

Joakim Lönngren

Nackademin

Programutveckling .NET – PROG23

# Examensarbete - iXpense

En applikation för kvittoregistrering

1. Inledning
  - 1.1 Bakgrund
  - 1.2 Syfte
  - 1.3 Mål
- 2 Krav
  - 2.1 Kravspecifikation
  - 2.2 Prioriteringsordning
- 3 Metod och arbetssätt
- 4 Teknisk specifikation
  - 4.1 Tekniker och ramverk
    - 4.1.1 – Backend
    - 4.1.2 - Frontend
  - 4.2 Arkitektur
    - 4.2.1 Lagerstruktur i backend
    - 4.2.2 Swagger och Endpoints
  - 4.3 Entity Framework Core
  - 4.4 Databas
  - 4.5 Autentisering och säkerhet
- 5 Testning och validering
  - 5.1 Backend
  - 5.2 Frontend
- 6 Diagram
- 7 Versionshantering
  - 7.1 GitHub repository iXpenseBackend
  - 7.2 GitHub repository iXpenseClient
- 8 Tänkt att implementera
- 9 Avslutning och reflektion
  - 9.1 - Vad gick bra?
  - 9.2 - Utmaningar/svårigheter
  - 9.3 - Kunde jag gjort något annorlunda?
  - 9.4 - Egenskaper jag utvecklat under projektet

## **1. Inledning**

### **1.1 Bakgrund**

Bakgrunden för denna applikation bygger på att jag själv under en tid tyckt att det skulle vara av intresse att använda sig utav en applikation som hanterar ens utgifter och som på ett enkelt och snabbt sätt kan redovisa detta vid behov.

### **1.2 Syfte**

Tanken med detta projekt är att det ska ligga som grund för en större lösning i framtiden, där man skapar en bra förutsättning till att vidareutveckla denna applikation. I applikationen ska man kunna registrera sig som användare, registrera sina utgifter – när man senare använt applikation så är tanken att man ska kunna se vart man spenderar mest pengar och om man på något sätt kan ändra ett potentiellt onödigt beteende i sitt spenderingsmönster.

### **1.3 Målsättning**

Målsättning för mitt examensarbete var att skapa en bra grund för en applikation som så enkelt som möjligt kan skalas upp i framtiden och addera nya funktioner alternativt innehåll. Tanken är i framtiden att kunna använda sig utav bibliotek eller extensions som kan hantera exempelvis BAR-codes alternativt tolka information ifrån en bild och applicera detta som kvittoinnehåll.

## **2. Krav**

### **2.1 Kravspecifikation**

- Applikationen ska kunna registrera och logga in användare.
- Autentisering ska ske med hjälp av Identity och JWTs.
- Varje användare ska kunna registrera kvitton som enbart är kopplade och synliga för den användaren.
- Varje kvitto ska innehålla minst en produkt, en produkt ska tillhöra en kategori samt att kvittot ska tillhöra en användare.
- Entiteter sparas i en SQL-databas.
- Under processen när man skapar ett kvitto så ska man kunna växla mellan olika steg utan att information går förlorad.
- Ett enkelt och modernt UI med välstrukturerad SCSS.
- En användare ska kunna visa alla sina kvitton.
- Få redovisat diagram i en eller flera former som visar vilka produkter eller kategorier man spenderat mest pengar på under en viss period.

## 2.2 Prioriteringsordning

För att förhålla sig till tidslängden på examensarbetet som är 6 veckor lång så är prioriteringen att få en så bra grund som möjligt där möjligheterna för att bygga en skalbar lösning är prioriteringen. En initialtanke var att användaren skulle kunna få ut genererade tips på hur man kan förbättra sin spendering baserat på hur användaren har spenderat över tid. Detta insåg jag ganska fort skulle bli svårt då tiden för att bygga en sådan funktion skulle bli för knapp för att kunna göra tillräckligt bra. Prioriteringsfrågan för vad som ska göras har blivit följande:

