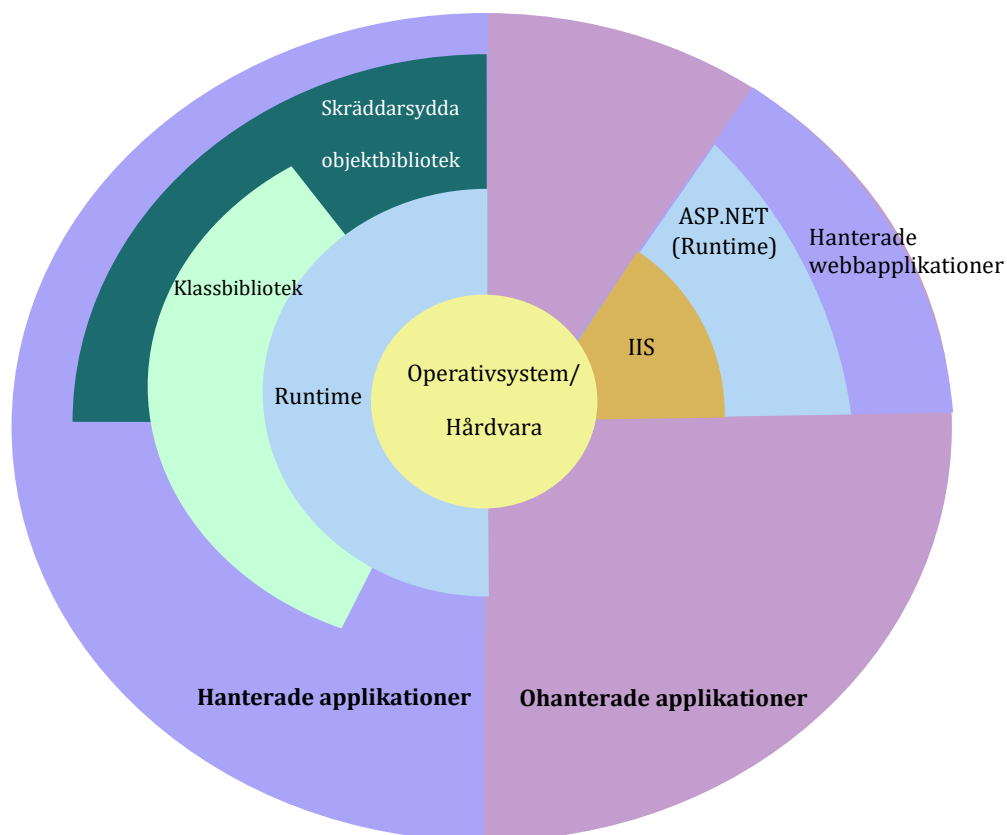
3 Teknik

I detta kapitel presenteras .Net Framework, C# och LINQ to XML som är de tekniker och programmeringsspråk som används i detta arbete. .Net Framework och C# kommer att behandlas på ett ytligt plan eftersom dessa tekniker valdes på grund av att den redan existerande källkoden för BOOKIT är skriven i C#. För LINQ to XML kommer allmän information och säkerhetsaspekter att presenteras.

3.1 .Net Framework

.Net Framework har stöd för att bygga och köra Windows applikationer och webbtjänster och är utformat för att uppfylla bland annat följande mål. Ett av dessa mål är att tillhandahålla en programmeringsmiljö som är konsekvent och objektorienterad oberoende om koden körs och blir lagrad lokalt eller körs lokalt men är webbdistribuerad eller om koden blir exekverad på distans. .Net Framework består av Common Language Runtime (CLR), har hand om att hantera exekveringen av olika .Net applikationer oavsett vilket .Net programmeringsspråk applikationerna har blivit skrivna i, och klassbiblioteket .Net Framework som tillhandahåller ett bibliotek med testad och återanvändbar kod som en utvecklare kan anropa från sina egna applikationer.

CLR hanterar koden vid körningstid och tillhandahåller flera kärntjänster som bland annat minneshantering, trådhantering och fjärrstyrning. Samtidigt upprätthåller CLR en strikt typsäkerhet och andra former av kodnoggrannhet som bidrar till en säker och robust kod. Konceptet med kodhantering är en grundläggande princip för CLR och kod som är inriktad på CLR brukar kallas för hanterad kod och kod som inte är det brukar kallas för ohanterad kod. Eftersom CLR har inbyggda funktioner för bland annat minneshantering, kodsäkerhetsverifiering och kompilering blir dessa funktioner implementerade direkt av kompilatorn utan att göra något mellanliggande anrop till ett bibliotek i den hanterade koden som körs i CLR.



Figur 1. Schematisk bild över huvudfunktionerna i .Net Framework.

I Figur 1 ser man en schematisk framställning av huvudfunktionerna som .Net Framework består av indelade i sektioner. Dessa sektioner kan delas in i hanterade applikationer och ohanterade applikationer och mitt emellan dessa finns operativsystemet och hårdvaran. Till de hanterade applikationerna hör skräddarsydda objektbibliotek, klassbibliotek och körningstid. Till de ohanterade applikationerna hör olika hanterade webbapplikationer, ASP.NET (runtime) och IIS.

Klassbiblioteket .Net Framework är en omfattande, objektorienterad samling av återanvändbara typer som används vid utveckling av traditionella

kommandoradsapplikationer eller applikationer med ett grafiskt användargränssnitt till applikationer baserade på ASP.Net som Web Forms eller XML-webbtjänster. .Net Framework kan vara värd för ohanterade komponenter som laddar in CLR i sina processer och initierar exekvering av hanterad kod och en mjukvarumiljö som använder sig av både hanterade och ohanterade funktioner skapas. (Warren, o.a., 2021).

3.2 C# .Net

C# är ett objektorienterat programmeringsspråk som är skapat av Microsoft och som körs på .Net Framework. Första versionen av språket släpptes år 2002 och den senaste versionen (version 10.0) släpptes 2021. C# används bland annat för att skapa mobil- och skrivbordsapplikationer. (W3Schools, u.d.).

Det finns flera olika funktioner i C# som finns till för att hjälpa att skapa stabila och hållbara applikationer som till exempel stöd för LINQ syntax, som skapar ett gemensamt mönster för att arbeta med data från vilken källa som helst och stöd för asynkrona funktioner. Efter att programmeringsspråket blev utgivet har det implementerats nya funktioner som stöder mera moderna metoder för design och skapande av mjukvara. För programmeringsspråket C# finns dessutom stöd för både användardefinierade referenstyper och värdetyper. Det tillåter också dynamisk allokering av objekt och in-line lagring av lättare strukturer. Stöd för generiska metoder och typer, vilket resulterar i en ökad typsäkerhet och bättre prestanda, finns också.

För att göra det möjligt att definiera anpassade beteenden för klientkoden vid implementeringar av olika samlingsklasser, tillhandahåller programmeringsspråket C# iteratorer. För att kunna säkerställa att olika program och bibliotek kan utvecklas på ett kompatibelt sätt över tiden har C# betoning på versionshantering. De aspekter av programmeringsspråkets design som påverkades direkt av versionsöverväganden är bland annat de separata virtual- och override-modifierarna och reglerna för

metodöverbelastningslösning samt stöd för explicita deklarationer av gränssnittsmedlemmar. (Wagner, o.a., 2021).

