Informatik

Skriftlig individuell hemtentamen i Serverprogrammering i JavaScript (ISGB17)

Ansvariga lärare:	Peter Bellström (054 – 700 16 92), Pierre Sjöberg (054 – 700 13 90) och Alexander Vestin (054 – 700 11 32).
Datum:	Lördagen den 28 augusti 2021
Klockan:	08.15 – 13.15
Betygsgränser:	Se kriterierna i texten nedan.
Icke tillåtna hjälpmedel:	Samarbeta och/eller ta hjälp av någon annan.
Övrigt:	I Canvas stänger möjligheten att lämna in svar på tentamensuppgiften klockan 13.30. Därefter är det alltså inte möjligt att lämna in någon lösning. Den skriftliga individuella tentamen pågår dock bara till 13.15 men för att undvika problem med ”sista minuten”-inlämningar och problem med tekniken är uppgiften öppen till 13.30.

LYCKA TILL!

Guess the Number

Din kamrat har påbörjat att ta fram en JavaScript/NodeJS-löning för ett nytt spel kallas för ”*Guess the Number*”. Spelet går ut på att en server slumpar ut ett tal och sedan ska två spelare gissa på talet. Den spelaren som gissar på rätt nummer på minst antal gissningar vinner. Din kamrat har dessvärre bara färdigställa klientkoden och några enstaka metoder/funktioner för serversidan.

För att underlätta för dig har din kamrat tagit fram startkod innehållande HTML, CSS och JavaScriptfiler samt även en kravspecifikation över hur spelet skall fungera. Din kamrat har också delat upp kravspecifikationen i fyra delar och *kravet för betyget **godkänt** är att du minst lämnar in lösningar för två av delarna och dessa fungerar att köra och löser uppgifterna enligt kraven från din kamrat*.

*Kravet för betyget **väl godkänt** är att du minst lämnar i lösningar för minst tre delar och dessa fungerar att köra och löser uppgifterna enligt kraven från din kamrat.* Då din kamrat är mycket noggrann och vill ha koden och lösningen på sitt sätt har hen också formulerat följande krav som gäller både för betyget **godkänt** för betyget **väl godkänt**:

1. Din lösning skall köras i ”strict mode” vilket med andra ord betyder att **'use strict';** är ett krav.
2. Din lösning skall vara skriven för NodeJS i JavaScript (JS)
3. I din lösning tillämpar du strukturerad kod. Du skall alltså använda dig av indentering (tab) och därtill börjar skall start-spetsparantesen ({, ”måsvinge”) alltid finnas på första raden i uttrycket. T ex
   if() {
   }
4. Att göra några ändringar i index.html, tenta-modul.js eller client-script.js är inte tillåtet.
5. Du får bara skriva kod i filen server.js.
6. När du lämnar in din lösning på Canvas skall du komprimera samman samtliga filer och kataloger som utgör examinationsuppgiften och lämna in som ett zip- eller rar-arkiv (fil).
7. Din kamrat kommer att provköra dina lösningar (dina funktioner) och förväntar sig att dessa är körbara. Är inte en funktioner körbar eller inte löser uppgiften på specificerat sätt kommer din kamrat att sluta provköra funktion och den uppfyller då ***inte*** kraven för en godkänd funktion.
8. I filen server.js skall du högst upp skriva den kod du erhöll när du anmälde dig till tentamen.
9. ***I och med att du laddar upp din lösning på Canvas intygar du samtidigt att det är din lösning och du har inte har samarbetat och inte heller tagit hjälp av någon annan person.***

För att lösa uppgifterna (de fyra delarna) och ta hand om alla krav har din kamrat skrivit mer eller mindre detaljerade listor för respektive del (se sidorna som följer). För två av delarna finns en mer detaljerad lista, en ”kokbok”, över vad koden i delen skall göra och för två av delarna finns endast en övergripande förklaring till vad din kamrat vill att delarna skall lösa inom ramen för spelet. De fyra delarna du skall skriva kod för är:

1. Skapa webbserver med två endpoints
2. Starta socketanslutningar samt skicka händelsen ”game-start”
3. Skapa funktionalitet för sockethändelsen ”player-ready” och ”player-play”
4. Skapa funktionalitet för sockethändelsen ”player-done”

Lycka till!

För del 1 har din kamrat skrivit följande kravlista (”kokbok”):

I **del 1** skall du med NPM installera paketen express, socket.io och cookie-parser och sedan skriva funktionalitet för följande:

1. Installera paketen express, socket.io och cookie-parser.
2. Starta upp en webbserver som lyssnar efter anrop på port 3000
3. Gör mappen ”*public*” still static (åtkomlig för klienterna)
4. Vid GET-anrop på endpoint ”/” skall följande ske
    a. Hämta ut kakan ”namn” och spara den i variabeln namn
    b. Hämta ut kakan ”gdpr” och spara den i variabeln gdpr
    c. Kontrollera om namn eller gdpr är **undefined**, om sant:
        i. Skicka filen *”login.html”* till klienten
    d. Om namn och gdpr inte är **undefined**
        i. Skicka filen *”index.html”* till klienten
5. Vid POST-anrop på endpointen ”/” ska följande ske:
    a. Hämta ut värdet på fältet med name=”*namn*” i *login.html* och spara i variabeln namn.
    b. Hämta ut värdet på fältet med name=”gdpr” i *login.html* och spara i variabeln gdpr.
    c. Skapa ett try-catch-block i vilket du kontrollerar följande:
        i. Om namn är tomt, kasta felet ”Du måste ange ett namn”
        ii. Om namn är mindre än 5 tecken, kasta felet: ”Namnet måste vara minst 5 tecken långt”
        iii. Om namn inte börjar med en versal, kasta felet: ”*Namnet måste börja med en versal”*. Använd funktionen ”*isFirstUpperCase(str)”* som finns i tenta-modul.js för att kolla om en sträng börjar med en versal
        iv. Om gdpr är **undefined**, kasta felet: *”Du måste tillåta att ingen data kommer att sparas!”*
    d. Om inget fel kastas, spara namn i en kaka med namnet ”namn” och spara gdpr i en kaka med namnet ”gdpr”. Kakorna skall finnas kvar i en dag.
    e. Dirigera om klienten med GET på endpoint ”/”
    f. Om fel har fångats upp, skicka det uppfångade meddelandet som text till klienten

