Webbaserad multiplayer quiz spel byggd på JavaScript och Socket.IO

Atte Hätinen

Examensarbete

Informationsteknik

2022

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informationsteknik
Identifikationsnummer:	8553
Författare:	
Arbetets namn:	Webbaserad multiplayer quiz spel byggd på JavaScript och Socket.IO
Handledare (Arcada):	Andrey Shcherbakov
Uppdragsgivare:	

Sammandrag:
Socket.IO är ett JavaScript bibliotek som hanterar realtidskommunikation mellan en klient och en server. Socket.IO använder WebSocket-protokollet för kommunikationen. Detta arbete är dedikerat för att bygga ett multiplayer webbaserat quiz spel som använder sig av Socket.IO för att kommunicera spellogiken från klienterna till servern. I mjukvaruutvecklingsprocessen användes metodologin *Rapid application development*. Övriga verktyg som behövdes för att utveckla med Socket.IO var: JavaScript, Node.js, HTML, CSS, Nodemon och Visual Studio Code. Mjukvaruutvecklingsprocessen började med att planera basfunktionaliteten som ett quiz spel kräver. Första milstolpen i mjukvaruutvecklingsprocessen var en lokal prototyp. Efter lokala prototypen utvecklades spelet vidare att kommunicera med en server och andra spelare. Sista fasen av mjukvaruutvecklingsprocessen var att utveckla på layout, design och extra funktionalitet. Resultatet blev ett spel som använder sig av Socket.IO för att kommunicera i realtid mellan servern och klienten. Spellogiken skickas in från klienterna och servern distribuerar det åt övriga klienter. Resultatet visar att Socket.IO är ett lämpligt verktyg för att utveckla program som använder sig av realtidskommunikation över internet. Socket.IO hade inga direkt identifierbara begränsningar. Enligt tidigare forskning i ämnet uppstår begränsningar i Socket.IO först i program som har en större skala av användning och Socket.IO är mest lämpligt för program som har ett begränsat antal användare och endast en server vilket håller överens med detta arbete. Fokuset av arbetet var att skapa ett program som använder sig av simpel spellogik vilket ledde till att arbetet inte utforskar en stor del av funktionaliteten som Socket.IO har att erbjuda men fungerar bra som en bas för vidareutveckling.

Nyckelord:	Mjukvaruutveckling, Socket.IO, WebSocket, Node.js, JavaScript-bibliotek, Realtidskommunikation, Quiz, Multiplayer, Front-end, Back-end
Sidantal:	34
Språk:	Svenska
Datum för godkännande:	2.5.2022

DEGREE THESIS	
Arcada	
Degree Programme:	Information technology
Identification number:	8553
Author:	
Title:	Web-based multiplayer quiz game built with JavaScript and Socket.IO
Supervisor (Arcada):	Andrey Shcherbakov
Commissioned by:	

Abstract:
Socket.IO is a JavaScript library that handles real-time communication between a client and a server. Socket.IO is using the WebSocket protocol for communication. This work is dedicated to building a multiplayer web-based quiz game that uses Socket.IO to communicate the game logic from the clients to the server. The methodology *Rapid application development* was used in the software development process. Other tools needed to develop with Socket.IO were JavaScript, Node.js, HTML, CSS, Nodemon and Visual Studio Code. The software development process began with a planning phase, which consisted of planning the basic functionality that a quiz game requires. The first milestone in the software development process was a local prototype. Following the local prototype, the game was further developed to communicate with a server and other players. The last phase of the software development process was to develop the layout, design, and extra functionality. The result was a game that uses Socket.IO to communicate in real time between the server and the client. The game logic is sent in from the clients and the server distributes it to all other clients. The results show that Socket.IO is a suitable tool for developing applications that use real-time communication over the internet. Socket.IO had no directly identifiable restrictions. According to previous studies restrictions in Socket.IO only occur in programs that have a larger scale of use and Socket.IO is best suited for programs that have a limited number of users and only one server, which agrees with the results of this work. The focus of the work was to create a program that uses simple game logic, which led to the work not exploring a large part of the functionality that Socket.IO has to offer but works well as a base for further development.

Keywords:	Software development, Socket.IO, WebSocket, Node.js, JavaScript-library, Real-time communication, Quiz, Multiplayer, Frontend, Backend
Number of pages:	34
Language:	Swedish
Date of acceptance:	2.5.2022

INNEHÅLLSFÖRTECKNING

KODEXEMPEL

FIGURER

FÖRKORTNINGAR

HTTP - Hypertext Transfer Protocol

AJAX - Asynchronous JavaScript and XML

CSS - Cascading Style Sheets

HTML - Hypertext Markup Language

NPM - Node Package Manager

Opentdb - Open Trivia Database

JSON - JavaScript Object Notation

API - Application programming interface

FÖRORD

Detta arbete har skapats av eget intresse i WebSocket-teknologin. Under mina år av programmering samt under min utbildning inom informationsteknik har jag inte stött på WebSocket-teknologin. Jag hade före detta arbete endast arbetat med endast HTTP vilket gav mig ett intresse hur man implementerar WebSocket-protokollet. Idén att använda Socket.IO biblioteket var rentav slumpmässig, Jag hittade det en kort tid före jag började planera arbetet och det verkade vara intressant. Jag har förr använt ett flertal av bibliotek som finns i Node Package Manager och har alltid varit imponerad hur lätta de är att installera och använda. Socket.IO är inget undantag och det har varit ett nöje att arbeta med denna teknologi.

Helsingfors 27.4.2022

Atte Hätinen

1 INLEDNING

Ett sätt att minska på latens i webbkommunikation är att använda WebSocket-protokollet i stället för till exempel det mera traditionella http. WebSocket-protokollet är ett dubbelriktat och händelsedrivet nätverksprotokoll. WebSocket protokoll är generellt ett bättre val för kommunikation som sker i realtid efter som den är händelsedriven, vilket betyder att dataöverföring sker direkt då det är tillgängligt till skillnad från HTTP-protokollet som förutsätter att man gör en begäran för uppdateringar vid konstanta mellanrum. (Parker Software, 2022)

WebSocket-protokollet har vidareutvecklats till flera bibliotek vilka bygger sina egna tilläggsfunktioner på WebSocket-protokollet vilket gör den alltmer effektiv för att utveckla program som behöver webbkommunikation.

1.1 Bakgrund

Webbkommunikationsteknologin har utvecklats kontinuerligt sedan början av internet. Dagens webbkommunikation består av flera nätverksprotokoll som har olika egenskaper och användningsområden. I internet-utvecklingens tidiga år använde man sig främst av HTTP-protokollet för att kommunicera över internet. HTTP-protokollet är uppbyggd av fyra steg. Det första steget är att öppna en förbindelse vartefter man beskriver i en *HTTP request* vilken information man begär från en server. Efter detta får man svar på begäran och till sist stängs förbindelsen. (Kozierok, 2005)

En stor utveckling som behövdes för att förbättra funktionalitet för moderna webbsidor var introduktionen av konceptet AJAX i början av 2000-talet. AJAX är en kollektion av teknologier som möjliggör modifiering av en webbsida med att hämta data från en server utan att ladda om hela sidan, vilket var en begränsande faktor i dåtida webbsidor. AJAX konceptet löste många problem och AJAX teknologier utvecklas även idag. (IBM, 2021)

I tidiga skeden av internets evolution var utvecklingens mål att försöka förbättra kommunikationen mellan människor för officiella ändamål det vill säga främst för militäriska, vetenskapliga och statliga ändamål. Numera används majoriteten av internets bandbredd för nöjets skull i stället för endast kommunikation mellan personer och organisationer. Sandvines Rapport *The Global Internet Phenomena Report January 2022* visar att alltmer av internetanvändning är för nöjets skull. Till exempel sociala medier och videon kombinerat tar upp över tolv procent av totala internettrafiken. Enligt rapporten tar videospel över fem procent av totala internettrafiken, vilket är en indikation på att allt fler spel spelas online. (Sandvine, 2022)

Onlinespel behöver en låg latens för att inte kvaliteten av spelet ska lida. Latens är ett mått mätt i millisekunder på tiden det tar för data att överföras från en server till en klient och vice versa. Under 100 millisekunders latens anses vara acceptabelt medan mellan 20 och 40 millisekunders latens anses vara optimal. (Screenbeam, 2022).