I Kodexempel 1 ser man ett exempel på hur syntaxen i C# ser ut. Koden är ett exempel på hur man kan skapa funktioner som antingen adderar, subtraherar, multiplicerar eller dividera olika tal beroende på vilket räknesätt som matas in i funktionen Calculate.

Kodexempel 1. Exempel på hur syntax ser ut i C#.

```
public decimal Calculate(decimal number1, decimal number2, char operation)
{
    switch (operation)
    {
        case '+':
            return Add(number1, number2);
        case '-':
            return Subtract(number1, number2);
        case '*':
            return Multiply(number1, number2);
        case '/':
            return Divide(number1, number2);
        default:
            return 0;
    }
}

private decimal Add(decimal term1, decimal term2)
{
    decimal result = 0;
    result = term1 + term2;
    return result;
}

private decimal Subtract(decimal term1, decimal term2)
{
    decimal result = 0;
    result = term1 - term2;
    return result;
}

private decimal Multiply(decimal factor1, decimal factor2)
{
    decimal result = 0;
    result = factor1 * factor2;
    return result;
}

private decimal Divide(decimal numerator, decimal denominator)
{
    decimal result = 0;
    result = numerator / denominator;
    return result;
}
```

3.3 LINQ to XML

Denna namnrymd gör det möjligt att använda LINQ to XML-programmeringsklasserna.

LINQ to XML tillhandahåller ett in-memory XML-programmeringsgränssnitt som utnyttjar .NET Language-Integrated Query Framework och är jämförbar med ett omarbetat och uppdaterat DOM XML-programmeringsgränssnitt. LINQ to XML är likt DOM på det sättet att det läser in ett XML-dokumentet i datorns minne, där man kan skriva frågor (eng. queries) och ändringar till dokumentet och efteråt spara dokumentet som en fil eller serialisera det och skicka det över internet. Det som dock skiljer LINQ to XML från DOM är det att LINQ to XML tillhandahåller en ny objektmodell som är mera lättviktig och enklare att jobba med samt att man får ta del av fördelarna med programmeringsspråken C# och Visual Basic.

Den största fördelen med LINQ to XML är dess integration med Language Integrated Query (LINQ) som ger en utvecklare möjligheten att skriva frågor till XML-dokumentet som finns i minnet och hämta ut samlingar av element och attribut. Eftersom LINQ på samma gång finns integrerat i programmeringsspråken C# och Visual Basic får man också bland annat bättre kontroll av kompileringstiden och förbättrat felsökningsstöd.

En annan fördel är möjligheten att kunna använda resultatet från en fråga som parameter till XElement och XAttribute objektkonstruktörerna vilket ger ett kraftfullare tillvägagångssätt när man ska skapa ett XML-träd. Detta tillvägagångssätt gör det enklare för en utvecklare att omvandla ett XML-träd från en form till en annan. (LINQ to XML overview, 2021).

Kodexempel 2. Exempel på hur LINQ to XML syntax ser ut.

```
XElement root = XElement.Load(Constants.xmlFile);
IEnumerable<XElement> address = from el in root.Elements("Adress")
                                where el.Attribute("Type").Value == "Fakturering"
                                select el;
```

I Kodexempel 2 ovan ser man ett exempel på hur LINQ to XML ser ut vid användning med programmeringsspråket C#. Kodexemplet visar hur man kan hitta ett element som heter Adress och som har ett attribut Type med värdet Fakturering i en XML-fil. Filsökvägen till XML-filen är sparad som en konstant med namnet xmlFile.

3.3.1 Säkerhetsaspekter med LINQ to XML

Eftersom de flesta scenarier där man behandlar data i XML-format kommer det från pålitliga dokument i stället för opålitliga XML-dokument som laddas upp till en server. Därför är LINQ to XML mer optimerat för användning vid dessa scenarier och designat för mer programmeringsbekvämlighet än för applikationer på serversidan med stränga säkerhetskrav. Ifall scenarier uppstår där man behöver behandla data från opålitliga källor, rekommenderar Microsoft att man använder sig av en instans av XmlReader klassen som har blivit konfigurerad att filtrera ut kända XML Denial of Service-attacker (DoS-attacker). Denna källa kommer att gälla för resten av detta kapitel (LINQ to XML security, 2021).

Nedan följer en lista på 10 punkter utan inbördesordning av ytterligare aspekter att ta i beaktande vad gäller säkerheten med LINQ to XML:

1. Exponera inte felmeddelanden för opålitliga anropsprocedurer

Eftersom ett felmeddelande kan avslöja data om till exempel vilken sorts data som transformeras, olika filnamn eller detaljer om implementeringen. Därför ska

felmeddelanden inte exponeras för anropsprocedurer som är opålitliga och så bör man fånga upp alla fel och rapportera dessa med anpassade felmeddelanden.

2. Anropa inte `CodeAccessPermissions.Assert` i en händelsehanterare

En assembly kan ha mindre eller större behörigheter, en assembly som har större behörighet har större kontroll över datorn och dess miljöer. Om kod i en assembly med större behörigheter anropar `CodeAccessPermissions.Assert` i en händelsehanterare och sedan skickar XML-trädet till en skadlig assembly som har begränsade behörigheter, kan den skadliga assemblyn orsaka att en händelse tas upp. Eftersom händelsen kör kod som finns i assemblyn med större behörigheter skulle den skadliga assemblyn sedan fungera med förhöjda privilegier.

3. Acceptera inte Document Type Definition (DTD) från otillförlitliga källor

Eftersom enheter i DTD i sig inte är säkra är det möjligt för ett skadligt XML-dokument som innehåller en DTD att få parsern att använda allt minne och CPU-tid. Vilket orsakar en DoS - attack. Därför är DTD -bearbetning inaktiverad som standard i LINQ till XML då man inte bör acceptera DTD från otillförlitliga källor.

4. Undvik överdriven buffertallokering

När man gör en utveckling bör man vara medveten om att extremt stora datakällor kan leda till uttömning av resurser och attacker mot förnekande av tjänster. Om en skadlig användare skickar in eller laddar upp ett mycket stort XML-dokument kan det orsaka att LINQ to XML förbrukar alltför stora systemresurser. Detta kan utgöra en DoS - attack. För att förhindra detta kan man ställa in egenskapen `XmlReaderSettings.MaxCharactersInDocument` och skapa en läsare som sedan är begränsad i storleken på dokumentet som den kan ladda. Läsaren använder man sedan för att skapa XML-trädet.

5. Undvik överdriven expansion av entiteter

Vid användning av DTD är en känd DoS-attack när ett dokument orsakar överdriven expanderings av entiteter. För att kunna förhindra detta ska man ställa in egenskapen `XmlReaderSettings.MaxCharactersFromEntities` och skapa en läsare som sedan är begränsad i antalet tecken som är resultatet av entitetsexpansionen. Den skapade läsaren används sedan för att skapa XML-trädet.

6. Begränsa djupet i XML -hierarkin

När ett dokument som har ett överdrivet djup i hierarkin skickas in kan detta leda till en DoS-attack. För att undvika detta kan man slå in en `XmlReader` i klassen som räknar elementens djup. Om djupet överstiger en förutbestämd rimlig nivå kan man avsluta behandlingen av det skadliga dokumentet.