För **del 2** har din kamrat skrivit följande kravlista (”kokbok”):

I **del 2** skall du starta upp socket kommunikationen och skriva funktionalitet för följande:

1. Vid en inkommande socketanslutning skall följande göras:
    a. Hämta ut en sträng innehållande de anslutande klienternas eventuella kakor
    b. Använda den medskickade metoden parseCookies (globalObject) för att göra om ”kak-strängen” till ett nyckel-värde objekt och spara detta i objektvariabeln cookielist.
    c. Kontrollera om cookielist.namn och cookielist.gdpr är skiljt från **undefined** och om det är så, gör följande:
        i. Kontrollera om anslutande klient är 1 genom att kontrollera ”clientsCount” (io.engine). Om 1 gör följande:
            1. Spara ned cookielist.namn i *playerOneName*
            2. Sätt *playerOneReady* (globalObject) till false
            3. Sätt *playerOneSocketId* (globalObject) till anslutningens id (socket.id)
            4. Sätt *playerOneTotalMoves* (globalObject) till 0
        ii. Om clientCounts är 2, gör följande:
            1. Spara ned cookielist.namn i *playerTwoName*
            2. Sätt *playerTwoReady* (globalObject) till false
            3. Sätt *playerTwoSocketId* (globalObject) till anslutningens id (socket.id)
            4. Sätt *playerTwoTotalMoves* (globalObject) till 0
        iii. Om det är fler än 2 klienter som blir anslutna, då ska du köra metoden ”*socket.disconnect*” med meddelandet: *”Spelet är fullt”*
    d. Om namn är **undefined,** då ska du köra metoden ”*socket.disconnect*” med meddelandet: *”Namn saknas”*

För **del 3** har din kamrat skrivit följande instruktioner:

Här ska du skapa två sockethändelser, ”player-ready” och ”player-play”.

Vid sockethändelsen ”player-ready” ska servern kontrollera om båda spelarna är redo för att spela.

Sätt variblerna playerOneReady (globalObject) till true om klient 1 har tryck på redo-knappen och playerTwoReady (globalObject) till true om klient 2 har tryck på redo-knappen.

Om båda spelarna är redo ska servern skicka ut sockethändelsen ”game-start” till båda klienterna. Servern ska också slumpa ut ett tal mellan 0–10 och spara det i variabeln currentNumber (globalObject). Din kamrat har skrivit metoden getARandomNumber(min, max) (globalObject) som du kan använd för att slumpa ut ett tal.

Vid sockethändelsen ”player-play” ska servern kontrollera om någon av spelarna har vunnit. Skapa variabeln status. Sockethändelsen ska ta emot ett värde som heter ”number”. Om ”number” är mer än currentNumber (globalObject) sätt status till 1 och ”number” är mindre än currentNumber (globalObject) sätt status till 0.

Kontrollera vilken spelare det är och öka numret i variabeln playerOneTotalMoves eller playerTwoTotalMoves (globalObject) med 1 (beroende på vilket klient det är). Sedan ska servern skicka ut sockethändelsen ”game-status” som innehåller ett JSON-objekt till den klient som har gissat på ett nummer. JSON-objektet ska innehålla det är:

”status” – Ska innehålla värdet på variabeln status

”totalMoves – Innehåller hur många gånger spelaren (den som gissade) har gissat

”lastNumber” – Ska innehålla det numret användaren gissade på.

För del 4 har din kamrat skrivit följande instruktioner:

Vid sockethändelsen ”player-done” ska servern kontrollera om båda spelarna har gissat rätt nummer med hjälp av variablerna playerOneDone (globalObject) och playerTwoDone (globalObject). Servern ska sedan kontrollera vilken person som har gjort flest gissningar.

Om båda spelarna har gissar rätt nummer, då ska servern skicka ut sockethändelsen ”game-score” till båda klienterna. Sockethändelsen ska skicka med ett JSON-objekt som ska innehålla:

”p1Winner” – Innehåller värdet: -1 om **Spelare 2** vann, 0 om det blev lika eller 1 om **spelare 1** vann

”p2Winner” – Innehåller värdet: -1 om **Spelare 1** vann, 0 om det blev lika eller 1 om **spelare 2** vann

”p1Name” – Innehåller namnet på **spelare 1**

”p1TotalMoves” – Innehåller det totala antalet gissningar som **spelare 1** har gjort.

”p2Name” – Innehåller namnet på **spelare 2**

”p2TotalMoves” – Innehåller det totala antalet gissningar som **spelare 2** har gjort.