### 2.2.1 Backend

- Se till att Backend skapas och innehåller rätt paket för förutsättningarna som existerar. Detta innefattar:
  - Microsoft.AspNetCore.Authentication.JwtBearer
  - Microsoft.AspNetCore.Identity.EntityFrameworkCore
  - Microsoft.EntityFrameworkCore.SqlServer
  - Microsoft.EntityFrameworkCore.Tools
- Skapa Sql-databas genom code-first med rätt kopplingar mellan entiteter:
  - Category.cs
  - Item.cs
  - Receipt.cs
- Versionshantering i GitHub för backend.
- Skapa relevanta DTO:er som används vid sina specifika syften.
- Sätta upp initial lagerstruktur:
  - Context
  - Repository
  - Service
  - Controller
- Sätta upp Registrering, Login samt skapa JWT-hantering.
- Skapa fungerande Category och Item-lagerstruktur.
- Påbörja skapandet av Receipt-lagerstruktur.
- Mid-cycle check för att se om Backend fungerar som den ska gällande allt det övre.

### 2.2.2 Frontend

- Skapa Frontend-grund med React-vite och se till att projektet är cleant och redo att produceras i, att det innehar rätt certifikat och kan skicka data via HTTPS.
- Versionshantering i GitHub för frontend.
- Skapa en Layout-struktur som innehåller en Header-del och en Main-del, detta kommer användas för alla huvud-pages förutom Register och Login.
- Skapa en ApiClient som ansvarar för den huvudsakliga [URL:en](#) som anropas samt token-hämtning.
- Skapa ApiUser som använder ApiClient vid varje anrop men specificerar vad den har för uppgift, loginUser eller registerUser.
- Skapa login och register-page och useContext som handhåller information i localStorage.
- Skapa en profilePage som agerar som sidan man kommer till efter man loggat in.
- ProfilePage:n ska ge enkel information om iXpense och härifrån ska man kunna navigera till alla andra sidor samt kunna logga ut.
- Skapa CreateReceipt som ska innehålla ett flerstegs formulär med sidor som:
  - StoreAndDate
  - AddProduct
  - ReceiptSummary
- Skapa FormContext som hanterar information om vilket steg användaren är på och hanterar datan som användaren skickar in.
- Skapa ReceiptList där användaren får upp sina kvitton och när användaren trycker på ett kvitto kommer fullständig och detaljerad information upp.
- Skapa en SummaryPage där användaren kan välja start och slut-datum och då få upp visuell representation av data ifrån användarens kvitton i form av diagram, information av vilken produkt som användaren inhandlat mest samt toppkategori.
- Skapa enkel men håll och skalbar css genom SASS och module.scss.

## 3 Metod och arbetssätt

Jag valde att axla detta projekt med att först bygga en fungerade backend med tydlig struktur och indelning av olika sektioner i form av lagerstruktur samt entiteter och DTO:er som jag visste skulle komma till användning. När detta hade uppnåtts fokuserade jag på att få till en registrering för användare och sedan funktionalitet för att kunna logga in. Härifrån blev det ett lite mera dynamiskt mellan de olika projekten och addering av funktionalitet blev implementerat i hela kedjan (frontend och backend) efter behov. Designtanken för klienten var att dela in majoriteten av komponenterna i egna module.scss och hålla det väldigt simpelt tills

funktionaliteten och innehållet var bestämt. Testningen varierade mellan att använda sig utav Swagger samt att använda sig utav browser-information.

## **4 Teknisk specifikation**

### **4.1 Tekniker och ramverk**

Jag kommer använda mig utav dessa teknologier/ramverk:

#### **4.1.1 Backend**

ASP.NET Core  
Entity Framework Core  
Microsoft SQL Server  
ASP.NET Identity  
Json Web Token  
Swagger

#### **4.1.2 Frontend**

React.js  
ReCharts – visuell representation i form av diagram baserad på data.  
Axios – Skickar anrop till backend huvudsakligen via ApiClient.  
JwtDecode – Dekrypterar JWT-token.  
SASS – via modulbaserad CSS som är enkel att återanvända.

## **4.2 Arkitektur**

### **4.2.1 Lagerstruktur i backend**

- Controller – presentationslagret som tar emot request ifrån klienten. Stämmer av så att informationen som skickas är giltig för att sedan skickas vidare till rätt service-lager. Returnerar svar och status till klient.
- Service – Logiklagret som hanterar affärslogiken och hur datan som finns i requesten ska behandlas.
- Repository – lagret som är i direktkontakt med databasen.
- DbContext – kopplingen mellan applikation och databasen genom Entity Framework Core. Ansvarar för att utföra det som Repository-lagret begär.