7. Skydda mot implementeringar av `XmlReader` eller `XmlWriter` som är opålitliga

Administratörer bör verifiera att alla `XmlReader`- eller `XmlWriter`-implementeringar som tillhandahålls externt har starka namn och har blivit registrerade i maskinkonfigurationen. Detta förhindrar laddning av skadlig kod som maskeras som en läsare, eng. reader, eller skrivare, eng. writer.

8. Frigör objekt som refererar till `XName` periodiskt

För att skydda mot vissa typer av attacker bör programmerare regelbundet frigöra de objekt som refererar till ett `XName` - objekt i applikationsdomänen.

9. Skydda mot slumpmässiga XML-namn

Ifall de applikationer som man skapar kommer att hämta data från otillförlitliga källor bör man överväga att använda en `XmlReader` som är inslagen i anpassad kod som kontrollerar ifall det finns slumpmässiga XML-namn och namnrymder. Om sådana finns kan programmet sedan avsluta behandlingen av det skadliga dokumentet. Man kanske vill

begränsa antalet namn i en given namnrymd (inklusive namn utan namnrymd) till en rimlig gräns.

10. Serialisering av LINQ to XML-objekt till XML-text innan de skickas till en otillförlitlig komponent

LINQ to XML kan användas för att bygga processpipelines där olika applikationskomponenter laddar, validerar, frågar, transformerar, uppdaterar och sparar XML-data som skickas mellan komponenter som XML-träd. Detta kan hjälpa till att optimera prestanda eftersom omkostnaderna för att ladda och serialisera objekt till XML-text görs endast i slutet av pipeline. Programmerare måste dock vara medvetna om att alla kommentarer och händelsehanterare som skapats av en komponent är tillgängliga för andra komponenter. Detta kan skapa ett antal sårbarheter om komponenterna har olika förtroendenivåer. För att bygga säkra pipelines över mindre betrodda komponenter måste man serialisera LINQ to XML-objekt till XML-text innan data skickas till en otillförlitlig komponent.

En attack som består av höjning av privilegier ger en skadlig assembly mer kontroll över sin miljö. En sådan framgångsrik attack kan resultera i bland annat utlämnande av data (DoS-attacker) och därför bör inte applikationer avslöja data för användare som inte har behörighet att se den. DoS-attacker gör att XML-parsern eller LINQ to XML förbrukar stora mängder minne eller CPU-tid. Eftersom en viss säkerhet tillhandahålls av CLR kan till exempel en komponent som inte innehåller en privat klass inte komma åt annoteringar som den klassen anger. Annoteringar kan dock tas bort av komponenter som inte kan läsa dem, vilket gör att detta kan användas som en manipuleringsattack. (LINQ to XML security, 2021).

4 Utförande

I detta kapitel kommer själva implementationen att bli presenterad. Eftersom detta arbete har blivit gjort som en teknisk utredning på uppdrag av arbetsgivaren kommer implementationen endast att bli presenterad som en teoretisk implementation.

Uppdraget bestod av att utreda hur man skulle kunna förverkliga en plattform där färjerederierna som kund enkelt kan gå in och testa samt köpa tilläggsfunktionaliteter till sin BOOKIT-installation, samt vilka tekniker man skulle kunna använda sig av. Lösningen till plattformen var att man skulle bygga vidare på en redan existerande kundportals plattform där man sedan listar de tilläggsfunktioner som enkelt kan aktiveras utan extra hjälp av en programmerare. För att enkelt kunna uppdatera listan med lämpliga tilläggsfunktioner med ändringar som visas på portalen kom vi fram till att man läsa upp funktionerna ur en XML-fil, som kommer med BOOKIT-installationen.

4.1 Allmänna strukturen

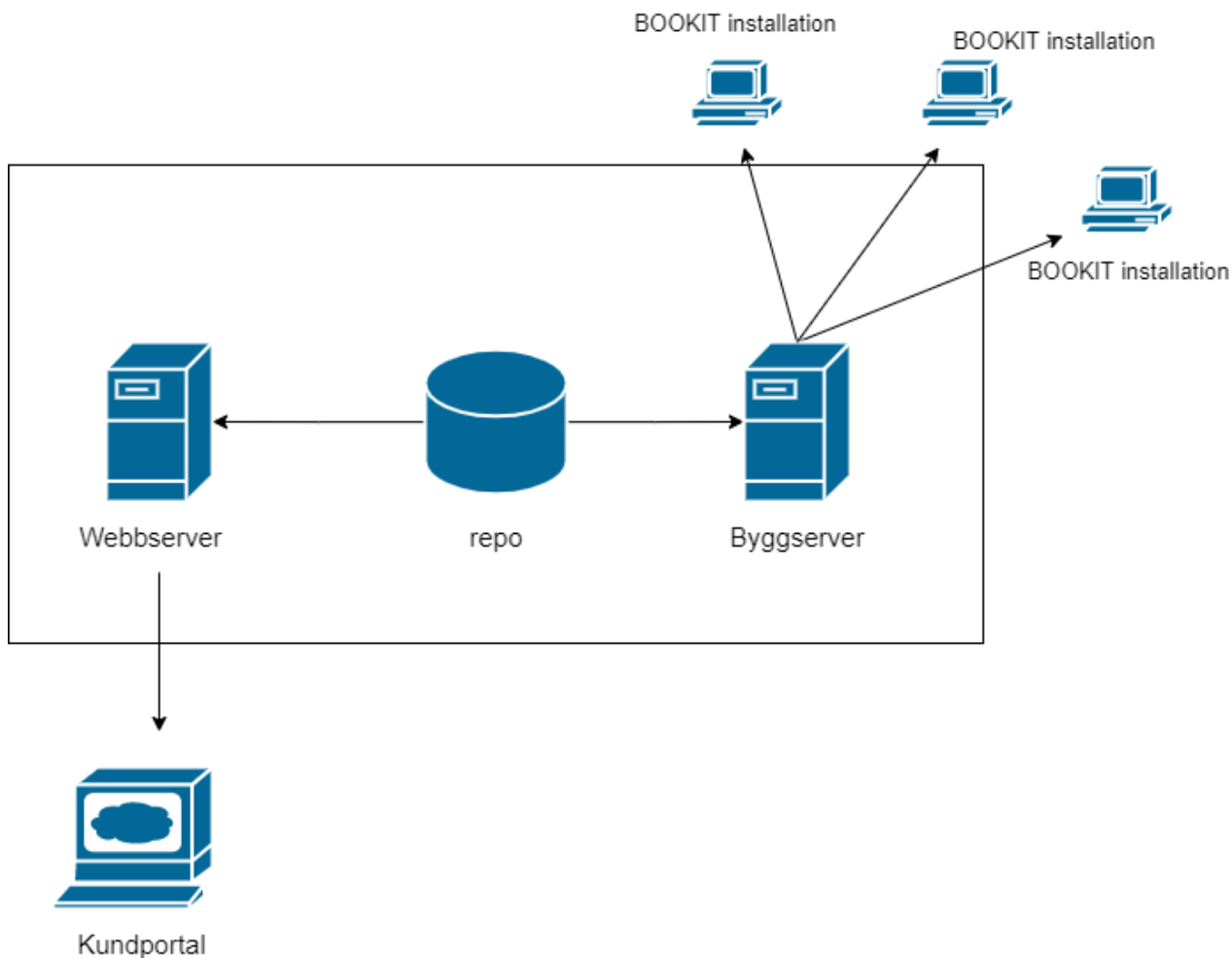
Data om tilläggsfunktioner skrivs in i konfigurationsfilen. Från denna fil läses sedan data in genom att använda LINQ to XML-kod för att gå igenom filen och hitta relevant information. Denna information skickas sedan till webbapplikationen där den visas ut i kundportalen.

