

Återkoppling till punkter i kundens kravspecifikation

1. Tillgång till Koden

För att säkra att koden inte går förlorad samt att en ny utvecklare kan få åtkomst till koden har projektet lagrats i den webbaserade lagringstjänsten GitHub som tillämpar versionshanteringssystemet Git. Denna lösning möjliggör för flera utvecklare att samarbeta oberoende av plats.

2. Kom igång med koden

Förutom tillgång till koden initialt via GitHub behöver framtida utvecklare viss mjukvara installerad för att vidareutveckla samt underhålla projektet.

Länk till GitHub Repository:

<https://github.com/JacobMcElhinney/CentrumBiblioteketRepository.git>

Observera att CentrumBiblioteket behöver ett GitHub konto för att skapa ett eget repository inför produktionssättnings-processen.

2.1. Nödvändig Programvara/Utvecklingsmiljö

- **Microsoft Visual Studio (VS):** För underhåll och vidareutveckling av applikationen samt de klasser/modeller som används för att skapa up tabeller i kundens SQL databas.
 - **GitHub Extension for Visual Studio:** För att kunna arbeta i VS mot ett GitHub Repository oberoende av plats med praktisk versionshantering.
 - **Entity Framework (EF):** Nödvändigt ramverk för möjliggöra VS kompatibilitet med SQL databasen.
 - **Nuget packages:** Nödvändiga paket för att utnyttja kraften i EF (OBS! Samtliga paket redan inkluderande i koden/Projektet).
 - "Microsoft.AspNetCore.Mvc.NewtonsoftJson" Version="3.1.10"
 - "Microsoft.EntityFrameworkCore" Version="3.1.10"
 - "Microsoft.EntityFrameworkCore.Design" Version="3.1.10"
 - "Microsoft.EntityFrameworkCore.Sqlite" Version="3.1.10"
 - "Microsoft.EntityFrameworkCore.SqlServer" Version="3.1.10"
 - "Microsoft.EntityFrameworkCore.Tools"
 - "Microsoft.VisualStudio.Web.CodeGeneration.Design"
- **Microsoft Sql Server Management Studio (SSMS):** För underhåll och vidareutveckling av databasen samt direkt manipulation av data.
- **Postman:** Programmet gör det möjligt att enkelt skriva till och läsa från databasen via HTTP anrop som går via ett API (del av applikationen som kommunicerar med databasen utan grafiskt användargränssnitt).

3. Användning och Produktionssättning av koden

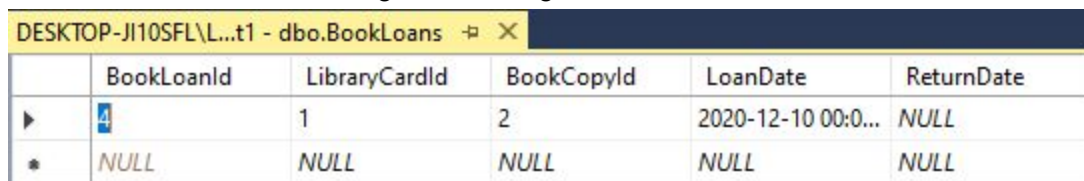
Nedan följer kortfattad förklaring av hur koden kan komma att användas på riktigt samt en översiktlig plan för produktionssättning och uppdatering.

3.1 Användning av koden (Ny Funktionalitet) - Wireframe:

När applikationen (MVC projektet) körs returneras en vy till användaren. Denna vy inkluderar ett "viewmodel object" med data från databasen. Data filtreras innan den returneras så att endast den efterfrågade informationen presenteras för användaren.

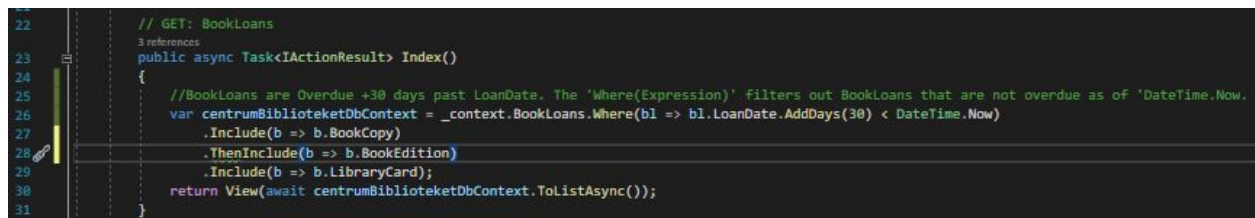
Följande bildserie (4 bilder) illustrerar program körningens mest centrala steg (Funktionen):

Bild 1: Data tabell där samtliga boklån registrerats



	BookLoanId	LibraryCardId	BookCopyId	LoanDate	ReturnDate
▶	4	1	2	2020-12-10 00:00:00...	NULL
•	NULL	NULL	NULL	NULL	NULL