AJAX teknologier kan vara för långsamma att kommunicera mellan klienten och servern i ett videospel. År 2011 standardiserades WebSocket-protokollet. WebSocket-protokollets största utveckling är att förbindelsen mellan klienten och servern hålls öppen efter första förbindelsen vilket möjliggör en konstant ström av data mellan klienten och servern det vill säga WebSocket-teknologin är fullduplex. Full duplexkommunikation kan vara ett sätt att minska på latens i kommunikation över internet. (Parker Software, 2022)

Numera finns det även gratis programbibliotek som använder sig av WebSocket-protokollet, vilket gör det nybörjarvänligt att börja och arbeta med teknologin. Programbiblioteken kan underlätta arbetet signifikant för en utvecklare och de kan möjliggöra funktionalitet som inte finns i själva WebSocket-protokollet. Av dessa är de tre mest populäraste: ws med över 57 miljoner nerladdningar per vecka, sockjs med över elva miljoner nerladdningar per vecka och Socket.IO med över fyra miljoner nerladdningar i veckan. (npm Inc., 2022)

1.2 Syfte, mål och forskningsfråga

Syftet med arbetet är att ta reda på om Socket.IO lämpar sig som ett verktyg för att utveckla ett multiplayer spel som har realtidskommunikation mellan klienten och servern.

Målet med detta arbete är att utveckla ett webbaserat multiplayer quiz spel som använder Socket.IO för att kommunicera mellan klienten och servern. Spelet ska kommunicera i realtid mellan spelarna och spellogiken överförs mellan spelare via Socket.IO.

Ett lämpligt spel för detta arbete är ett quiz spel. Ett quiz spel har tillräcklig funktionalitet för att dra nytta av Socket.IO medan spellogiken inte är för komplex Till exempel när man startar spelet, ska spelet starta till alla spelare samtidigt. Spelet måste på klientsidan ta reda på om spelaren svarade rätt eller fel, och skicka det till servern och därifrån ska servern skicka resultaten åt alla. Tiden som spelare har på sig att svara på frågorna måste vara samma för alla och spelet ska sluta samtidigt för alla och det ska Socket.IO hantera.

Forskningsfråga:
Hur lämpar sig Socket.IO för att utveckla ett multiplayer spel som har realtidskommunikation mellan klienten och servern?

Arbetets hypotes är att Socket.IO är ett lämpligt verktyg för att utveckla ett multiplayer spel som använder realtidskommunikation. Hypotesen grundar sig på att Socket.IO är ett JavaScript-bibliotek som möjliggör dubbelriktad händelsedriven kommunikation med låg latens mellan en klient och en server. (Socket.IO, 2022)

1.3 Avgränsning

Arbetet avgränsas till endast enkel spellogik med motiveringen att fokuset i arbetet ligger i att svara på frågan om Socket.IO är ett lämpligt verktyg för ett spel som har

realtidskommunikation mellan klienten och servern. Det är en risk att programmera komplex spellogik för att det kan orsaka oförväntade problem i utvecklingsprocessen vilket kan leda till ett sämre resultat. Om spellogiken är enkel finns det en låg risk att stöta på problem i utvecklingsprocessen vilket ökar på sannolikheten att få ett bra resultat av arbetet.

2 METOD

I metodkapitlet beskrivs tillgängliga mjukvaruutvecklingsmetoder. Därefter väljs den metoden som passar bäst för detta arbete. Efter metoden är vald planeras funktionalitetskraven för spelet samt vilka teknologier som behövs för att uppnå kraven.

2.1 Tillgängliga mjukvaruutvecklingsmetoderna

Agile är en metodik inom mjukvaruutveckling som innehåller flera liknande metoder. Namnet *Agile* kommer från engelskan och betyder lättrörlig. Lättrörlighet i detta sammanhang betyder att lätt kunna ändra på och iterativt utveckla mjukvaran under utvecklingsprocessen. Utvecklingen i en *Agile* metod görs i korta etapper med diverse längd som kallas för sprintar. En sprint börjar med en sprintplanering där man planerar alla uppgifter som en utvecklare ska försöka hinna göra under en sprint. Under sprinten har man dagliga möten där man reflekterar över utvecklingen man arbetar med och rapporterar sin status med utvecklingen. Avslutningsfasen går ut på att presentera en demo över vad man har utvecklat samt reflektioner över hela sprinten. Agila metoder möjliggör iterativ släppning av mjukvara vilket är ett effektivt sätt att hitta defekter och ytterligare förbättringsidéer. Agila metoder är beroende av utmärkt kommunikation mellan utvecklare i ett team vilket kan orsaka problem med nybörjare eller utvecklare som inte jobbar effektivt i ett team. (Laudon, 2022)

Vattenfallsmetoden anses vara den mest traditionella mjukvaruutvecklingsmetoden. Vattenfallsmetoden är en linjär modell som består av fem sekventiella faser: krav, design, implementation, verifikation och underhåll. Varje fas måste vara komplett före man flyttar sig till nästa fas och man går sällan tillbaka till en tidigare fas. Metoden är lätt att arbeta med för projekt som har stabila och klara krav men eftersom

utvecklingsprocessen är linjär kan den vara långsam ifall kraven på utvecklingen ändras under utvecklingen (Synopsys Editorial Team, 2022).

*Rapid application development (*RAD) är en mjukvaruutvecklingsmetod som fokuserar på snabb och iterativ utveckling med en kort planeringsfas. Metoden består av fyra faser. Första fasen är att definiera kraven på projektet. Andra och tredje fasen är att bygga en prototyp samt att absorbera feedback. Dessa faser upprepas tills prototypen har mött alla krav. I sista fasen slutför man projektet och optimerar projektets stabilitet och underhållbarhet. (Outsystems, 2022)

2.2 Valet av metod

RAD metoden är valet för detta arbete. RAD metoden passar detta arbete bäst eftersom det viktigaste i ett quiz spel är basfunktionaliteten av spelet och basfunktionaliteten är det enda kritiska som behövs planeras i förväg. I design, layout och funktionalitet utanför basfunktionaliteten påverkar inte utvecklingen med Socket.IO, utan kan väljas iterativt under utvecklingens lopp. Vattenfallsmetoden går ut på att ha en strikt plan om design och implementation i början vilket inte passar detta arbete. Första steget i utvecklingen är att skapa en prototyp som har funktionaliteten av ett quiz spel utan fokus på utseendet. RAD metoden passar även en ensam utvecklare, vilket en Agil metod inte passar eftersom agila metoder går ut på att ha ett team och ofta en kund som ger feedback.

2.3 Krav för ett quiz spel

Ett krav på quiz spelet är multiplayer aspektet av spelet. Spelare ska kunna ansluta sig till en spelserver över internet på en webbläsare. Eftersom en quiz är en tävling måste spelet ha funktionalitet för att räkna poäng på basis av rätt eller fel svar och visa poängställningen efter varje runda av frågor. Tävlingsnaturen av en quiz kräver en begränsning på svarstid. Detta innebär att spelet måste ha en klocka som räknar ner tiden man har att svara på en fråga.