Eftersom de tilläggsfunktioner som kommer att listas på kundportalen skall vara sådana funktioner som kunden kan använda, måste funktionerna också vara kompatibla med kundens installation av BOOKIT.

Kodexempel 3. Exempel på hur strukturen ser ut i BOOKIT-konfigurationsfilen.

```
<BookitConfiguration>
  <BootStrappers></BootStrappers>
  <ClientConfigurations></ClientConfigurations>
  <Services></Services>
  <Drivers></Drivers>
  <Features>
    <MessageSystemEnabled>true</MessageSystemEnabled>
    <PaymentFeesEnabled>false</PaymentFeesEnabled>
    <BookingLinksEnabled>true</BookingLinksEnabled>
    <BookingSequenceEnabled>false</BookingSequenceEnabled>
    <CopyBookingEnabled>true</CopyBookingEnabled>
  </Features>
  <DataSource></DataSource>
</BookitConfiguration>
```

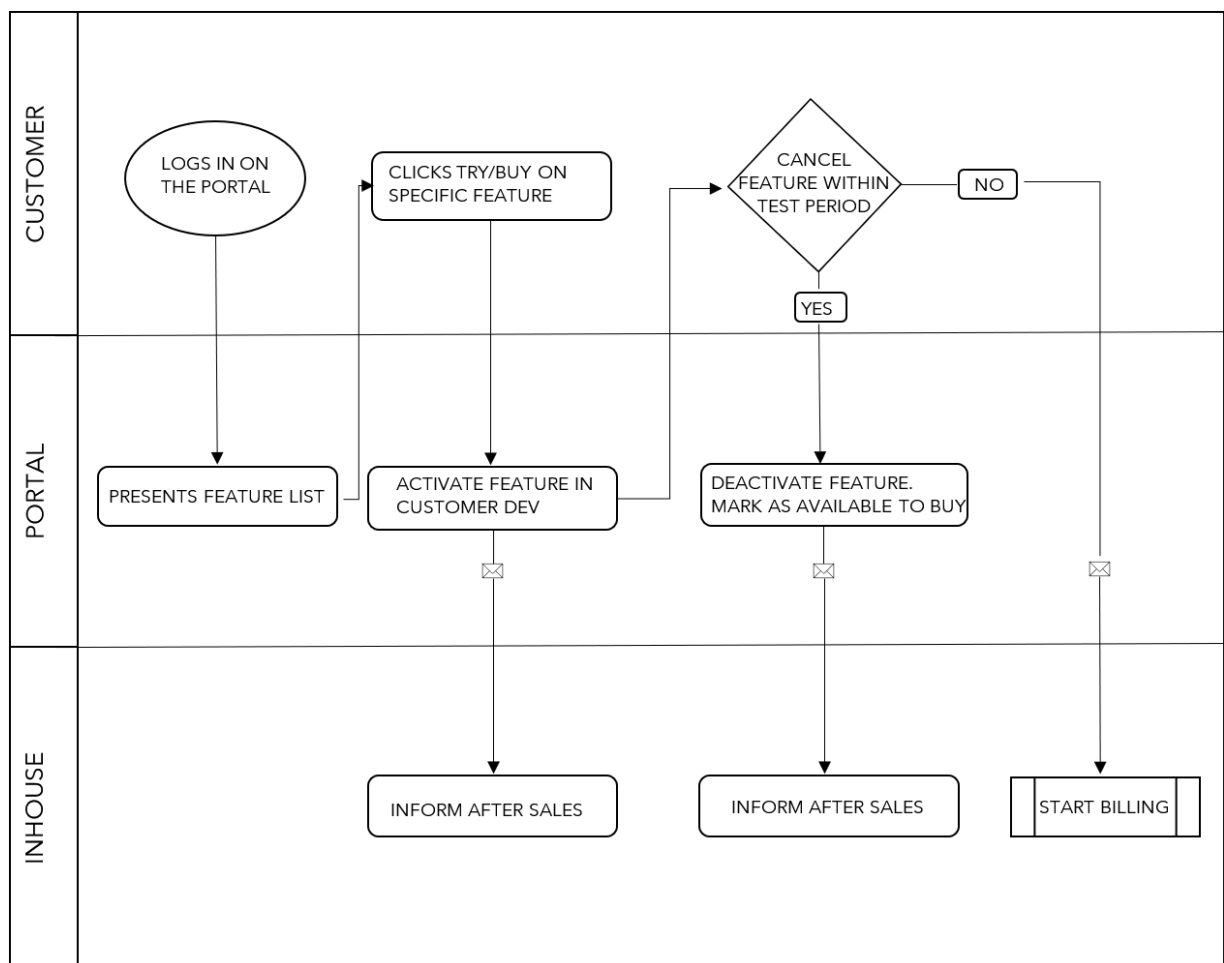
I Kodexempel 3 ser man ett exempel på hur XML-strukturen i konfigurationsfilen för BOOKIT ser ut. I denna fil görs de konfigurationer som kan behöva en specifik inställning som till exempel ett programtillägg som ska aktiveras så parametern ändras till sann. Om det behövs en särskild lösenordspolicy så definieras denna i BOOKIT-konfigurationsfilen. Under Features elementet listas de tilläggsfunktioner som enkelt kan aktiveras med en ändring av sann/falsk parameter, där de tilläggsfunktionerna som har falsk som parametervärde är avaktiverade och de tilläggsfunktionerna med parametervärde satt till sann är aktiverade.



Figur 2. Strukturdiagram över infrastrukturen.

I Figur 2 ser man hur infrastrukturen kommer att se ut när detta arbete har blivit implementerat samt hur data om tilläggfunktionerna kommer att hämtas till kundportalen. Nuvarande infrastruktur går enligt följande, bortsett från webbservern och kundportalen, där källkoden till BOOKIT lagras i repo därifrån blir koden sedan incheckad till byggservern. Byggservern kompilar koden för att den sedan skall kunna bli distribuerad till de olika BOOKIT installationer som finns hos kunderna. Denna distribution sker vid en programuppdatering som kunden själv beställer. Vid implementation av detta arbete kommer webbservern och kundportalen att inkluderas i denna infrastruktur.

Det användarflöde som följer efter att kunden har loggat in på kundportalen till det att kunden antingen köper eller inleder en testperiod av en tilläggfunktion kan urskiljas i Figur 3.



Figur 3. Schematisk bild över användarflödet på kundportalen. Hogia Ferry Systems AB (Diskussionsmöte, 19.2.2021).