#### 4.2.2 Swagger och Endpoints

Swagger-UI:et ger en översikt på de olika endpoints som backend:en förser. I detta fall innehar det metoder som Get, Post och Delete. En fördel med att använda Swagger är att man på ett enkelt och överskådligt sätt kan testa sina endpoints och se vad dom returnerar i form av data, svar, headers eller statuskod, utöver detta är det också ett bra sätt att se vad backend:en innehar och vad som behövs adderas, framför allt under utvecklingsfas.

### Category

GET

/api/Category/GetAllCategories

- Hämtar en lista med alla tillgängliga kategorier.

### Receipt

POST

/api/Receipt/CreateReceipt

- Skapar ett nytt kvitto med information som skickats ifrån klienten.

DELETE

/api/Receipt/{id}

- Tar bort ett specifikt kvitto med hjälp av ReceiptId

GET

/api/Receipt/GetUserReceipts

- Hämtar alla kvitton som tillhör användaren.

GET

/api/Receipt/GetMostPurchasedItem

GET

/api/Receipt  
/GetMostPurchasedCategory

- Hämtar det item och den kategorin som registrerats mest för en användare mellan två specifika datum.

## User

POST

/api/User/Register

- Registrerar en ny användare med information Username, Email och Password.

POST

/api/User/Login

- Loggar in användaren och returnerar en JWT-key för autentisering.

### 4.3 Entity Framework Core

EFC är ett ORM (Object-Relational mapper) för .NET och kopplar ihop klasser som skapas i C# med tabeller som finns i databasen. Genom att använda sig utav EFC så kan jag kommunicera med databasen via objekt och LINQ-frågor istället för att skriva SQL-kod manuellt i mina anrop. Bland annat så förenklar det:

- Skapa och ändra hur databasen ska se ut med hjälp av migrations.
- Spara, ändra, hämta, ta bort data med hjälp av enkla anrop.
- Bygga en bra kodbas med tydlig kodstruktur.

Detta gör att kodbasen blir enklare att underhålla och utveckla samt lättare skalbar i framtiden.

### 4.4 Databas

Utöver UserIdentity så består databasen utav Category, Item och Receipt-entiteter.

- Category representerar produktens kategori, exempelvis mat, resor, nöje osv. Varje kategori kan ha flera produkter kopplade till sig vilket då utgör en en-till-många relation mellan Category och Item.
- Item är varje enskild produkt som finns på kvittot. Hur mycket den produkten kostar och vilken kategori den tillhör. Exempelvis iPhone - 9990kr, HP datorskärm 24" – 1690kr osv.
- Receipt representerar kvittot som innehar totalbeloppet av alla produkter på kvittot, var och när inköpet har gjorts samt utav vilken användare. Ett kvitto kan innehålla flera items vilket gör det till en en-till-många relation även här. Exempelvis Coop 05-05-2025 Apple x2 4kr totalbelopp 8kr inköpt av Joakim.



## 4.5 Autentisering och säkerhet

iXpense autentiseras med hjälp utav ASP.NET Identity samt nyttjar JSON Web Tokens (JWT) nycklar. Detta gör så att backendens endpoints är lättare att skydda och kan då säkerhetsställa att anrops som görs av användare som är godkända av systemet får tillgång till dessa eller vissa endpoints medan den nekar användare som inte har giltig tillgång. Kort och gott, den ser till att användaren är autentiserad för en specifik endpoint eller inte.

*Flödet ser ut såhär för en ny användare:*

Användaren gör anrop mot Register och skickar med information kring email, användarnamn och lösenord. Om informationen är giltig skapas kontot och användaren kan då logga in. Vid en lyckad inloggning genereras en JWT-nyckel som då skickas tillbaka till klienten.

JWT-nyckeln innehåller information om användaren, bland annat UserId och ett TokenId. Nyckeln innehar en digital signatur som hämtas ifrån JwtConfig:Secret som lagras i appsettings.json och har en utgångstid på ett dygn efter att den skapats. En token valideras varje gång och vid anrop mot skyddade endpoints måste en giltig JWT finnas med i anropet för att anropet ska kunna lyckas.

I frontend-delen av projektet (iXpense-client) går all kommunikation mellan backend och frontend via en ApiClient-komponent som använder Axios som har en baseUrl mot backend. En request interceptor används för att hämta token ifrån localStorage och adderar den sedan till Authorization.