Ett grundkrav för i ett quiz spel är en stor mängd frågor. Spelet måste ha tillgång till många frågor för att det ska vara användbart. Ett quiz spel blir snabbt ointressant om samma frågor repeteras varje spelomgång. Eftersom det blir för mycket arbete för spelare att skriva in frågorna för hand varje spelomgång måste frågorna hämtas från en utomstående API.

2.4 Val av teknologier

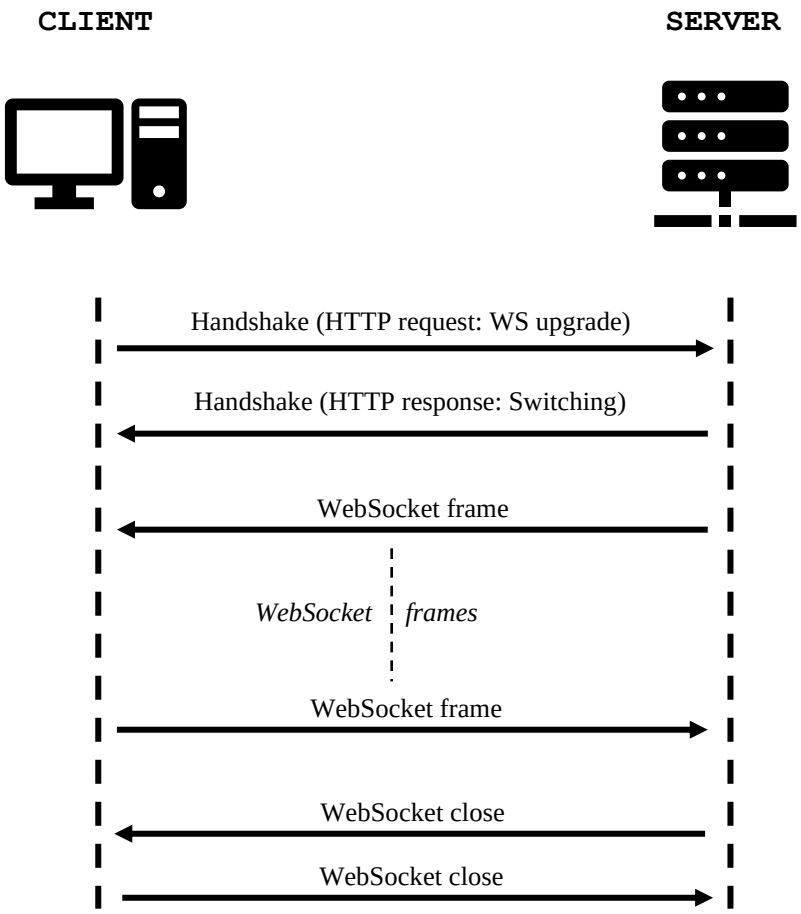
I utvecklingsprocessen används JavaScript-biblioteket Socket.IO som möjliggör kommunikationen mellan klienten och servern. JavaScript är programmeringsspråket som används för att implementera Socket.IO samt för att manipulera DOM och programmera spellogiken. Node.js en plattform som möjliggör exekvering av JavaScript kod på servernivån. HTML och CSS används för att ändra på utseendet av spelet. Övriga teknologier som används är Nodemon som är ett tilläggsverktyg för att automatiskt starta om en Node.js process samt Visual Studio Code som är en kodredigerare som själva programmeringen av spelet görs i.

2.4.1 Socket.IO

Socket.IO är ett bibliotek som möjliggör kommunikation mellan en klient och en server. Socket.IO är dubbelriktad och händelsebaserad samt har låg latens. Socket.IO är byggt på WebSocket-protokollet. Det finns flera implementationer av Socket.IO. I detta arbete implementeras både server och klientsidan med JavaScript. Förutom JavaScript kan man implementera serversidan av Socket.IO med Java och Python samt klientsidan med de flesta största programmeringsspråken.

Socket.IO använder sig av två transportsätt för att transportera nätverkspaket, WebSocket och *HTTP long-polling.* Socket.IO förbindelsen börjar med en *handshake* via en *HTTP GET request* från klienten till en server. Efter att klienten har fått svaret tillbaka från servern uppgraderas förbindelsen till en WebSocket-förbindelse. Förbindelsen kan stängas endera av klienten eller servern. Socket.IO har en *fallback* funktion vilket betyder att ifall en WebSocket anslutning slutar att fungera av någon orsak kan Socket.IO automatiskt falla tillbaka till en *HTTP long-polling* anslutning.

HTTP long-polling består av många successiva *GET* och *POST requests* till servern för att motta data och sända data. Socket.IO kan även återansluta sig automatiskt ifall en koppling oväntat upphör. Figuren nedan illustrerar förbindelsen i WebSocket-protokollet.



Figur 1 WebSocket anslutning

Fast Socket.IO är byggd på WebSocket-protokollet kan man inte ansluta sig till en WebSocket med Socket.IO. Implementationen av Socket.IO och WebSocket ser liknande ut men Socket.IO tillägger metadata i transporten av nätverkspaket vilket orsakar inkompabiliteten.

Andra egenskaper som Socket IO har är funktionalitet för en buffert, vilket betyder att om klienten tappar anslutningen till servern kan servern hålla en buffert av data som

klienten får efter återanslutning. Denna funktionalitet är viktigt för användarupplevelsen i ett spel.

Socket.IO har även funktionalitet för ”rum” vilket WebSocket inte har ursprungligen. Rum i Socket.IO är kanaler som användare kan ansluta sig till och lämna. Målet med rummen är möjligheten att skicka data till specifika rum där datan når endast användarna i det rummet. Rum-funktionalitet kan vara viktigt för till exempel ett spel som har flera instanser av spelet i gång samtidigt.

Broadcasting funktionalitet ger möjligheten att skicka data till alla klienter som är anslutna på samma gång. En annan viktig del av *broadcast*-funktionaliteten är att en klient kan skicka data till alla andra klienter utom sig själv. (Socket.IO, 2022)

```
io.to("spelrum 1").emit("Kalle leder!");
io.to("spelrum 2").emit("Spelet har avslutat");
```

Kodexempel 1 Socket.IO funktionalitet för rum

```
io.emit("hej", "Hej alla på servern!");
```

Kodexempel 2 Emit-händelsen i Socket.IO

Första steget för att skapa ett Socket.IO program är att importera serven från Socket.IO biblioteket med en *import statement*. En server skapar man med att använda *new*-operatorn som skapar en instans av objektet Server. Portnumret ges som en parameter till Server objektet.

Grunden av all funktionalitet i Socket.IO består av händelserna *emit* och *on*. I exemplet nedan skickar vi från servern ett meddelande med *emit*-händelsen och nämner paketet *hello from server*. På klientsidan binder vi paketets namn i *on*-händelsen vilket möjliggör att meddelandet hittar fram. *Emit*-händelsen skickar in data, medan *on*-händelsen lyssnar för data som skickas av *emit*-händelsen. Att skicka data åt det motsatta hållet, från klienten till servern fungerar lika.