Enligt Figur 3 ovan kan man avläsa att kunden börjar med att logga in på kundportalen där de först presenteras av huvudsidan, för att sedan se innehållet om tilläggsfunktionaliteter behöver de navigera vidare till sektionen som heter Features. Där kommer de att se de tilläggsfunktionaliteter som HFS valt att göra tillgängliga för att köpas eller testat via kundportalen. Efter att kunden sedan valt en tilläggsfunktion som verkar intressant och som de inte redan har köpt kan de välja att antingen inleda en testperiod eller att köpa tilläggsfunktionaliteten direkt.

Oberoende av vilket val som görs så kommer tilläggsfunktionaliteten aktiveras i kundens utvecklingsmiljö, som finns specifikt till för att testa nya funktioner. Det kommer på samma

gång att ske ett automatiskt mailutskick till eftermarknadsavdelning hos HFS där information om vilken kund samt vilken tilläggsfunktionalitet som har blivit aktiverad.

Ifall kunden valt att inleda en testperiod så kan de under denna period när som helst välja ifall de vill avbryta testningen och avaktivera tilläggsfunktionen eller efter testperiodens slut köpa tilläggsfunktionaliteten.

Om kunden går enligt Yes-alternativet enligt flödesdiagrammet kommer den valda tilläggsfunktionaliteten att avaktiveras och eftermarknadsavdelning blir informerad om valet med ett mailutskick. Ifall kunden går enligt No-alternativet på flödesdiagrammet kommer både ekonomi- och eftermarknadsavdelningen att bli informerad och den valda tilläggsfunktionaliteten läggs till i faktureringen.

4.2 XML-hantering

Hantering av XML-data blir aktuellt när man skall läsa upp data om tilläggsfunktionerna från konfigurations-filen. För att hitta rätt data i XML-filen så kommer man att använda sig av LINQ to XML i programmeringsspråket C#.

Eftersom Features-noden, som man kan se i Kodexempel 3, är den vi är intresserade av och befinner sig en nivå under huvudnoden kommer LINQ to XML-kodsatsen att först behöva lokalisera huvudnoden för att sedan gå ner till Features-noden där den kommer att söka efter respektive tilläggsfunktionalitetselement. När man hittat rätt element behöver koden sedan avläsa innehållet som kommer att bestå av antingen true för aktiverad eller false för avaktiverad. Det är sedan denna parameter som ska kunna ändras från portalen då man väljer att antingen köpa en tillgänglig tilläggsfunktion eller att påbörja en testperiod för den. Om man väljer att inleda en testperiod måste LINQ to XML-koden kunna behandla detta genom att sätta parametern till false efter att testperioden löpt ut.

Kodexempel 4. Exempel på kod i en klickhändelse för en knapp som ändrar parametervärde från falsk till sann.

```
private void btnBuy_Click(object sender, EventArgs e)
{
    IEnumerable<XElement> feature = from el in
                                    doc.Elements("Features").Descendants()
                                    select el;

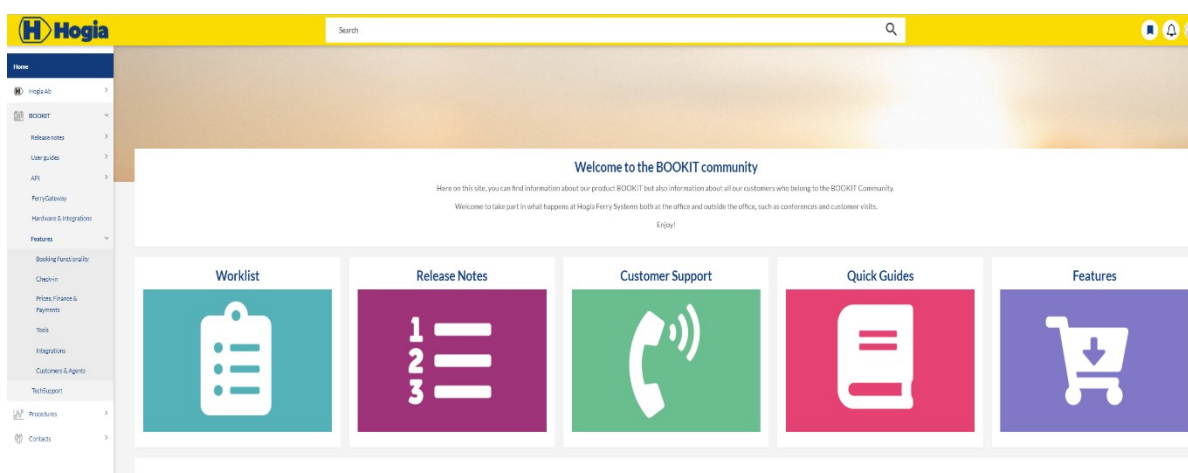
    foreach (var item in feature)
    {
        if (txtFeatureToBuy.Text == item.Name && item.Value == "false")
        {
            item.SetValue("true");
            doc.Save(Constants.BOOKITDEMOFILE);
            break;
        }
    }
}
```

I Kodexempel 4 ovan ser man ett exempel på kod som kommer att ändra parametervärdet på ett element från false till true när man trycker på en knapp. Koden börjar med att hämta upp de element som finns under Features och sparar dessa i en `IEnumerable<XElement>` variabel vid namn `feature`. Man går sedan igenom variabeln och kollar för varje iteration ifall värdet i `item` stämmer överens med de givna villkoren i `if`-satsen. De villkor som skall stämma är att värdet i `item.Name` skall vara lika med text-värdet som blir inmatat i textboxen `txtFeatureToBuy` och på samma gång skall parametervärdet för `item.Value` vara `false`. Ifall villkoren blir uppfyllda kommer koden som finns inuti `if`-satsen att kunna köras vilket gör att värdet som finns i `item.Value` blir satt till `true` och XML-filen sparas sedan avbryts `foreach`-loopen och klickhändelsen för knappen är färdig körd.

