Brisconline

Applicazioni e Servizi Web

Lorenzo Simoncini - 0000931899 {lorenzo.simoncini2@studio.unibo.it}
Maichol Dadi - 0000921828 {maichol.dadi@studio.unibo.it}

24 Agosto 2020

Contents

1 Introduzione

Il progetto si pone l'obiettivo di realizzare una versione online del gioco Briscola, che permette sia di allenarsi contro un'IA di diversi livelli di difficoltà che di giocare contro avversari umani. L'applicazione web permette di registrarsi con un proprio account permettendo così di guadagnare punti giocando le partite e di sbloccare contenuti di personalizzazione estetica esclusivi; guadagnare punti permette anche di scalare la classifica globale dei giocatori registrati all'applicazione. Inoltre, in vari contesti è possibile usufruire di una chat a schermo con cui comunicare con gli altri giocatori.

2 Requisiti

Per poter avere una panoramica di quelle che possono essere le peculiarità del sistema è stato eseguito un meeting i cui punti chiave sono presentati qui di seguito:

- "*Dovremmo poter registrarci al sistema per mantenere le informazioni e le statistiche relative alle partite giocate*"

- "*Dovremmo poter effettuare una personalizzazione degli elementi grafici del gioco (tavolo, carte, ecc.)* "

- "*Vorremmo poterci allenare contro un'IA a difficoltà variabile, oltre alla possibilità di giocare contro altri giocatori*"

2.1 Requisiti Funzionali

Utente

- Deve potersi registrare al sistema, inserendo e-mail, username e password; l'indirizzo mail e lo username dovranno essere univoci.

- Avrà la possibilità di creare una lobby, sia per affrontare altri giocatori che per giocare contro l'IA; è possibile crearla impostando una password, così da prevenire accessi indesiderati.

- Deve poter visualizzare le lobby disponibili create dagli altri utenti e accedervi.

- Potrà ricercare una specifica lobby.

- L'utente avrà la possibilità di interagire con il sistema per giocare le proprie carte all'interno di una partita.

- Un utente potrà uscire in qualsiasi momento da una lobby o una partita notificando l'avvenimento all'avversario. Nel caso in cui questo fosse il creatore della lobby, questa verrà chiusa.

- Potrà visualizzare il proprio profilo con le relative statistiche (partite vinte, giocate, ecc.) e personalizzare alcuni elementi grafici del gioco con i contenuti sbloccati.

- Gli sarà possibile comunicare con gli altri utenti registrati al sistema in una chat pubblica e in una chat privata all'interno della partita.

- Se l'utente non è loggato e prova ad effettuare azioni che necessitano l'accesso, verrà reindirizzato alla schermata di login.

Sistema

- Dovrà mostrare la classifica globale dei giocatori, ordinati in base ai punti ottenuti.

- Gestirà il comportamento dell'IA all'interno delle partite, gestendo le scelte da effettuare

- Dovrà gestire il refresh delle informazioni (messaggi, informazioni sulle lobby/partita, ecc.) in tempo reale.

3 Design

Per realizzare il sistema si è seguito un approccio User Centerd Design (UCD), definendo dei target e sviluppando due Personas utilizzate per simulare diversi tipi di utilizzo dell'applicazione. Il design delle interfacce è stato pensato seguendo la metodologia KISS cercando di focalizzarci sulla semplicità di utilizzo e di comprensione da parte dell'utente.
Inoltre, sono stati applicati dei principi di responsive design per garantire l'usabilità del sistema da tutti i dispositivi e migliorare la User Experience.

3.1 Target User Analysis

Sono stati individuati i seguenti target:

- Giocatore principiante
- Giocatore che gioca soprattutto con i suoi amici

Personas: Luca

Luca è un ragazzo che ha da poco scoperto le regole della briscola, ma non avendo amici che ci giocano ha bisogno di un modo in cui esercitarsi.