Två kodexempel nedan illustrerar hur man implementerar kommunikationsfunktioner för Socket.IO i JavaScript-kod.

```
Import { Server } from "socket.io";
const io = new Server(3000);
io.on("connection", (socket) => {
  socket.emit("hello from server");
  socket.on("hello from client", (message) => {
    console.log(message);
  });
});
```

Kodexempel 3 Serversidan

```
Import { io } from ”socket.io-client”;
const socket = io("ws://localhost:3000");
socket.emit("hello from client");

socket.on("hello from server", (message) => {
  console.log(message);
});
```

Kodexempel 4 Klientsidan

Socket.IO valdes som ett av verktygen för detta arbete eftersom biblioteket är lätt installerat från NPM och dokumentationen för biblioteket är utmärkt. Förutom detta har Socket.IO följande fördelar över vanliga program med endast WebSocket-protokollet utan ett Socket.IO. Socket.IO fokuserar sig på användarvänlighet och pålitlighet över serverprestanda, vilket passar bra för ett litet spel. Socket.IO har ytterligare inbyggt en *fallback* funktion över en traditionell WebSocket implementation vilket betyder att anslutningen faller tillbaka till traditionella *HTTP polling* ifall WebSocket anslutningen avkopplas av för någon orsak (Socket.IO, 2022).

2.4.2 JavaScript

JavaScript är ett programmeringsspråk som kan exekvera komplexa funktioner på en webbsida på klient-sidan. Typiska ändamål för JavaScript är att manipulera dokumentobjektmodellen (DOM) av en webbsida det vill säga modifiera utseendet av webbsidan. JavaScript används av över 97 % av dagens webbsidor för front-end funktionalitet JavaScript kan även drivas på serversidan med hjälp av andra teknologier som till exempel Node.js. (W3techs, 2022) JavaScript var ett klart val för detta projekt

eftersom den är mest använd och bäst dokumenterad samt JavaScript behövs för att använda Socket.IO med en Node.js server (Mdn, 2022).

2.4.3 Node.js

Node.js är en back-end omgivning som möjliggör exekvering av JavaScript program på en server. Mjukvaran är gratis och är kompatibel med diverse plattformar. Motivering för att använda Node.JS plattformen är att serverimplementationen av Socket.IO kräver Node.JS för att exekvera JavaScript kod på servernivå. Förutom det är Installering av Socket.IO lätt med Node.js inbyggda pakethanterare Node Package Manager.

2.4.4 Övriga *front-end* teknologier

Förutom JavaScript som är en av byggstenarna av klientsidan av webben behövs även de två andra byggstenarna, HTML och CSS för att skapa helhetliga webbsidor.

HTML är ett märkspråk vilket har är av grundstenarna i *front-end* utveckling. HTML har som uppgift att ge strukturen för en webbsida. HTML bygger upp innehållet av en webbsida med element. Element kan vara till exempel rubriker, listor och knappar.

CSS är ett stilspråk som man använder för att ändra på stilen av HTML element på en webbsida. Utan CSS är HTML element begränsade endast till webbläsarens standardstil för elementen. Viktiga funktioner av CSS är till exempel att byta färg och position av HTML element på en webbsida.

2.4.5 Nodemon

Nodemon är ett verktyg i NPM som hjälper utveckling av Node.js applikationer med att automatiskt upptäcka förändringar i projektfilerna och automatiskt starta om Node.js servern, vilket utvecklare annars måste genomföra för hand varje gång man gör en förändring i en fil (npm Inc, 2022).

2.4.6 Visual Studio Code

Visual Studio Code är kodeditorn som användes för detta arbete. Visual Studio Code är en programvara med öppen källkod och den har många verktyg för att hjälpa utvecklare som till exempel *Intellisense*-kodkomplettering, parametertips och underlättad navigering i projektfilerna med mera (Microsoft, 2022).

3 RESULTATREDOVISNING

I kapitlet beskrivs utvecklingsprocessen med *Rapid application development* metodologin samt resultatet av utvecklingen presenteras och analyseras.

3.1 Utvecklingsprocessen

Utvecklingsprocessen enligt RAD-metoden består av fyra steg. Första steget handlar om att definiera krav. Eftersom en av nyckelprinciperna i RAD-metoden är snabb utveckling och endast liten fokus på planering planeras först en prototyp med lösa krav av basfunktionalitet som ett quiz spel behöver. Andra fasen handlar om att utveckla en prototyp som har basfunktionalitet av ett spel men är utseendemässigt samt funktionalitetsmässigt inte färdig. Tredje fasen är att absorbera feedback det vill säga planering och tillägg av funktionalitet och design på basis av prototypen. Sista fasen är att slutföra produkten. Till sista fasen hör att göra sista optimeringen av funktionalitet och slutföra design och layout av spelet.

3.1.1 Planeringsfasen

Planeringsfasen bestod av att planera basfunktionalitet som ett quiz spel behöver och göra en skiss över hur layouten av ett quiz spel ska se ut. Första identifierbara milstolpen i utvecklingen var att färdiggöra en prototyp av spelet som saknar visuell utveckling men har de viktigaste funktionaliteten som ett quiz spel ska ha.

En fas i planeringen var att planera hur spelet ska få in data för frågorna. Det skulle vara för mycket arbete för spelare att mata in sina frågor varje gång man börjar ett nytt spel. Ett bättre alternativ är att använda en API som hämtar frågor från en databas. Till denna

funktionalitet finns Open Trivia Database som är en gratis, användarbidragen service för att hämta frågor från en databas utan en API nyckel. Opentdb har över 4000 frågor att välja mellan. Opentdb har även filtreringsfunktion på basis av kategori, svårighetsgrad och typ av fråga dvs. flervalsfrågor eller sant/falskt frågor (PixelTail Games, 2022).

3.1.2 Prototyp

Filstrukturen skapades först i Visual Studio Code. Klientsidan av projektet har JavaScript filer, CSS filer och html filer, medan serversidan har en JavaScript fil som Node.js kör. Node.js skapar även en *node_modules* mapp som innehåller alla verktyg och bibliotek som man installerar med NPM, i detta fall Socket.IO.

Före utvecklingen av prototypen kan börjas måste Socket.IO installeras. Socket.IO installeras med Node.js inbyggda pakethanterare Node Package Manager. Man behöver endast ge ett *install* terminalkommando i terminalen och NPM installerar paketet och tillägger det i projektfilerna.

```
npm install socket.io
```

Kodexempel 5 Terminal-kommando för att installera Socket.IO

Utvecklingen av klientsidan började med en knapp som exekverar en HTTP *GET-request* för att hämta frågor från Opentdb API och därefter visas frågorna i utvecklarkonsolen. Frågorna kommer in med JSON-format.


```
{
  "response_code": 0,
  "results": [
    {
      "category": "Geography",
      "type": "multiple",
      "difficulty": "medium",
      "question": "Which of the following is not a megadiverse country - one
  that harbors a high number of the earth&#039;s endemic species?",
      "correct_answer": "Thailand",
      "incorrect_answers": [
        "Peru",
        "Mexico",
        "South Africa"
      ]
    }
  ]
}
```

Kodexempel 6 Råa JSON-datan från Opentdb API

Första prototypen hade hårdkodade parametrar i API kallelsen eftersom parametervalet inte påverkar spelets funktionalitet och kan programmeras in senare i användargränssnittet.

Flervalsfrågorna som kommer från Opentdb har alltid fyra svar totalt varav tre svar är rätta och ett är fel. Detta betyder att fyra HTML element måste skapas för att hålla svarsalternativen. Rätt/fel frågor har endast två svarsalternativ vilket kräver endast två HTML element för att visa svarsalternativen.

Vid nästa utvecklingssteg skapades HTML element på klientsidan och föra in data till dem med JavaScript. Första prototypen hade ett element för att visa frågorna samt svaren och en indikator både för egna poäng och rundtiden.

3.1.3 Serverfunktionalitet

Efter att ha en färdig lokal prototyp av spelet blir utvecklingen av serversidan lättare eftersom man lättare kan planera vilka händelser som behövdes programmeras för att Socket.IO ska kunna hantera multiplayer funktionaliteten av spelet.

Utvecklingen av serversidan bestod till en början av att använda webbläsarens funktion för att logga data i utvecklarkonsolen samt använda Node.js terminalen för att logga data på serversidan för att klargöra att rätt data överförs mellan servern och klienten.