4.3 Implementation

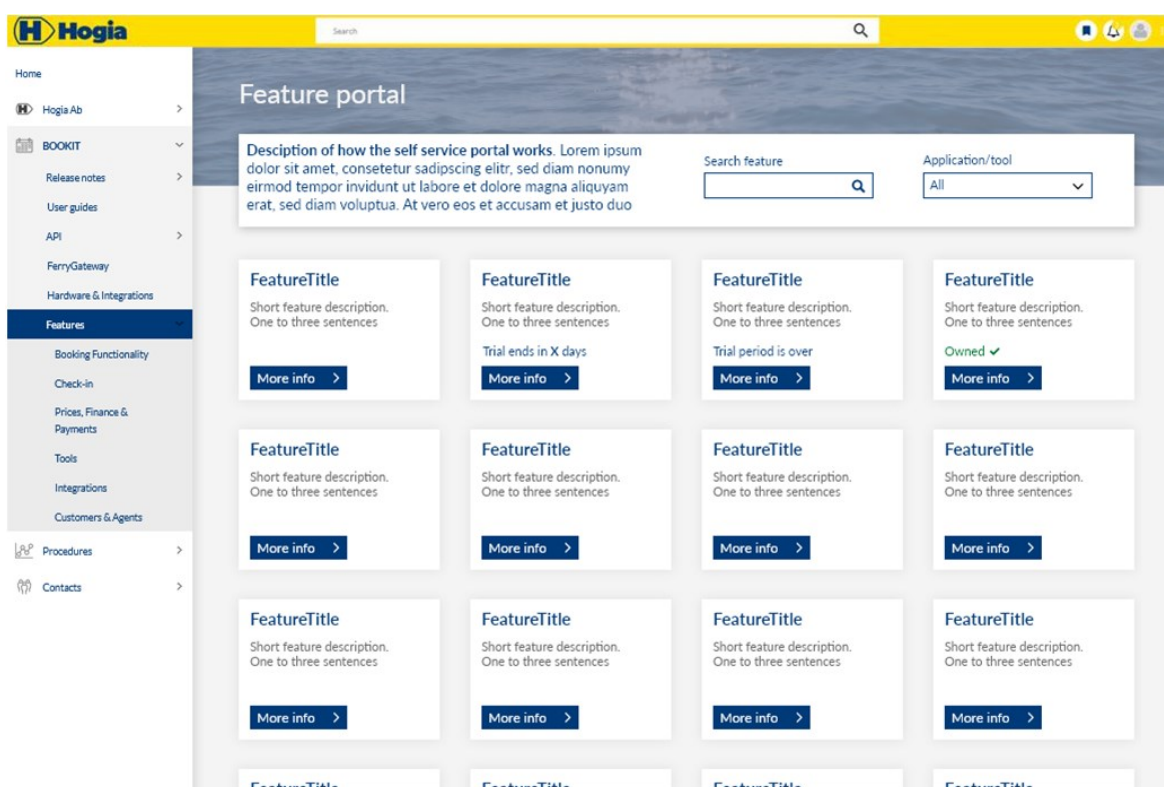
Under denna rubrik kommer konceptidén med kundportalstillägget att presenteras samt hur det kan börja se ut efter att det blivit implementerat.

Efter att ha loggat in på kundportalen ser kunden sidan i Figur 4. I menyn till vänster finns länkar till olika sidor beroende på vad man vill hitta för information som till exempel Release notes, Användarmanualer eller API dokumentation. För att se de tilläggsfunktioner som är tillgängliga för att antingen köpas eller testas, navigerar man till Features-sidan genom att klicka i menyn till vänster.



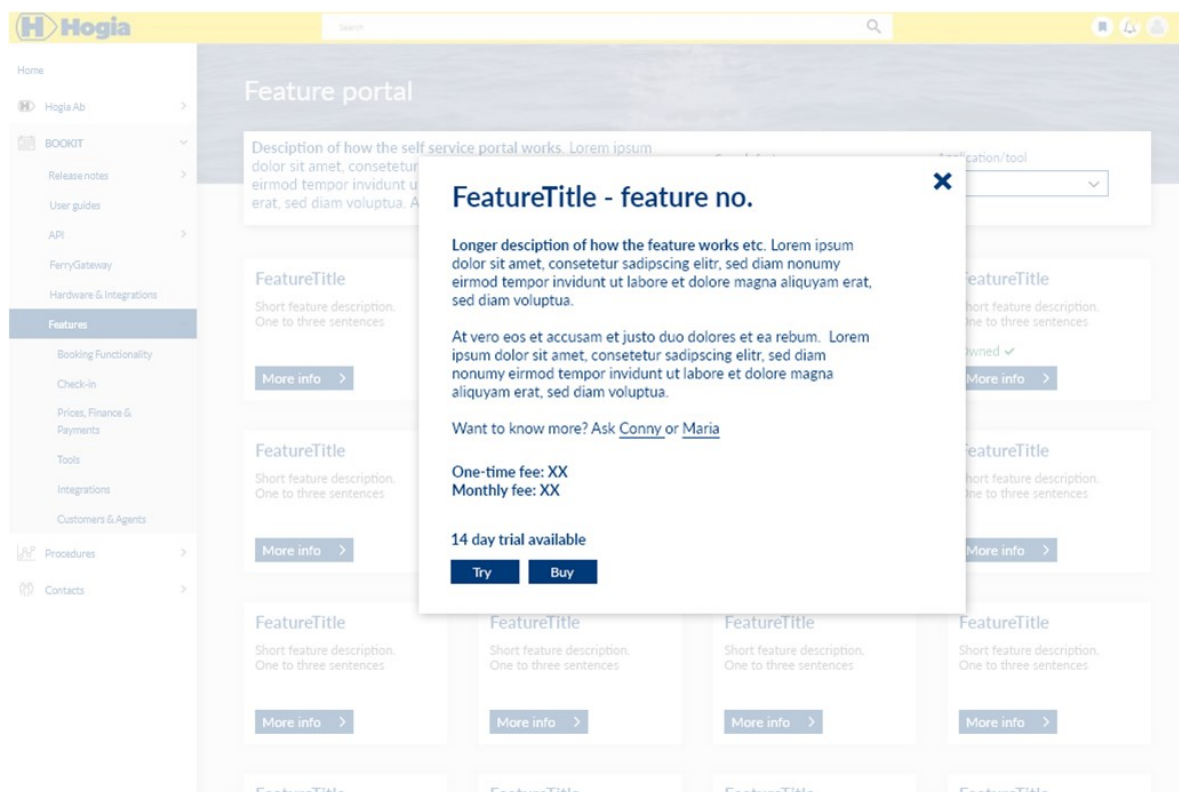
Figur 4. Huvudsida på kundportalen efter inloggning.

Efter att detta arbete blivit implementerat ska man mötas av denna sida när klickar på Features-avsnittet. Denna sida ska då visa ut de tilläggsfunktioner som är tillgängliga att antingen köpas eller testas och de tilläggsfunktioner som redan blivit köpta. Men eftersom en kunds BOOKIT-installation kan vara olik en annan kunds installation betyder det att alla tilläggsfunktioner inte kommer att vara kompatibla med alla olika BOOKIT-installationer. Ett exempel på hur sidan med tilläggsfunktioner kommer att se ut syns i Figur 5.