- Contesto A: Luca si vuole allenare e vuole trovare un modo per farlo

Scenario d'uso A	
1	Luca si registra al sito inserendo i propri dati
2	Luca scrive in chat per chiedere consigli
3	Luca crea una lobby scegliendo l'IA facile come avversario
4	Luca completa la sua partita

- Contesto B: Luca si è già registrato al sito e ha già giocato diverse partite contro l'IA a livello facile, aumentando il proprio livello

Scenario d'uso B	
1	Luca entra nel suo profilo e personalizza la sua icona e il tavolo da gioco
2	Luca crea una lobby per affrontare l'IA a livello difficile
3	Luca completa la sua partita

Personas: Martina

Martina gioca assiduamente a briscola e vuole un modo per farlo online anche con i suoi amici.

- Contesto A: Martina entra nella lobby di un suo amico protetta da password

Scenario d'uso A	
1	Martina accede al sito inserendo i propri dati
2	Martina scrive l'id della lobby nell'apposita barra di ricerca
3	Martina entra nella lobby inserendo la password
4	Martina gioca contro il suo amico

- Contesto B: Martina crea una lobby per giocare contro un amico

Scenario d'uso B	
1	Martina clicca il tasto di creazione della lobby e imposta una password, che comunicherà al suo amico insieme all'id
2	Martina attende nella lobby l'arrivo del suo amico, personalizzando il proprio profilo
3	All'arrivo dell'amico, Martina inizia la partita