Eftersom spelet bara kan spelas i en instans måste startknappen inaktiveras då en spelare startar spelet vilket görs med en *emit*-händelse som skickar ett meddelande från spelaren som startar spelet via servern till övriga spelare.

En utmaning som måste lösas var att frågorna som kommer från Opentdb API är i slumpmässig ordning vilket betyder att frågorna måste hämtas en gång och sedan distribueras till alla klienter via servern. Parametrarna som sätts i *GET-request* som hämtar frågorna sätts in på klientsidan varifrån de hämtas först till spelaren som startade

spelet varefter de skickas till servern. Efter det distribuerar servern frågorna till övriga spelare.

För att starta spelet för alla spelare programmerades en *emit*-händelse som klienterna lyssnar på med *on*-händelsen och kör en funktion som startar spelet när händelsen kommer in.

3.1.4 Förbättringsfasen

Efter att prototypen var färdig, utvecklades spelet vidare med ytterligare funktionalitet enligt förbättringsidéer som märktes i första iterationen.

Data som hämtas från Opentdb kommer alltid i samma form och datan är i samma ordning. Frågan kommer först, sedan kommer rätt svar och till sist kommer en *array* av fel svar. Detta betyder att man inte endast kan visa svaren på skärmen i ordningen de kommer in utan man måste ordna om frågorna i en slumpmässig ordning, annars är första svarsalternativet alltid det rätta svaret.

```
//correct
var correct = document.createElement("option");
correct.value = "correct";
correct.className = "questionSelect";
correct.innerHTML = htmlDecode(data[i].correct_answer);

randomInsert = getRandomInt(0, data[i].incorrect_answers.length);

//loop through wrong answers and insert correct anser in random index
for (let j = 0; j < data[i].incorrect_answers.length; j++) {
  var incorrect = document.createElement("option");
  incorrect.value = "incorrect";
  incorrect.className = "questionSelect";
  incorrect.innerHTML = htmlDecode(data[i].incorrect_answers[j]);
  listViewItem.appendChild(incorrect);
  if (j == randomInsert) {
    listViewItem.appendChild(correct);
  }
}
```

Kodexempel 7 Frågorna ordnas om slumpmässigt i for-loopen som sköter om att visa och byta frågorna

En poänglista är viktig för ett quiz spel för att spelare ska kunna hålla koll på poängställningen samt se vem som leder spelet. Funktionaliteten för att lägga till poängen för spelare programmerades i klientsidan. Varje gång en spelare har valt en fråga, avgör spelet om det var rätt eller fel svar. Spelet har ett element som visar egna

poäng vilket är endast på kliensidan och den uppdateras direkt. Poänglistan uppdateras med att skicka in varje spelares poäng till servern varje gång en runda är över med en *emit*-händelse, vartefter servern gör en *emit*-händelse som skickar ut poänglistan åt alla klienter

Opentdb API har inbyggd funktionalitet för att filtrera frågor. I slutskedet av spellogiksprogrammeringen programmerades ett inställningselement i användargränssnittet. Elementet består av HTML *select* element som har parametrar att välja emellan antal frågor, kategori, svårighetsgrad och typ av fråga. Parametrarna insätts i API kallelsen som hämtar frågorna från Opendtb.

```
<select id="categories">
<option value="10">Entertainment: Books</option>
<option value="11">Entertainment: Film</option>
<option value="12">Entertainment: Music</option>
<select>

var category = "&category=" + document.getElementById("categories").value;
$.getJSON(https://opentdb.com/api.php?amount=10"+category+")
```

Kodexempel 8 Spelare väljer inställningar i klientsidan som sätts in som parametrar i API kallelsen

I första prototypen var rundtiden hårdkodad som en X mängd sekunder. Eftersom rundtiden väljs på klientsidan före man börjar spelet måste rundtiden skickas till spelare via servern. Spelaren som börjar spelet skickar rundtiden med en *emit*-händelse och servern skickar den med en *emit*-händelse till övriga spelare före spelet börjar

I förbättringsfasen av utvecklingsprocessen programmerades det en *Reset Game* knapp till spelet som har funktionaliteten att stoppa och återställa spelet. Funktionaliteten uppnås med en *emit*-händelse som skickas till servern när en spelare trycker på *Reset Game* knappen. Därefter skickar servern en *emit*-händelse till alla klienter som gör en exekvering av en funktion som återställer spelet på klientsidan.

3.1.5 Visuellt utseende av spelet

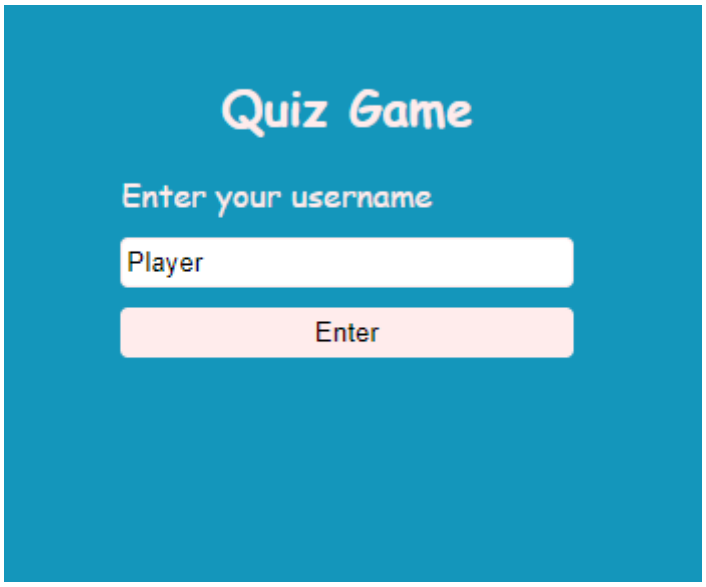
Det slutliga steget i utvecklingsprocessen var att förbättra den visuella delen av spelet. En av uppgifterna var att flytta på spelets element till logiska positioner på skärmen.

Elementen måste placeras i mitten av webbsidan med CSS för att standardlogiken i HTML är att placera element i vänstra sidan av webbsidan. Storleken av alla element gjordes större för att förbättra användarvänligheten. Till exempel knapparna i HTML har en relativt liten storlek utan förändringar med CSS. Spelets färgvärld är visuellt trög utan att tillägga färger med CSS vilket ledde till att utveckla färgerna i spelet. Färgvalet har som krav en lämplig kontrast mellan texten och bakgrunden så att texten har en bra synlighet. Standardfonten i de mest välkända webbläsarna är ointressanta vilket ledde till att byta fonten i spelet till en mera visuellt tilltalande font, vilket i detta fall blev fonten Comic Sans.