ApiClient importeras sedan till andra mer specifika anrop exempelvis: ApiUser som håller inloggning och registrering eller ApiReceipt som håller addReceipt och getReceipts. Dessa specificerar vad som ska skickas till servern.

Att dela upp lösningen på detta sett gör det mer överskådligt och lättare att kontrollera sina anrop.

## 5 Testning och validering

För att kunna säkerhetsställa att applikation och funktionalitet fungerar som förväntat har jag använt mig utav manuelltestning i både backend och frontend projekten. Detta var väsentligt för att få ut information om:

- Funktionlitet – gör den specifika funktionliteten det den ska?
- Säkerheten – Fungerar autentiseringen korrekt?
- Flöde – får jag rätt statuskoder när jag förväntar mig det?

## 5.1 Backend

Exempelvis via Swagger om en användare försöker skapa ett konto som redan existerar:

```
{
  "username": "Joakim",
  "email": "Joakim@gmail.com",
  "password": "Joakim123!"
}'
```

### Request URL

```
https://localhost:7183/api/User/Register
```

### Server response

Code	Details
500	Error: response status is 500

*Undocumented*

### Response body

```
{
  "status": "Error",
  "message": "User already exists"
}
```

Eller när man loggar in:

```
-d {  
  "username": "Joakim",  
  "password": "Joakim123!"  
}'
```

Request URL

```
https://localhost:7183/api/User/Login
```

Server response

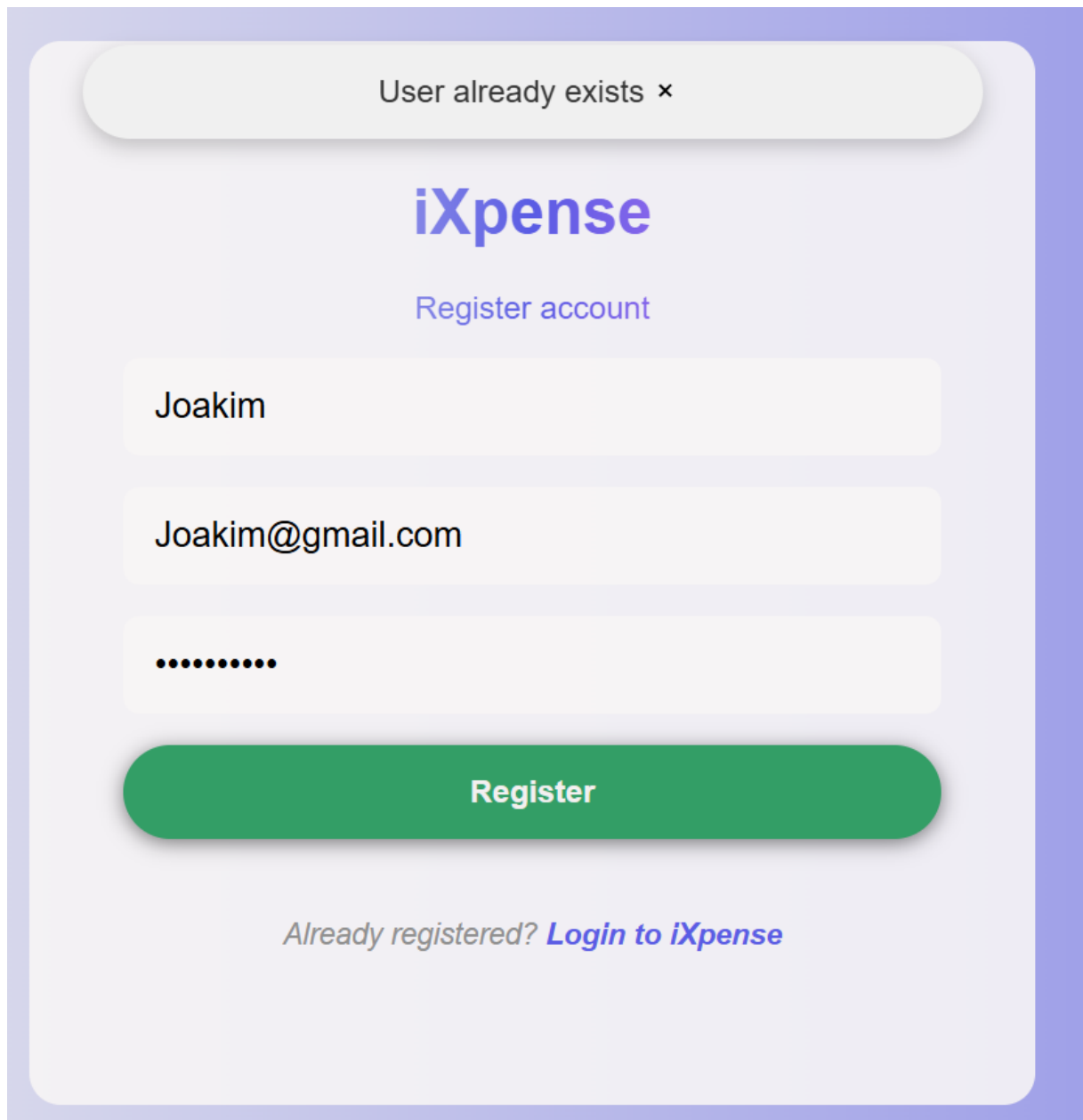
Code	Details
200	<p>Response body</p> <pre>{   "status": "Success",   "message": "Login successful",   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIzYmNiIiwiaWF0IjE5OTYyMDF9.y52K40ZtZtGHgJQJOvmu7JENP4jrTkjbF"</pre>