3.2 Storyboard

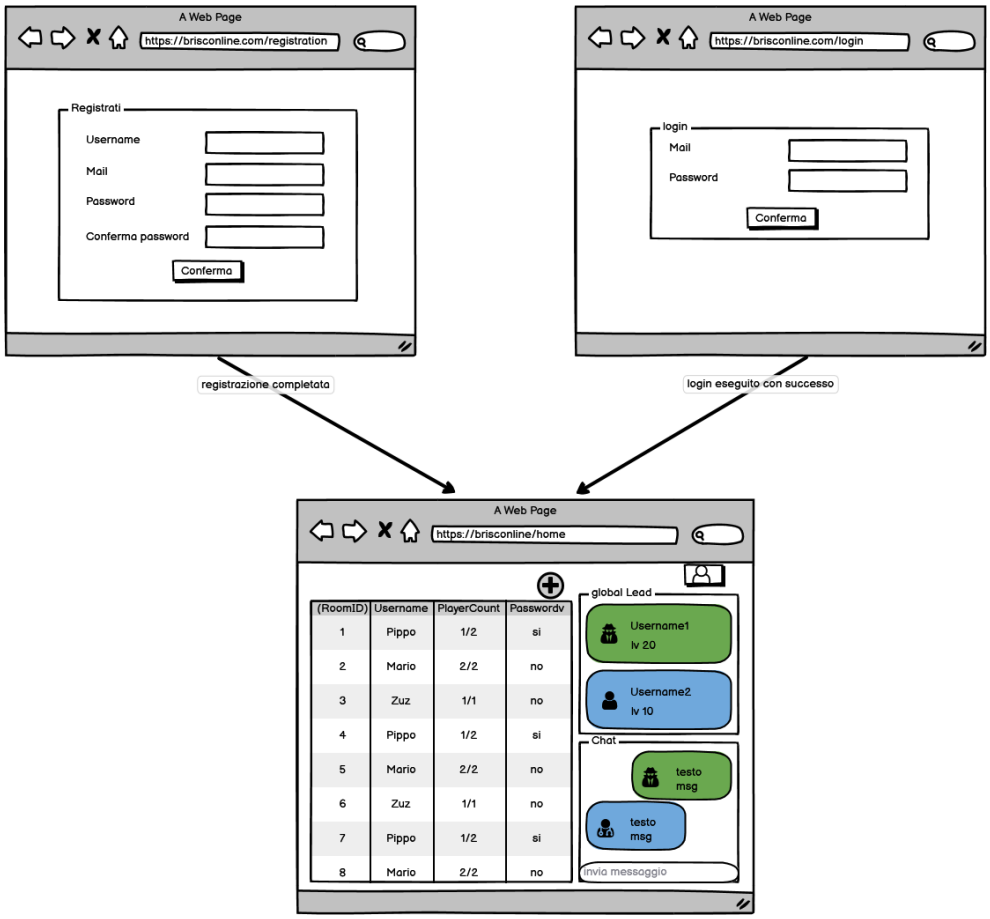

Figure 1: Login e registrazione

Gli utenti che vogliono utilizzare l'applicazione web dovranno prima di tutto registrarsi al sito oppure accedervi tramite le apposite schermate mostrate qua sopra. Una volta inseriti i propri dati con successo, si viene trasportati alla schermata Home.

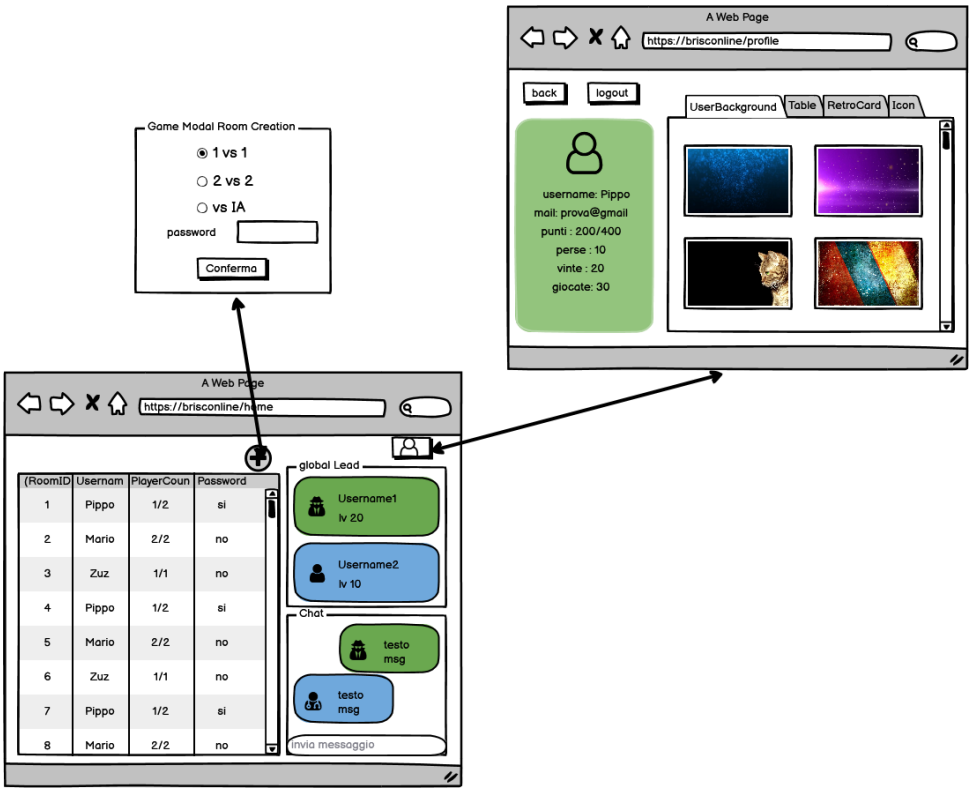

Figure 2: Schermata home e profilo utente

Una volta effettuato il login/registrazione, ci si trova nella schermata home, in cui è possibile visualizzare la lista delle lobby, la classifica degli utenti e la chat globale. Inoltre, dalla home è possibile creare una nuova lobby cliccando l'apposito tasto o accedere alla schermata del profilo utente, in cui visualizzare le proprie informazioni e modificare gli elementi estetici.