3.2 Resultat

3.2.1 Spellogik

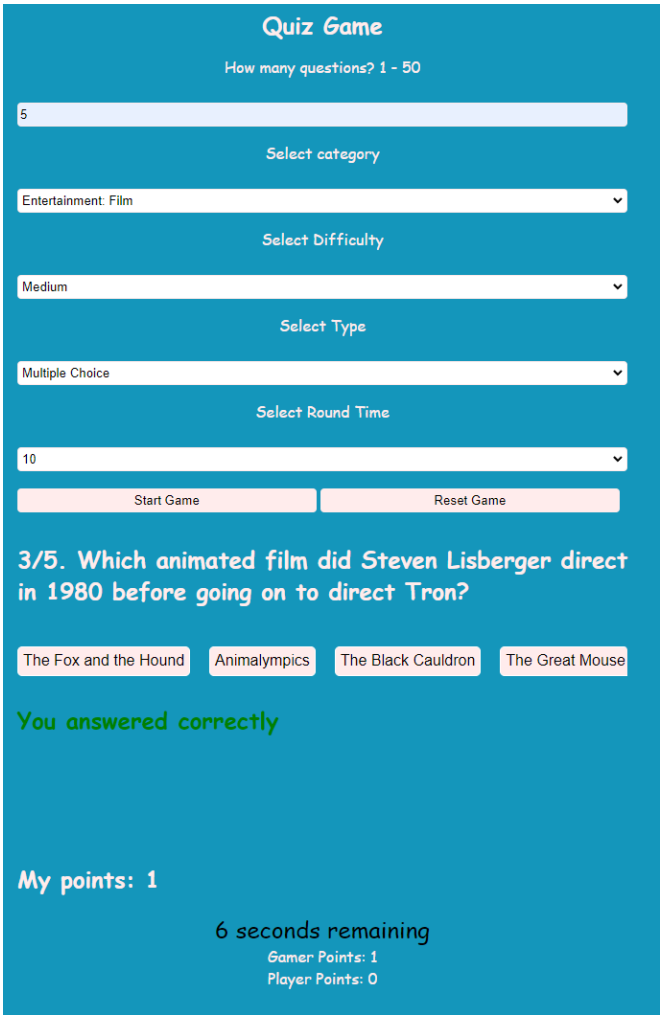
Spelet har en startsida där spelaren anger sitt namn som senare syns i poänglistan.



Figur 2 Skärmdump av startsidan

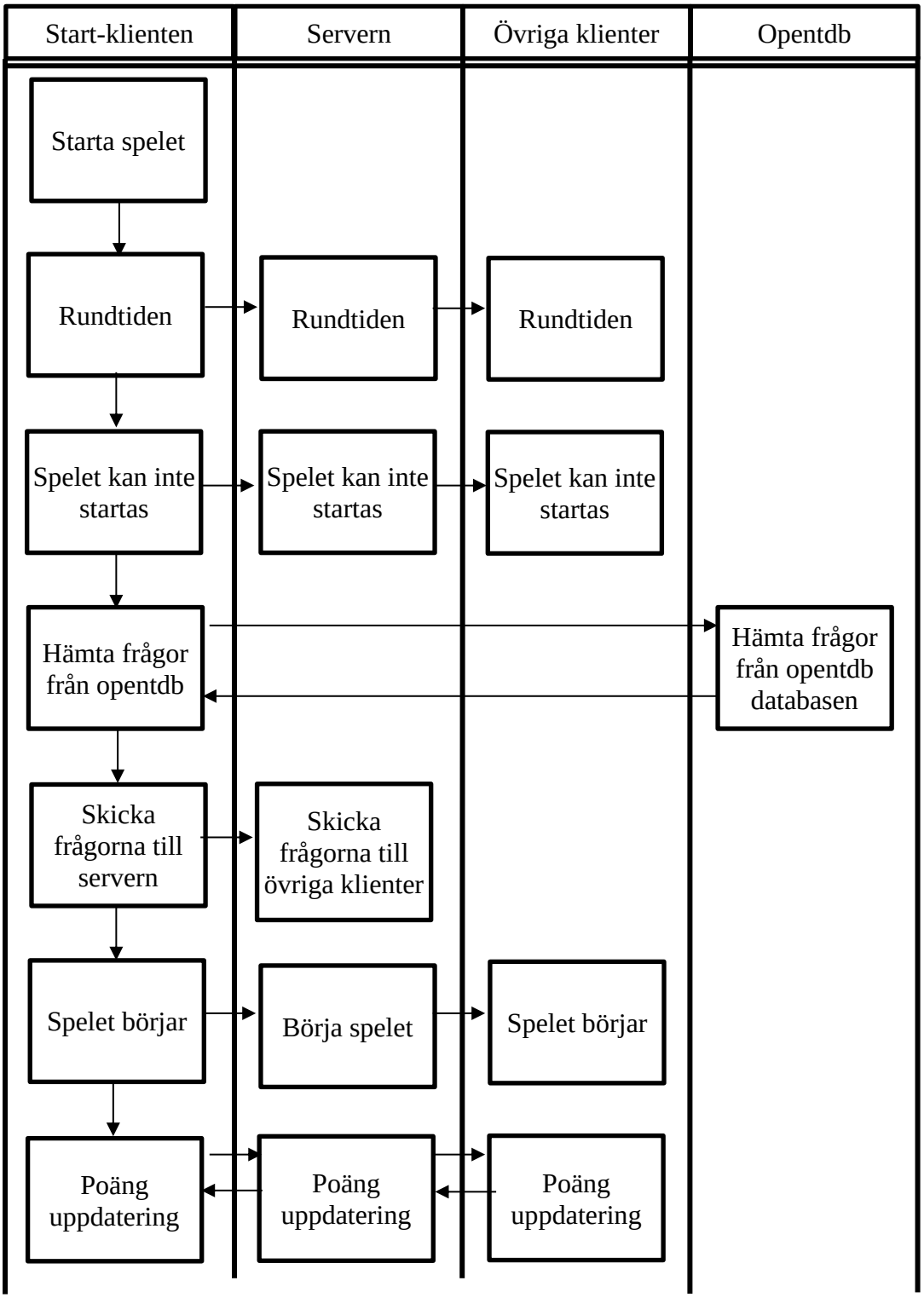
Efter startsidan kommer spelaren in i själva spelet. På spelsidan kan man byta inställningarna före man börjar. *Start Game* knappen startar spelet och webbläsaren hämtar frågorna in till spelet med att göra ett anrop till Opentdb API med parametrarna

från spelets inställningar. När någon av användarna trycker på startknappen börjar spelet för alla som är på spelsidan. Spelet börjar endast om man var inne på spelsidan då startknappen trycktes. Det går inte att gå med i spelet efter att den har börjat. Spelet matar ut frågor för spelarna att svara på före tiden tar slut. Om en spelare svarar rätt ökar mängden poäng i poänglistan för respektive spelare och ett grönt meddelande visas på skärmen som indikerar att man har svarat rätt. Om man svarar fel visas ett rött meddelande som säger att man har svarat fel. Efter att spelet tar slut blir poängen kvar och spelet ligger stilla. Spelare kan trycka på *Reset Game* för att nollställa spelet.



Figur 3 Skärmdump av spelsidan

3.2.2 Socket.IO och kommunikation mellan klienten och servern



Figur 4 Swimlane flowchart över kommunikation mellan servern och klienten med Socket.IO

Node.js servern och Socket.IO hanterar flera viktiga funktioner för quiz spelet. Socket.IO håller reda på om spelet har startat och ifall det har startat kan man inte starta spelet. Socket.IO startar spelet från servern för övriga spelare efter att spelet har börjats av en klient. Frågorna som hämtas från Opentdb sparas på servern med Socket.IO. Poängställningen uppdateras av Socket.IO vid varje runda efter att klienterna har skickat in deras poäng till servern.