## 5.2 Frontend

Vid inloggning i klienten lagras detta i localStorage:

Origin		https://localhost:5173
Key	Value	
token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJz...	
userId	3bca6173-b2db-456f-8ebb-7e9411da5d99	
username	Joakim	

Motsvarande test i klient som i backend om man försöker registrera en användare som redan finns:

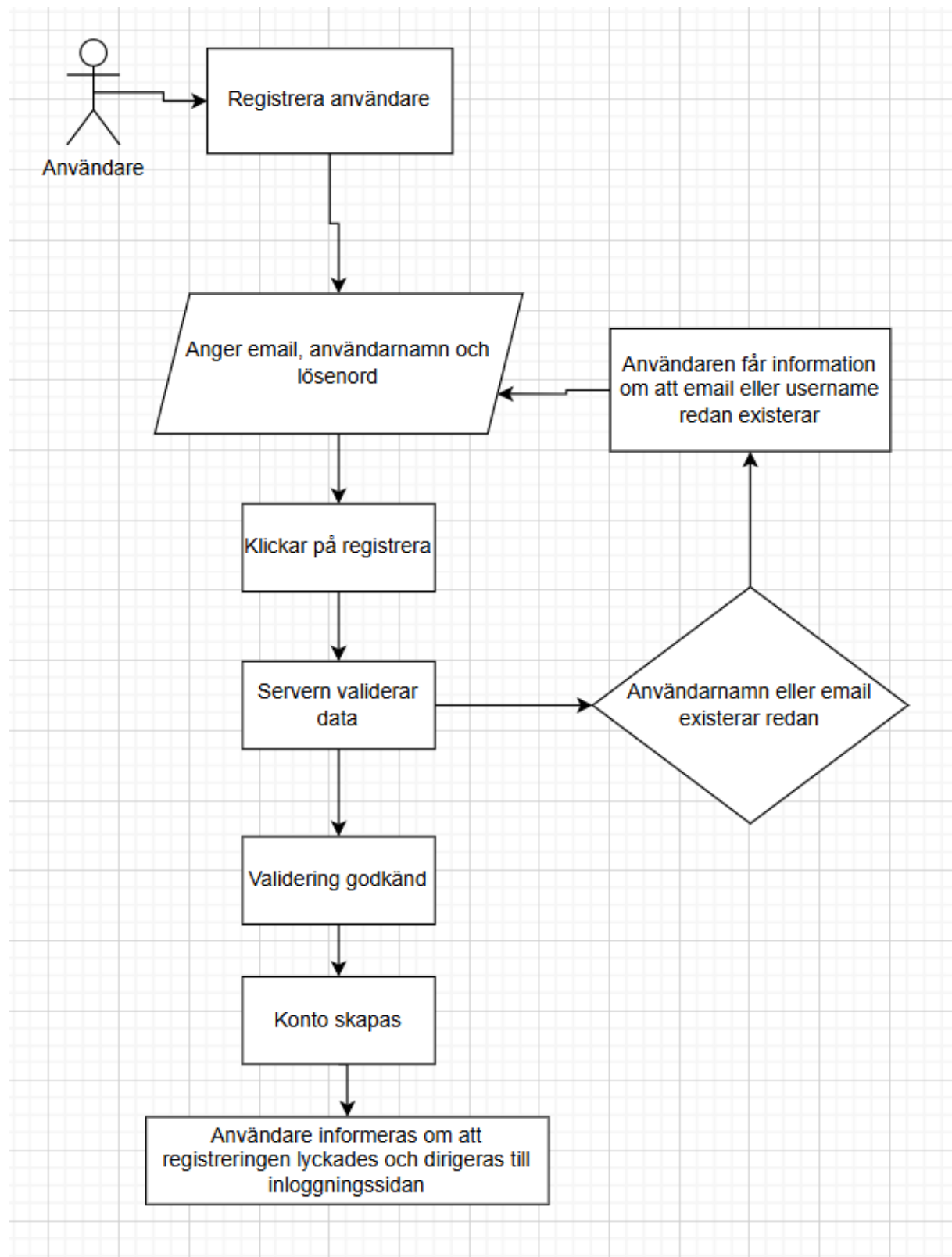


The screenshot shows a web form for registering an account on iXpense. At the top, a light gray error message box with a close icon (x) displays the text "User already exists". Below this, the iXpense logo is centered, followed by the text "Register account". The form contains three input fields: the first contains "Joakim", the second contains "Joakim@gmail.com", and the third is a password field represented by ten dots. A large green button with the text "Register" is positioned below the input fields. At the bottom of the form, there is a link that reads "Already registered? [Login to iXpense](#)".

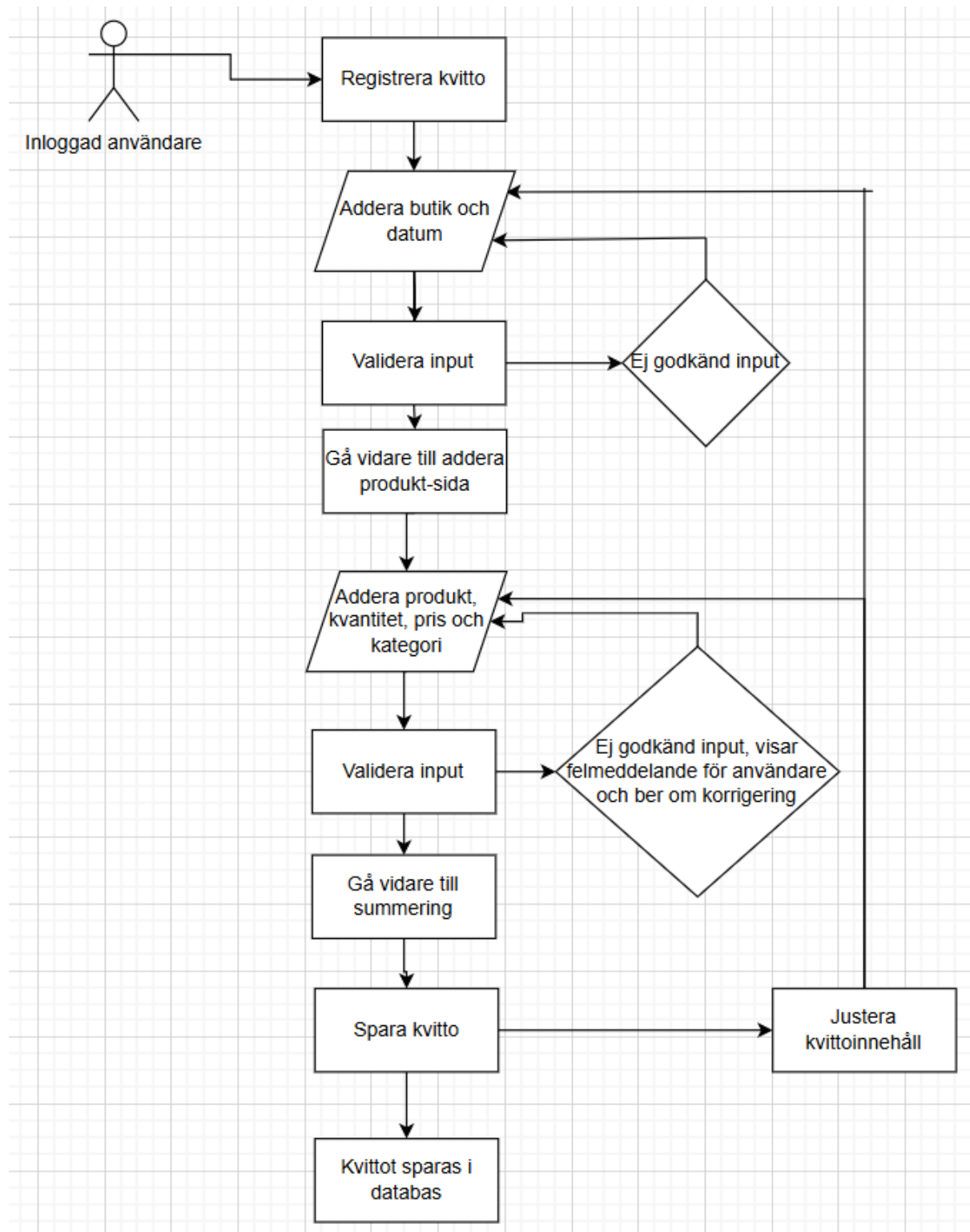
Testning genomfördes kontinuerligt under båda projektens gång och bidrog till att jag tidigt kunde upptäcka fel och åtgärda dessa samt att jag genom att göra det kunde minimera risken till att samma fel upprepades.