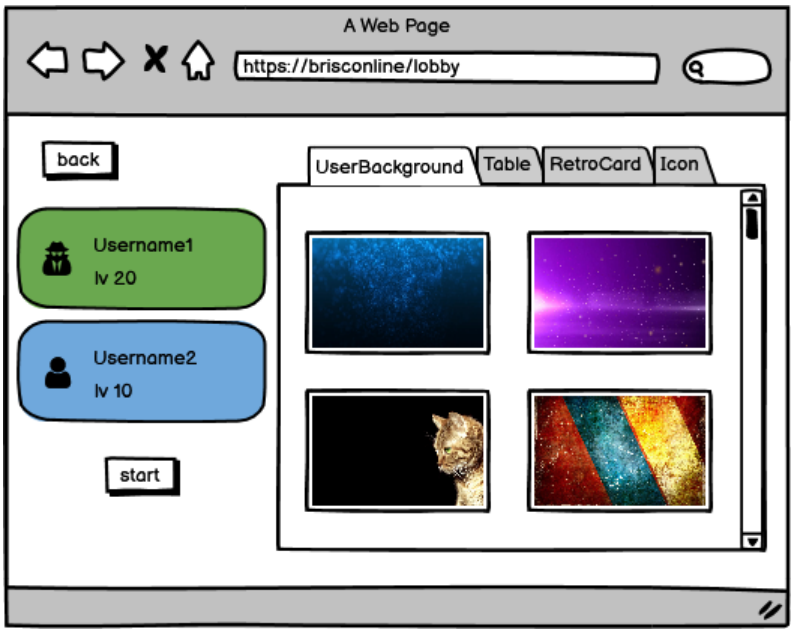

Figure 3: Schermata della lobby

Una volta che si accede ad una lobby creata da un altro giocatore o che se ne crea una con l'apposita finestra si viene reindirizzati alla schermata della lobby, dove è possibile vedere i giocatori attualmente all'interno e modificare gli elementi estetici. Inoltre, se l'utente è il creatore della lobby, è presente un tasto per iniziare la partita.

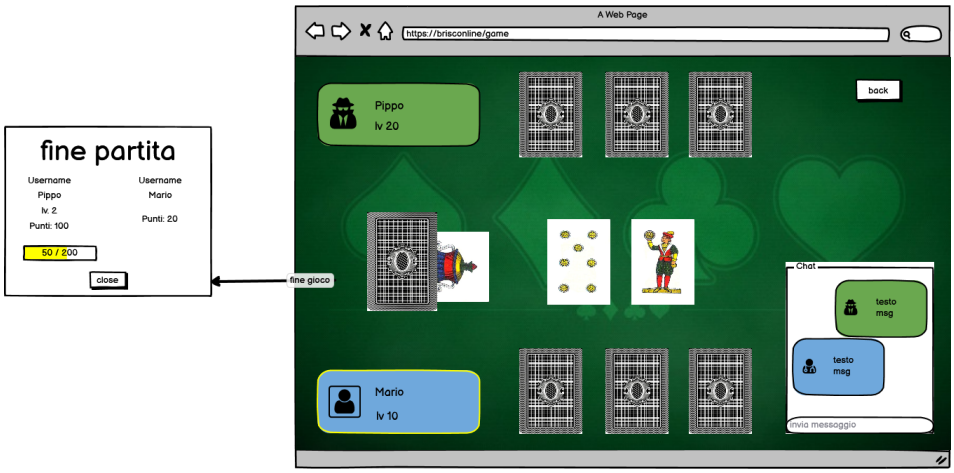

Figure 4: Schermata della partita

Dopo aver avviato il gioco, ci si ritrova nella schermata della partita; da qui è possibile vedere le proprie carte, quelle sul campo, i giocatori presenti e, se si sta giocando contro un avversario umano, una chat per comunicare. Una volta completata la partita apparirà una finestra in cui verrà mostrato il risultato.