Klienten har en *property* med booleskt värde som bestämmer om spelet kan startas. Man kan endast starta spelet om *boolean*-värdet *hasGameStarted* är negativt. När en spelare startar spelet skickar Socket.IO det till servern med en *emit*-händelse och därifrån till alla anslutna klienter som lyssnar med *on*-händelsen. Det leder till att ingen kan starta spelet mera.

```
//första spelaren skickar till servern att spelet har startat
socket.emit("startGameBool", hasGameStarted);

//servern skickar till alla anslutna spelare att spelet har startat
socket.on("startGameBool", (GameStarted) => {
  hasGameStarted = GameStarted;
  io.emit("sendHasGameStarted", hasGameStarted);
});

//klienten sparar boolean-värdet
socket.on("sendHasGameStarted", (GameStarted) => {
  hasGameStarted = GameStarted;
});
```

Kodexempel 9 Servern och Socket.IO hanterar när spelet börjar

En väsentlig uppgift som servern har är att spara frågorna som hämtas från Opentdb. Ifall klienterna varje gång skulle hämta frågorna själva med Opentdb API skulle frågorna vara olika för alla spelare eftersom de kommer i slumpmässig ordning. Spelaren som startar spelet hämtar frågorna från Opentdb som sparas i en JSON-data variabel. Socket.IO skickar en *emit*-händelse till servern med variablen varifrån den blir skickad ut till alla andra spelare.

```
//klienten hämtar frågorna från opentdb
$.getJSON("https://opentdb.com/api.php?amount=" +
   document.getElementById("questionsAmount").value + category + difficulty +
   type, function(result){ }

//frågorna skickas till servern
if (hasGameStarted == false) {
  socket.emit("gameData", result);
}
//listan av frågorna skickas till alla spelare
```

```
socket.on("gameData", (gameData) => {
  listView = gameData;
  if (listView) {
    io.emit("gameHasStarted", listView);
  }
});

//klienten sparar datan som kom från servern
socket.on("gameHasStarted", (sentgameData) => {
  receivedGameData = sentgameData;
});
```

Kodexempel 10 Frågorna hämtas från Opentdb, sparas i servern och skickas ut till spelare

Efter att klienten har hämtat frågorna skickar klienten en *startGameAll emit*-händelse till servern som sedan skickar ut en *emit*-händelse vidare till alla anslutna klienter och då körs det en funktion på klientens sida som startar spelet.

```
//Första spelaren som startar spelet skickar en händelse till servern
socket.emit("startGameForAll", "startgame");

//servern tar emot det och skickar åt alla anslutna spelare att starta spelet
socket.on("startGameForAll", (startgame) => {
  io.emit("startGameAll", "startgame");
});

//klientsidan startar spelet
socket.on("startGameAll", function (data) {
  if (isGameRunning == false) {
    startGame();
  }
  isGameRunning = true;
});
```

Kodexempel 11 Servern och Socket.IO hanterar när spelet börjar

Servern måste spara poängställningen och skicka ut en *emit*-händelse med nya poängställningen varje gång den ökar.

```
//spelaren skickar in sina poäng till servern
socket.emit("userPoints", { user: user, points: myPoints });

//servern sätter in nya värden i poänglistan och skickar den till spelarna
//som sedan visas på klientsidan
socket.on("userPoints", (data) => {
  userList[data.user] = data.points;
  setTimeout(() => {
    socket.emit("userList", userList);
  }, 500);
});
```

Kodexempel 12 Servern håller reda på poängställningen i spelet

Eftersom rundtiden av spelet kommer från klientsidans inställningar måste spelaren som startar spelet skicka en *emit*-händelse med rundtiden till servern varifrån den kan skickas åt alla spelare.

```
//spelaren som startade spelet skickar in hur långa rundorna är
socket.emit("roundTime", roundTime);

//servern tar emot tiden och skickar den till alla spelare
socket.on("roundTime", (data) => {
  io.emit("receiveRoundTime", data);
});

//rundtiden kommer från servern och sparas hos klienten
socket.on("receiveRoundTime", (rtime) => {
  receivedRoundTime = rtime;
});
```

Kodexempel 13 Klienten skickar in rundtiden till servern, varifrån den skickas till alla spelare.

3.3 Analys

3.3.1 Socket.IO som verktyg för nätkommunikation

Socket.IO är ett lämpligt verktyg för utveckling av program som kräver dubbelriktad händelsebaserad kommunikation. Quiz spelet har ingen märkbar latens och Socket.IO hanterar kommunikationen mellan klienten och servern i spelet som planerat och enligt alla krav. Planen var enligt kraven att spelet har all basfunktionalitet som krävs för ett quiz spel det vill säga Kravena var följande: Multiplayer, uppdatering av poängställningen, en klocka som räknar ner rundtiden samt frågorna måste komma från en utomstående API.

3.3.2 Användbarhet

Quiz spelet har all basfunktionalitet som sattes som krav för spelet. Flera spelare kan ansluta sig till spelet samtidigt. Man kan välja frågorna före man börjar spelet enligt genre, svårighetsgrad samt typ av fråga, det vill säga flervalsfrågor eller rätt/fel frågor. Spelet har en klocka som räknar ner runtiden. Poängställningen uppdateras från servern varje gång en spelare har svarat på en fråga.

Resulterande spelet är i grund och botten en prototyp och kräver vidareutveckling ifall man vill publicera spelet, men prototypen är tillräcklig för att bevisa att Socket.IO är ett lämpligt verktyg för utvecklingen av multiplayer spel som har dubbelriktad kommunikation mellan en klient och en server i realtid

4 DISKUSSION

I diskussionskapitlet beskrivs huvudresultatet av arbetet. Resultatet jämförs med ett liknande arbete som har gjorts tidigare och tidigare arbetets styrkor analyseras för att få förbättringsförslag till detta arbete. Därefter beskrivs allmänna styrkor och svagheter i detta arbete och förbättringsförslag ges för vidareutveckling.

4.1 Huvudresultat

Socket.IO lämpar sig bra för att utveckla ett multiplayer spel som har realtidskommunikation mellan klienten och servern på grund av följande egenskaper. Socket.IO är byggt på WebSocket-protokollet vilken är händelsebaserad och dubbelriktad vilket möjliggör kommunikation med låg latens vilket är en förutsättning för realtidskommunikation. Socket.IO har två huvudsakliga händelserna *emit* och *on*, vilka man kan använda för att överföra data för spellogiken mellan klienten och servern. Slutresultatet har inga oförväntade brister gällande kommunikationen mellan klienten och servern vilket tyder på att Socket.IO lämpar sig bra för att uppnå arbetets mål.

4.2 Tidigare forskning

År 2013 gjord skrev Brandon Barker blogginlägget *Creating a real-time multiplayer web game with Node.js and socket.io* var han beskriver utvecklingsprocessen av ett triviaspel som använder Socket.IO för realtidskommunikation mellan klienten och servern. Spelet som Barker utvecklade har många likheter till quiz spelet som beskrivs i detta arbete.

Spelet går ut på att gissa vilket år en film är släppt från topp-250 listan av Internet Movie Database (IMDb) Det vill säga spelet hämtar frågor från en utomstående källa lika som quiz spelet hämtar frågor från en utomstående API. Brandons spel har en *emit* och en *on* händelse vilka heter *logPoints* vilket liknar quiz spelets *userPoints*-händelse.

Funktionalitetsmässigt är Barker's spel på ungefär samma nivå som quiz spelet i detta arbete. Koden är bättre strukturerad samt spellogiken är byggt mera objektorienterat vilket ökar på läsbarheten av koden och förbättrar kodstrukturen. Främsta förbättringen som Brandons spel har är att spellogiken för att evaluera om svaret som spelaren gett är rätt eller fel görs på serversidan, vilket leder till att man inte kan fuska med att inspektera klientsidans källkod. Detta borde vara standarden i spel som är tävlingar. Barker kommenterar inte själv på hur Socket.IO lämpar sig för utvecklingen av spelet men eftersom spelet verkar fungera bra utan märkbara brister kan vi dra slutsatsen att Socket.IO var ett lämpligt val för triviaspelet. (Barker, 2013)

4.3 Styrkor och svagheter

Quiz spelet är inte tillräckligt utvecklad för att använda fulla kapabiliteten av Socket.IO. Socket.IO har mycket funktionalitet som kunde signifikant förbättra resultatet. Spelet fungerar som en bra bas för vidareutveckling med Socket.IO.