Bild 2: logik i applikationen som filtrerar data så att endast försenade boklån visas



```
// GET: BookLoans
3 references
public async Task<ActionResult> Index()
{
    //BookLoans are Overdue +30 days past LoanDate. The 'Where(Expression)' filters out BookLoans that are not overdue as of 'DateTime.Now'.
    var centrumBiblioteketDbContext = _context.BookLoans.Where(bl => bl.LoanDate.AddDays(30) < DateTime.Now)
        .Include(b => b.BookCopy)
        .ThenInclude(b => b.BookEdition)
        .Include(b => b.LibraryCard);
    return View(await centrumBiblioteketDbContext.ToListAsync());
}
```

Bild 3. Den vy som returneras till användare (resultatet i form av en "wireframe" illustration):

Overdue Bookloans

BookLoanId	BookTitle	FirstName	LastName	PhoneNumber	LoanDate
4	Gardening for Dummies	Jacob	McElhinney	0761300175	10/12/2020 12:00:00 AM

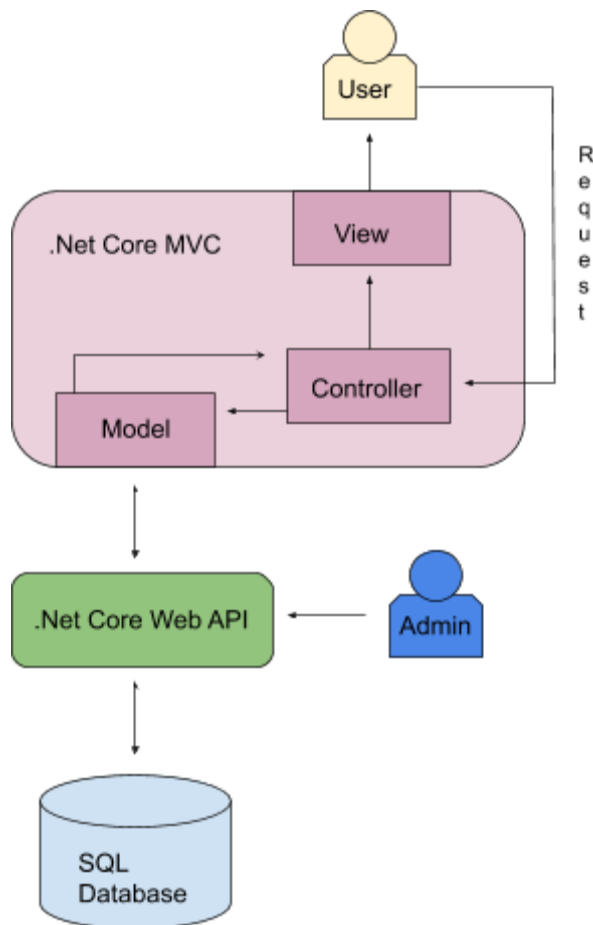
Användaren får här tillgång till samtliga försenade boklån samt de nödvändiga uppgifter som efterfrågats för att kunna kontakta bibliotekskunder.

För att ta bort ett boklån som registrerats går man via applikationens API (Application Programming Interface). Eftersom HTTP anrop mot ett API kräver viss teknisk kunskap är detta en funktionalitet som högst troligt kommer att utvecklas mot en mer användarvänlig lösning. För

närvarande kan Centrum Bibliotekets databas administratör använda sig av API:et för att utföra olika operationer på databasen.

I framtiden kan denna process skall automatiseras med hjälp av vidareutveckling samt teknisk utrustning (streckkodsläsare) som enkelt kan läsa av en bok och via API:et registrera den som returnerad i databasen. För närvarande kan administratörer använda sig av Postman eller SSMS (se 2.1 Nödvändig programvara/Utvecklingsmiljö ovan) för att göra ändringar i databasen.

Bild 4. Schemat nedan illustrerar hur komponenter och teknologier samverkar i processen:



OBS! Se också ReadMe.md filen i GitHub repot som beskriver den nya funktionen i ännu större detalj, dock med mer tekniska termer.

3.2 Produktionssättning

Målet med denna process är att CentrumBiblioteket, efter avslutad produktionssättning (samt kvalitetssäkring i form av testning - se efterföljande rubrik), skall ha ett fungerande system redo att användas med den nya funktionaliteten.