3.3 Focus Group

Per ottenere delle opinioni sulla user experience e sull'applicazione in generale è stato condotto un Focus Group tramite mezzi digitali. Il gruppo era composto da sette partecipanti di età compresa tra i 22 e i 27 anni, 4 maschi e tre femmine. Per prima cosa è stata presentata loro l'idea del progetto e gli è stato se conoscevano le regole del gioco, ottenendo risposte perlopiù positive. Successivamente è stata effettuata una presentazione più dettagliata delle funzionalità del sistema utilizzando i mockup visti nella sezione precedente.
Terminata la presentazione è iniziata una discussione generale con i membri del gruppo per capire cosa pensassero dell'idea e della sua applicazione, ottenendo pareri positivi e qualche idea interessante. La funzionalità che precedentemente era stata pensata come link d'invito ad una lobby è stata rimodellata come una ricerca tramite id a seguito di varie proposte, in quanto più immediato.
La scelta degli elementi di personalizzazione è stata eseguita anche in base ai pareri ed alle proposte emerse in questo focus group.

4 Tecnologie

Il sistema è stato realizzato adottando il solution stack **MEVN**; è stata scelta questa soluzione poiché è quella con cui il team di sviluppo ha maggiore confidenza e si adattava bene alle esigenze dell'applicativo.

Stile

Per quanto riguarda lo stile, si è fatto principalmente uso del supporto nativo offerto da Flexbox per ottenere un comportamento piu' predicibile e stabile degli elementi della pagina ed adattarne il contenuto in base alle dimensioni del display.

Persistenza

Nel caso della persistenza delle informazioni lato server, è stata adottata la liberia Mongoose, che permette di interfacciarsi al database MongoDB. Per agevolare lo sviluppo su più fronti, si è deciso di tenere il database sulla piattaforma cloud offerta da MongoDB.

Autenticazione e registrazione

Per gestire la confidenzialità delle informazioni durante le procedure di autenticazione e registrazione al sistema si è scelto di utilizzate il modulo **crypto-js** [2] che ci ha permesso di salvare le password sul database sotto forma di hash, si è scelto di utilizzare l' algoritmo di encriptazione SHA-256.

Comunicazione Client-Server

Per gestire la comunicazione tra client e server, e quindi l'invio di richieste alle API e relative risposte, si è scelto di utilizzare la libreria **axios**.

5 Codice

5.1 Login e registrazione

Dato che la maggior parte delle funzionalità dell'applicativo sono riservate solamente agli utenti che hanno eseguito l'accesso, è stato implementato un sistema per riconoscere gli utenti attraverso il modulo express-session. Prima di accedere ad ogni rotta viene fatto un controllo sulla session a lato server per verificare che l'utente abbia il permesso di accedervi; questo procedimento viene gestito dalle Navigation Guard del Vue Router.

```
1  router.beforeEach((to, from, next) => {
2    if (to.matched.some(record => record.meta.requiresAuth)) {
3      axios({
4        method: "GET",
5        "url": "/session/id",
6        withCredentials: true
7      }).then(result => {
8        if (result.data == "") {
9          next({
10           path: '/login', query: {redirect: to.fullPath}
11         })
12       } else
13         next()
14     })
15   } ...
```

Listing 1: Navigation guard per controllare il login

In questo caso, se l'utente prova ad entrare in una rotta per cui è necessario aver effettuato l'accesso possono verificarsi due casi:

- l'utente non ha eseguito la procedura di login, e quindi viene reindirizzato alla pagina di accesso
- l'utente ha già eseguito l'accesso e viene portato alla rotta desiderata

Dal momento che si ha la necessità di salvare le password nel database, è stato utilizzato il modulo Crypto-js per crittografare le password in hash prima che vengano inviate al server.

5.2 Home

Il componente che rappresenta la Home contiene al suo interno altri cinque componenti: RoomListTable, ModalGameConfig, Chat, GlobalLeadBoard e Header; questi componenti comunicano tra loro mediante l'utilizzo dei Custom Events messi a disposizione da Vue.
Di seguito è presentata una breve descrizione per ognuno di questi componenti:

- La RoomListTable si occupa di recuperare le informazioni sulle lobby create dal database e visualizzarle in un'apposita tabella; per fare ciò, esegue un aggiornamento periodico ogni due secondi, in cui, dopo aver interrogato il database attraverso una specifica API aggiorna la sua lista interna in base alle informazioni ottenute. Inoltre, dovrà gestire il doppio click sulle lobby per permettere l'accesso agli utenti.