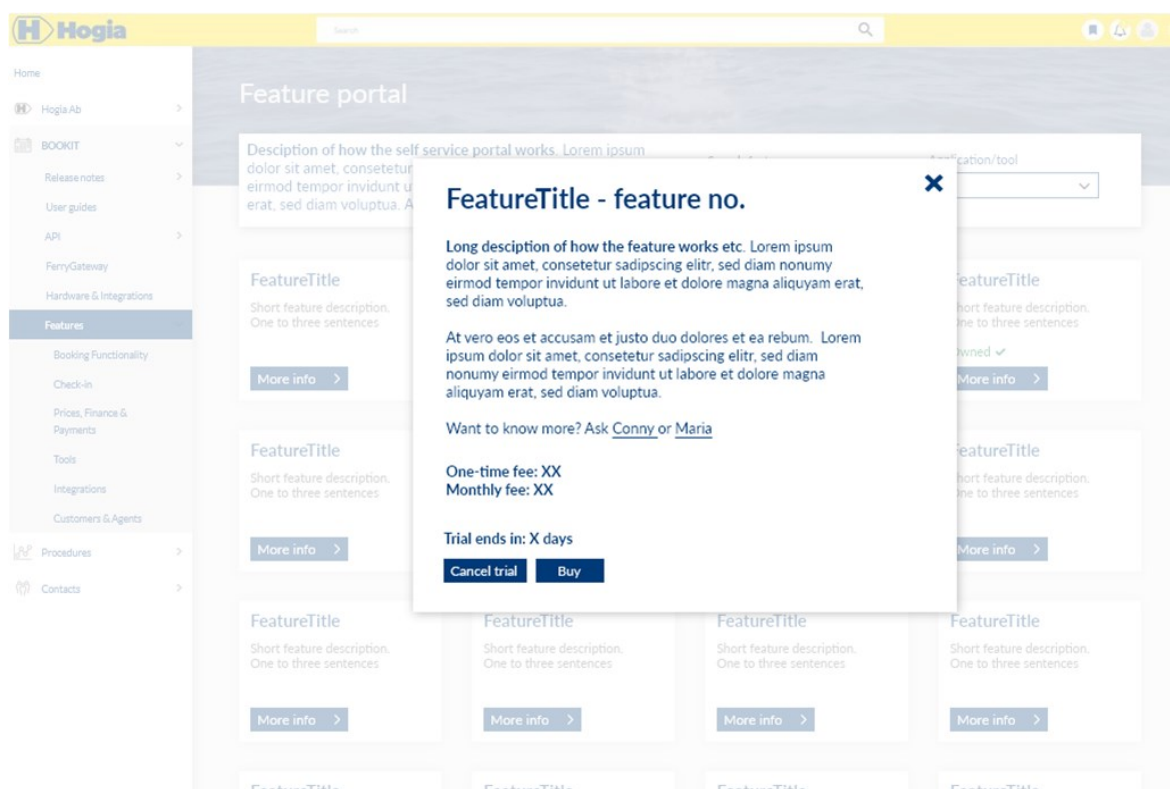
Figur 5. Tilläggsfunktioner i kundportalen.

Efter att ha klickat på en tilläggsfunktion som verkar intressant kommer kunden att mötas av två olika alternativ, att antingen köpa tilläggsfunktionen eller inleda en testningsperiod på 14 dagar. Hur detta kommer se ut ser man ett exempel på i Figur 6.



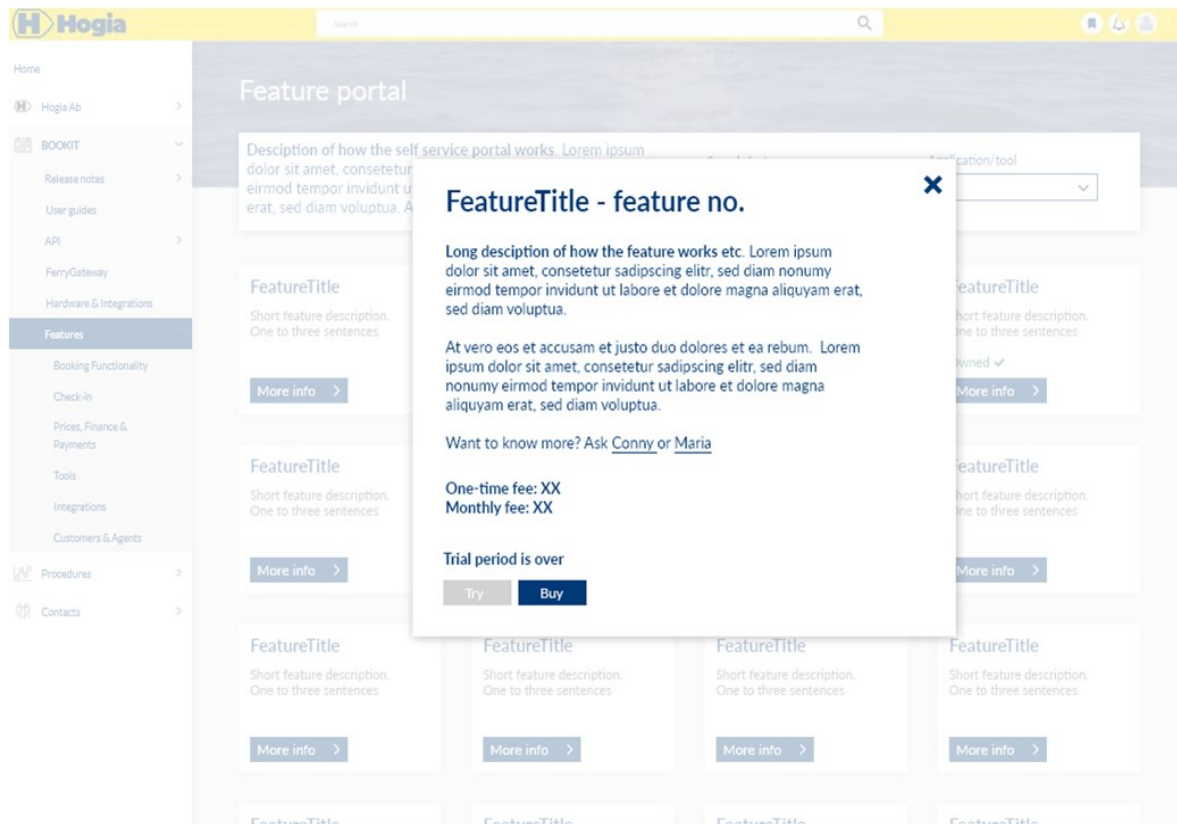
Figur 6. Hur det ser ut när man klickar på en tillgänglig tilläggsfunktion.

Ifall man väljer att inleda en testperiod kommer det att se ut enligt Figur 7. Det dyker då upp en rad ovanför knapparna där det står hur lång tid som återstår av testperioden. Ifall kunden inte väljer att avbryta testperioden innan den tar slut kommer tilläggfunktionen att automatiskt börja räknas som att den blivit köpt och ett automatiskt mejl blir skickat till ekonomiavdelningen om att en tilläggsfunktion har satts till.