En svaghet i detta arbete var brist på feedback från användare. Spelet kunde ha planerats och itererats bättre med mera feedback, än endast från en ensam utvecklare. Tävlingsmässigt är det lätt att fuska i spelet eftersom klientsidans HTML element innehåller information om vilka av svarsalternativen är rätt. Det kunde ha varit en viktig del av arbetet att mäta prestandan av Socket.IO när användarmängden stiger men det var inte möjligt med mängden användare som var tillgängliga.

Styrkan i resultatet är att spelet fungerar som planerat och lider inte av stora oförväntade brister. Socket.IO är ett nöje att arbeta med eftersom det är lätt att installera och börja använda samt dokumentationen är relativt bra.

4.4 Vidareutveckling

Quiz spelet är ett spel med endast nödvändig funktionalitet. Man kunde vidareutveckla projektet genom att tillägga funktionaliteter samt förbättra den visuella designen för att förbättra spelupplevelsen.

Funktionalitetsmässigt kunde spelet vidareutvecklas med att visa frågan efter rundan och indikera vilka svar som var rätt och fel samt hur stor andel av spelarna svarade på en specifik fråga. Spelet skulle också kunna visa en lista i slutet av spelet som innehåller alla frågor som frågades under spelomgången och visa statistik över svaren från spelarna. Informationen om rätta och fel svar bör gömmas från klientsidan så att man inte kan fuska med att inspektera HTML element med webbläsarens utvecklarverktyg.

Socket.IO har inbyggt funktionalitet för rum vilket vore ett krav ifall man skulle publicera spelet för allmänheten. Med rum-funktionalitet kunde man skapa flera instanser av spelet med egna väntrum och spelrum vilket i sin tur skulle öka på mängden av användare som kan spela samtidigt. Ett experiment som kan göras i vidareutvecklingen är att om spelarmängden överskrider ett visst antal kan man göra en till instans av spelet på en annan server för att se hur det skulle fungera med målet att minimera belastning på en server.

Datan som överförs mellan servern och klienten i detta spel är i form av primitiva datatyper eller objekt. Om man vill vidareutveckla spelet kunde man forska i vilka former av data som kan överföras med Socket.IO. Spelet kan dra nytta av att överföra till exempel bilder eller ljud

Visuellt utvecklande av spelet skulle göra spelet mera användarvänligt. Spelsidan kan utvecklas vidare som mål att vara mer visuellt tilltalande. Poänglistan och klockan är designmässigt bristfälliga och kan vidareutvecklas. Ett tillägg som skulle öka på spelets visuella värde är att spelare kan välja en avatar i spelet för att identifiera spelare, vilket är en allmän funktion som quiz spel ofta har.

Ett förbättringsförslag är att göra en prototyp av spelet och sedan publicera det på någon plattform där det kan spelas. Feedbacken från spelare kan vara en avgörande faktor i spelets användarvänlighet och intresse när man designar spel.

5 SLUTSATSER

Detta arbete har behandlat utvecklingen av ett webbaserat multiplayer spel som använder sig av dubbelriktad realtidskommunikation som hanteras av Socket.IO. För att få svar på forskningsfrågan har en prototyp av ett webbaserat quiz spel utvecklats. Svaret på forskningsfrågan är jakande: Socket.IO är ett lämpligt verktyg för att utveckla ett spel inom givna kraven. Hypotesen som gjordes före arbetet började var rätt.

RAD-metoden som användes i mjukvaruutvecklingsprocessen fungerade bra för detta arbete. Eftersom Socket.IO var ett nytt verktyg för mig var det lättare att se helheten av verktygets kapabilitet efter att man hade skapat en prototyp med basfunktionalitet. Jag är övertygad om att RAD-metoden var ett bra val i stället för en metod där man lägger fokus på planering i början och skriver långa specifikationer på exakt funktionalitet man vill att produkten ska ha.

Slutprodukten blev som planerat från början och ingen funktionalitet som hade planerats från början uteblev från spelet. Quiz spelet är en användbar produkt som innehåller basfunktionalitet för ett quiz spel.

Eftersom spelet inte var från början planerat att publiceras för allmänheten direkt utan endast som en prototyp som innehåller basfunktionaliteten av Socket.IO blir inte spelet publicerat i nuvarande form, men spelet är en bra bas för vidareutveckling i framtiden om man vill publicera den.

6 KÄLLOR

Barker, B. (2013). *Creating a real-time multiplayer web game with Node.js and socket.io.* [Online]. Brandon Barker Ramblings of a software developer and technology junkie. Tillgänglig: https://blog.brandonbarker.net/2013/09/02/creating-a-real-time-multiplayer-web-game-with-node-js-and-socket-io/ (Hämtad 28.04.2022)

IBM. (2021). *What is Ajax?* [Online]. IBM Rational Software Architect documentation. Tillgänglig: https://www.ibm.com/docs/en/rational-soft-arch/9.6.1?topic=page-asynchronous-javascript-xml-ajax-overview (Hämtad 27.04.2022)

Kozierok, C. (2005). *The TCP/IP Guide Version 3.0.* [Online]. Tillgänglig : http://www.TCPIPGuide.com (Hämtad 23.04.2022)

Laudon, N. (2022). *Vad menas med agil metodik och agila metoder?* [Online]. Cornerstone. Tillgänglig: https://www.cornerstone.se/inspiration/affarskompetens/projekt-och-agila-arbetssatt/vad-menas-med-agil-metodik-och-agila-metoder (Hämtad 27.04.2022)

Mdn (2022). *JavaScript.* [Online]. Resources for Developers, by Developers. Tillgänglig: https://developer.mozilla.org/en-US/docs/Web/JavaScript (Hämtad 23.04.2022)

Microsoft. (2022). *Getting Started.* [Online]. Visual Studio Code. Tillgänglig: https://code.visualstudio.com/docs (Hämtad 23.04.2022)

npm Inc. (2022). *NPM Inc.* [Online]. Tillgänglig: https://www.npmjs.com/ (Hämtad 23.04.2022)

Outsystems. (2022). *What is Rapid Application Development?* [Online]. Glossary. Tillgänglig: https://www.outsystems.com/glossary/what-is-rapid-application-development/ (Hämtad 23.04.2022)

Parker Software. (2022). *ELI5: the benefits of WebSockets.* [Online]. ELI5. Tillgänglig: https://www.thinkautomation.com/eli5/eli5-the-benefits-of-websockets (Hämtad 24.04.2022)

PixelTail Games. (2022). *Open Trivia Database.* [Online]. Tillgänglig: https://opentdb.com/ (Hämtad 23.04.2022)

Sandvine. (2022) *The Global Internet Phenomena Report January 2022.* [Online]. Global Internet Phenomena. Tillgänglig: https://www.sandvine.com/phenomena (Hämtad: 23.04.2022)

Screenbeam. (2022). *How to Reduce Latency or Lag in Gaming.* [Online]. Wifihelp. Tillgänglig: https://www.screenbeam.com/wifihelp/wifibooster/how-to-reduce-latency-or-lag-in-gaming-2/ (Hämtad 23.04.2022)

Socket.IO. (2022). *Documentation.* [Online]. Socket.IO. Tillgänglig: https://socket.io/docs/v4/ (Hämtad 23.04.2022)

Synopsys Editorial Team. (2017). *Top 4 software development methodologies. [Online*]. Synopsys, Inc. Tillgänglig: https://www.synopsys.com/blogs/software-security/top-4-software-development-methodologies/ (Hämtad 27.04.2022)

W3Techs. (2022) *Usage statistics of JavaScript as client-side programming language on websites* [Online]. JavaScript Libraries. Tillgänglig: https://w3techs.com/technologies/details/cp-javascript/ (Hämtad 23.04.2022)