## 6 Flödesdiagram

Visar det logiska flödet när en ny användare skapar ett konto. Felhantering finns inbyggt om en email eller användarnamn redan finns spara i databasen.



Visar flödet då en befintlig användare registrerar ett kvitto. Felhantering för tomma inputs eller felaktiga val finns och användaren har möjlighet att gå tillbaka till tidigare steg utan att vald eller adderat data försvinner.



## 7 GitHub

Under projektets gång har jag använt versionshantering för både Backend och Frontend. Det kändes logiskt att köra två separata repon just för att ha de olika lösningar uppdelade för tydligare översikt. I framtiden skulle dessa två kunna befinna sig i ett gemensamt huvudrepo men fortfarande vara uppdelade inom detta. Jag har så gott som 100% jobbat med branch-hantering och inte haft några problem med konflikter under merge så detta har fungerat väldigt bra.

### 7.1 iXpenseBackend

Backend-projektet i ASPNET-Core C#

### 7.2 iXpenseClient

Frontend-projektet skrivet i React.

## 8 Tänkta saker att implementera men som inte var genomförbart pga. tidsbrist

- Fullständig responsiv-design via breakpoints.
- Hantering av kvitto via telefon där man tar en bild och skickar in som sedan tolkas utav applikation och sparas korrekt.
- Användartips på förbättring baserat på tidigare spendering.

## 9 Avslutning och reflektion

### 9.1 Vad gick bra?

- Under min lra jobbade jag inom React och satt med i Frontend teamet, så ett projekt med stor fokus på både Backend och Frontend kändes som självklart. Jag tycker att jag har kombinerat dessa på ett bra sätt och lyckats med att få med kvalitativt innehåll inom båda dessa repositories baserat på den relativt korta tidsperiod som man skulle förhålla sig till.
- Versionshanteringen i GitHub fungerade väldigt bra.

### 9.2 - Utmaningar/svårigheter

- Återigen kopplat till den givna tiden med att försöka att få allt så enhetligt som möjligt. En blandning mellan att hitta en fungerande lösning som inte tummar för mycket på kvalitén.

### **9.3 - Kunde jag gjort något annorlunda?**

- Jag kunde försökt att förbereda research inom bar-codes och hur man kan gå tillväga för att använda sig utav mobilen och få ut information ifrån en bild. Det hade bland annat krävt att projektet skulle host:as på en server.
- Fokuserat mer på att göra applikationen responsiv i sin design.

### **9.4 - Egenskaper jag utvecklat under projektet**

- Större självsäkerhet inom utveckling mellan backend och frontend som leder till större självförtroende inom ens projekt-utvecklande.
- ReCharts är inget jag tidigare använt mig utav men det uppfattande jag som ett rätt så lätt bibliotek att använda. I framtiden kommer ett utav fokusområdena att bli att försöka skapa diagram/visuella representationer på ett mer hållbart sätt för större lösningar.