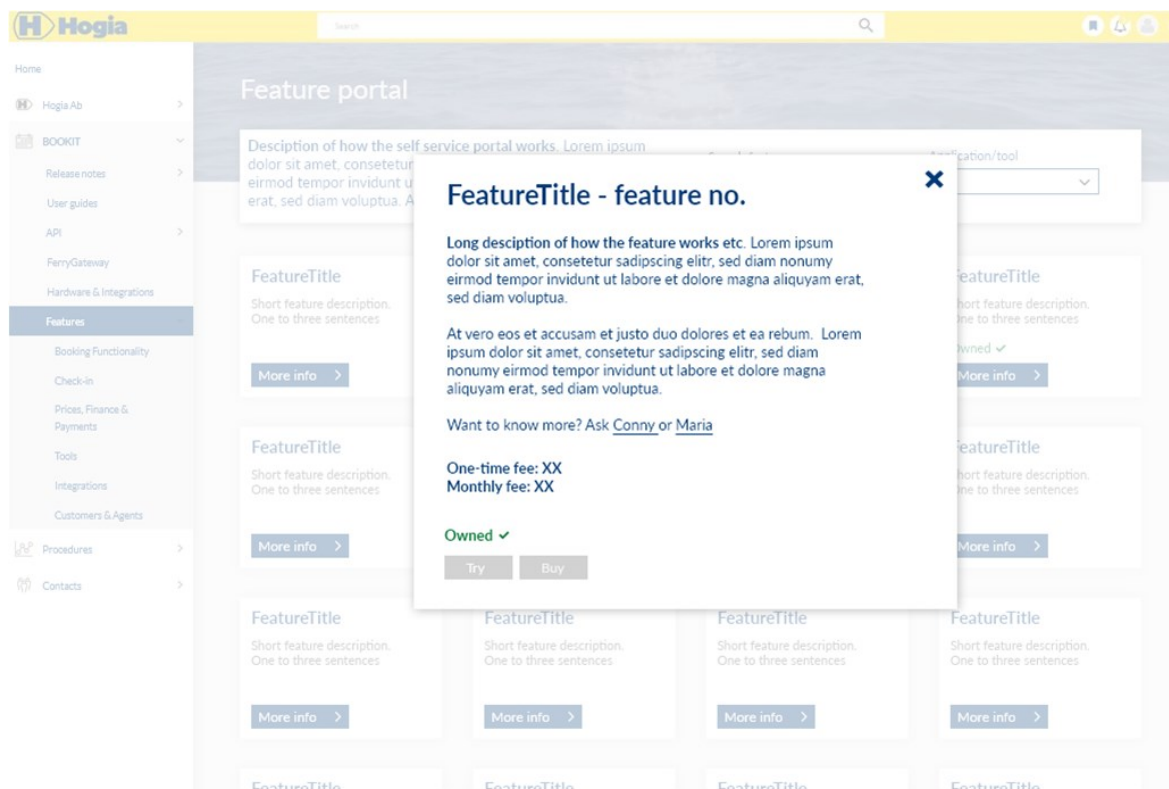
Figur 7. Tillägg aktiverat med testperiod.

Efter att testperioden är slut kommer det att se ut enligt Figur 8 där kunden nu bara kommer att ha ett alternativ kvar att välja på vilket är att köpa tilläggsfunktionen.



Figur 8. Testperioden är slut för tilläggsfunktionen.

Ifall kunden valt att köpa tilläggsfunktionaliteten direkt eller om den blir köpt efter att testperioden tagit slut kommer det att se enligt Figur 9.



Figur 9. Köpt tilläggfunktion.

Ifall kunden valt att köpa en tilläggsfunktion kommer det att skickas ett meddelande åt ekonomiansvarig på HFS samt eftermarknadsavdelningen och slutkundens kontaktperson på HFS med information om vilken tilläggsfunktionalitet kunden valt att köpa. Ett liknande meddelande kommer också att skickas ifall man väljer att inleda en testperiod av en tilläggsfunktion eller ifall man väljer att avbryta testperioden men då bara åt eftermarknadsavdelningen och slutkundens kontaktperson på HFS.

Till en början kommer meddelandet att bara bestå av ett automatiserat mailutskick från kundportalen som sedan kan byggas vidare på. Målet är att uppnå en i regel helt automatiserad portal där man kommer kunna inleda och avbryta en testperiod och köpa enklare tilläggsfunktionaliteter till den egna installationen av BOOKIT.

5 Resultat och diskussion

Uppgiften för examensarbetet var att göra en teknisk utredning om hur man skulle kunna förverkliga en plattform där HFS kunder skulle kunna aktivera tilläggsfunktioner till sina egna installationer av bokningssystemet BOOKIT.

Resultatet av den tekniska utredningen blev att man kan bygga vidare på en redan existerande kundportal genom att man exempelvis listar enklare tilläggsfunktioner som kan aktiveras utan extra hjälp av en programmerare. Listan kan sedan uppdateras med lämpliga tilläggsfunktioner som visas på portalen. För att göra detta kan tilläggsfunktionerna läsas upp ur en XML-konfigurationsfil som kommer med BOOKIT-installationen och för att hitta rätt data om tilläggsfunktionerna i konfigurationsfilen använder man sig av LINQ to XML i programmeringsspråket C#. Datan skickas sedan till webbapplikationen där den visas ut i kundportalen. De tilläggsfunktioner som kommer listas på kundportalen skall vara sådana funktioner som både är lämpliga för en aktivering på portalen och användningsbara för kunden. Funktionerna måste därmed vara kompatibla med kundens installation av BOOKIT samt kunna aktiveras genom att ändra parametervärdet från falsk till sann.

Även om den undersökta lösningen fungerar så finns det en risk att lösningen snabbt blir föråldrad. För att göra tillägget till kundportalen mer framtidssäkert och även framtidssäkra möjligheterna för vidareutveckling så skulle man i stället kunna exportera datan om tilläggsfunktionerna till en databas och därifrån läsa upp och hantera den. Eftersom den första iterationen av denna lösning bara innefattar tilläggsfunktioner som har en relativt låg komplexitet och är enkla att aktivera, kan en vidareutveckling av lösningen också omfatta mer komplexa tilläggsfunktioner och svårare aktiveringsmodeller.

Det som varit mest givande och intressant med detta arbete så är att jag har fått planera och fundera ut en större utvecklingslösning på egen hand. Jag är överlag nöjd med slutresultatet även om det finns delar som går att vidareutveckla och förbättra. En förhoppning är att detta slutarbete skall kunna implementeras och underlätta aktiveringen av tilläggsfunktioner från kundens sida.

Det jag upplevde som en av de största utmaningarna med arbetet var att det till största delen var teoretiskt, detta gjorde det svårare att visualisera slutresultatet. Det skulle ha varit intressant att få se en fungerande version av arbetet i praktiken och inte enbart hur det kan fungera teoretiskt.

Referenser

- Hogia BOOKIT - standard applications*. (u.d.). Hämtat från hogia.se:
<https://www.hogia.se/int/ferry-system/bookit> den 05 2 2022
- Vår historia*. (u.d.). Hämtat från hogia.se:
<https://www.hogia.se/hogiagruppen/v%C3%A5r-historia> den 05 2 2022
- W3Schools. (u.d.). *W3Schools*. Hämtat från C# Introduction:
https://www.w3schools.com/cs/cs_intro.php den 17 2 2022
- Wagner, B., & Kramer, J. (den 15 9 2021). *LINQ to XML security*. Hämtat från docs.microsoft.com: <https://docs.microsoft.com/en-us/dotnet/standard/linq/linq-xml-security> den 05 2 2022
- Wagner, B., Kramer, J., & Wenzel, M. (den 15 9 2021). *LINQ to XML overview*. Hämtat från docs.microsoft.com: <https://docs.microsoft.com/en-us/dotnet/standard/linq/linq-xml-overview> den 05 2 2022
- Wagner, B., Santos, R., Sharkey, K., Coulter, D., Newcomb, B., Warren, G., . . . Onderka, P. (den 30 11 2021). *A tour of the C# language*. Hämtat från docs.microsoft.com: <https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/> den 19 2 2022
- Warren, G., Coulter, D., Lee, D., Nakamura, E., Wenzel, M., A, A., . . . Wagner, B. (den 15 9 2021). *Overview of .NET Framework*. Hämtat från docs.microsoft.com: <https://docs.microsoft.com/en-us/dotnet/framework/get-started/overview?redirectedfrom=MSDN> den 05 2 2022