- ModalGameConfig si occupa di visualizzare la schermata in cui scegliere le caratteristiche di una lobby al momento della sua creazione.

- Il componente Chat visualizza la Chat tra i giocatori (in questo caso quella globale, verrà vista in seguito quella in game) e permette l'invio di messaggi attraverso di essa.

- GlobalLeadBoard visualizza una classifica dei giocatori in base al loro punteggio.

- Nel componente Header è presente un tasto che permette di accedere al profilo utente.

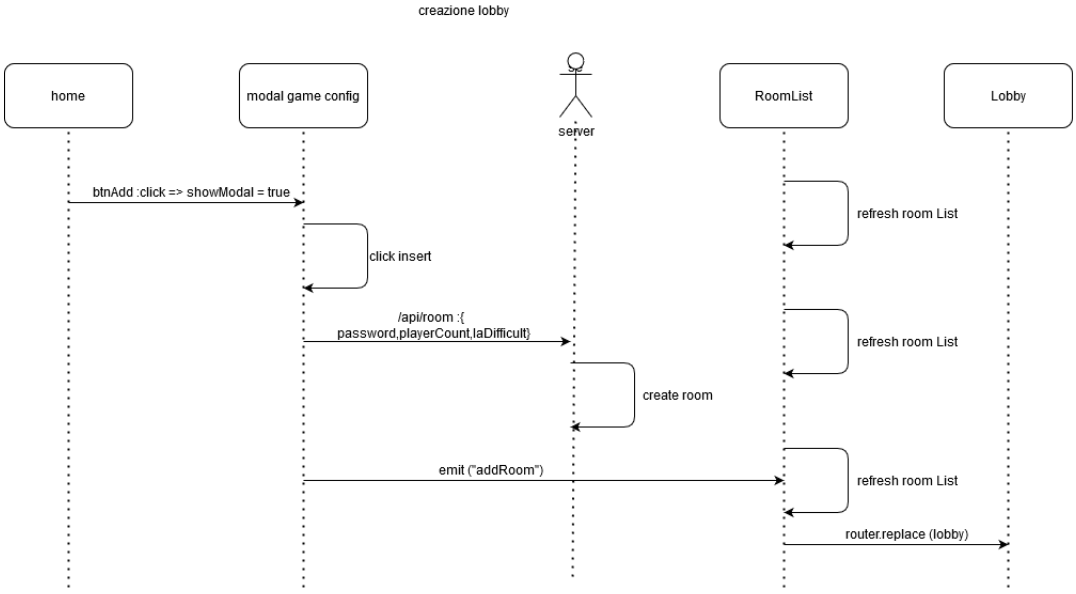

Figure 5: Diagramma di sequenza per la creazione delle lobby

5.3 Lobby

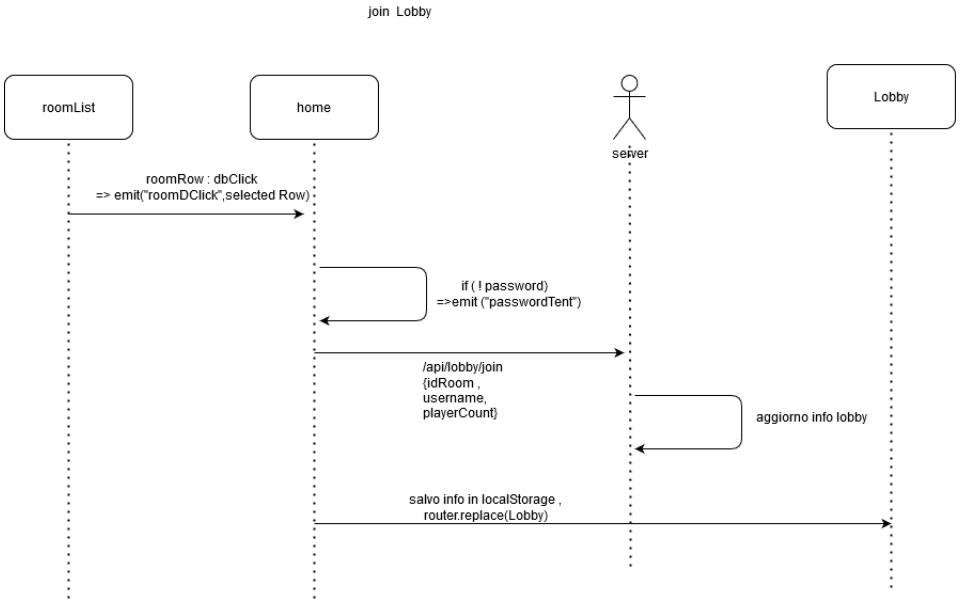

Figure 6: Diagramma di sequenza per l'entrata in una lobby

Nella lobby i giocatori attendono di iniziare la partita; all'interno di essa vengono visualizzate le informazioni sui giocatori attualmente presenti e una sezione in cui poter personalizzare i propri elementi estetici. Le informazioni sulla lobby e sui giocatori presenti vengono recuperate periodicamente dal database attraverso un apposito metodo getLobbyInfo() che si occupa di effettuare una chiamata all'apposita API.

```
getLobbyInfo() {
  axios.get("/api/lobby/info/" + localStorage.roomId)
  .then(response => {
    if(response.status == 204) {
      this.doLoop = false
      this.canExit = true
      Swal.fire({
        icon: 'warning',
        html: "La lobby e' stata chiusa",
        showCancelButton: false,
        focusConfirm: false,
        confirmButtonText:'OK'}).then(response => {
        router.replace({
          name: 'Home'
        }, () => router.go())
      })
    } else {
      var players = this.userInfo.length
      this.userInfo = []
      response.data.playersInfo.forEach((item, i) => {
        this.userInfo.push(item)
        this.playerJoined = players
        localStorage.playerCount = players+"/"+this.playerMaxCount
        if (response.data.started)
          this.pushToGame()
      })
    }
  })
}
```