Produktionssättning via Azure är en enkel och säker lösning som fungerar bra tillsammans med den utvecklingsmiljö och teknik som använts i projektet. Den tidigare kvalitetssäkrade delen av projektet (API funktionalitet) kan produktionssättas till ett Azure App Service genom Visual studio. Denna officiella dokumentation från Microsoft beskriver processen i en stegvis guide samt de ytterligare "exteions" som behövs för att produktionssätta med Azure:

Länk: [Deploy to Azure App Service using Visual Studio Code - Azure Pipelines | Microsoft Docs](#)

För närvarande finns endast driftmiljön "development". Azure tillhandahåller dock en säker testmiljö. Vidare configuration kommer också krävas för att applikationen skall få åtkomst till Centrum Bibliotekets databas. Tack vare generisk kod krävs endast två enkla modifikationer av nyckelvärdes-paret "DefaultConnection" och den sträng som angetts. Server, Database, User Id samt password måste stämma överens med Centrum Bibliotekets egna databas. Applikationen i sin helhet (the entire solution) består av två projekt som vardera har en fil vid namn "appsettings.json" där dessa ändringar skall genomföras. Observera att en ny migration för att skapa upp tabellerna i databasen behöver göras, om de skiljer sig från befintliga tabeller i databasen. Notera att detta kan innebära att data går förlorad, säkra därför befintlig data innan en migration genomförs.

Innan den nya funktionen produktionssätts skall den kvalitetssäkras (se 4. Kvalitetssäkring) genom testing. Testa individuella funktioner innan de läggs till i det befintliga Azure repository.

4. Kvalitetssäkring

För att säkerställa att den nya funktionen stämmer överens med kravspecifikationen samt fungerar felfritt behöver den kvalitetssäkras genom testning. Denna process kan ses som en del av produktionssättningen och skall genomföras med lyckat utfall innan produkten slutligen driftsätts. Nedan följer en Test plan samt förslag på specifika tester för avgränsade funktioner av applikationen

4.1 Test Plan

Efter varje förändring i koden (ny version av applikationen) skall ett regressionstest utföras för att säkerställa att följande funktioner fortfarande fungerar efter att den förändring (variation) som gjorts i koden.

4.1.1 Regressionstest

Regressionstest av API ("CentrumBiblioteket"):

1. I Visual studio ange "CentrumBiblioteket" som startup projekt och starta debuggern (IIS Express)
2. Kopiera URL från browserns adressfält när appen kört igång.
3. Starta Postman komplettera URL:en med URI: "/Api/BookLoans" och gör ett GET anrop
 - Kontrollera att resultatet genererar ett "Status: 200 OK" utfall
 - Försäkra dig om att data från tabellen BookLoans returneras i Json objekt (alternativt en tom array om ingen data finns i databasen)
4. Samma steg kan upprepas för övriga tabeller
5. Lägg till "dummy data" med "POST" till samtliga tabeller kopplade till "BookLoans".
6. Använd sedan POST-funktionen för att registrera ett nytt "dummy-bookLoan" vars lånedatum initialiseras som minst 31 dagar tillbaka i tiden (för senare test).
7. Återupprepa steg 3 och försäkra dig om att det nya dummy-BookLoan returneras samt status: 200 OK. Regressionstest av av API nu slutfört.

Regressionstest av MVC app ("MVCFunctionality"):

1. Ange "MVCFunctionality" som startup projekt och starta debugger (IIS Express)
2. En vy skall nu returneras i browsern där samtliga BookLoans i databasen, vars LoanDate är längre tillbaka i tiden än trettio dagar, listas under "Overdue BookLoans"
 - Kontrollera att samtliga BookLoans listas samt det dummy-BookLoan som lades till under API testet.
3. Förutsatt att detta är utfallet är testet av "MVCFunctionality" nu slutfört. Stegen i ovanstående två testsekvenser kan med fördel upprepas för att på så vis ta bort dummy-BookLoanet via API och sedan säkerställa att det inte dyker upp i vyn under MVC app test.

4.1.2 Smoke Test

För att försäkra att den nya funktionen och applikationen fungerar som det är tänkt skall ett slutligen ett **smoke test** utföras. Exempelvis kan bibliotekarierna själva få agera testare efter att programmet finns tillgängligt på någon av dem datorer det är tänkt att användas på.

4.1.3 Övriga Reflektioner

Eftersom Postman förhindrar felaktig input behövs minimalt med validering för att vårt API skall

fungera utan risk för exekveringsfel. Viss validering har dock inkluderats bland annat i BookLoansControllern (se ReadMe.mg) samt vissa "properties" i klasser/modeller
Exempelvis:

```
//Added attributes for input validation.  
[Required]  
[RegularExpression(@"[a-z]{2,}", ErrorMessage = "Please input {yes} or {no}.")]  
[StringLength(3, ErrorMessage = "Please input {yes} or {no}.", MinimumLength = 2)]  
2 references  
public string Available { get; set; }
```

I detta exempel förhindras att data som ej är i rätt format ej kan lagras i ett fält (cell i kolumn) samt att ett felmeddelande returneras med instruktioner.