Listing 2: Metodo usato per recuperare le informazioni della lobby

A seguito del recupero delle informazioni possono verificarsi differenti scenari:

- Un nuovo giocatore è entrato nella lobby, abilitando il pulsante adibito ad iniziare la partita al creatore della lobby
- Il proprietario della lobby ha iniziato la partita, avviando il trasferimento alla rotta Game

- L'altro giocatore ha chiuso la lobby, e se questi ne era il creatore ne provoca l'eliminazione, altrimenti le informazioni vengono semplicemente aggiornate

```
backPressure() {
      Swal.fire({
        icon: 'question',
        html: 'Vuoi davvero uscire dalla lobby?',
        showCancelButton: true,
        focusConfirm: false,
        confirmButtonText: 'Esci',
        cancelButtonText: 'Annulla',
      }).then((result) => {
        if (result.value) {
          if (localStorage.username == localStorage.roomLead) {
            axios.get("/api/lobby/close/" + localStorage.roomId).
                then(response => {})
          } else {
            axios.post("/api/lobby/leave", null, {
              params: {
                lobbyId: localStorage.roomId,
                leavingPlayer: localStorage.username,
                playerJoined: localStorage.roomLead
              }
            }).then(response => {})
          }
          this.canExit = true
          router.go()
        }
    })
  }
}
```

Listing 3: Metodo che gestisce l'uscita dalla lobby

Nel caso i cui un giocatore volesse uscire dalla lobby può farlo cliccando il tasto apposito oppure il tasto back del browser, aprendo un popup di dialogo (mediante il modulo Swal [1])in cui viene richiesta la conferma della propria azione; nel caso in cui ad uscire sia il creatore, la lobby viene eliminata attraverso una chiamata API.

5.4 Game

L'inizio della partita porta i giocatori al componente Game e crea un'istanza nel database relativa al gioco. Quest'ultima procedura viene eseguita lato server in seguito alla chiamata di una API.

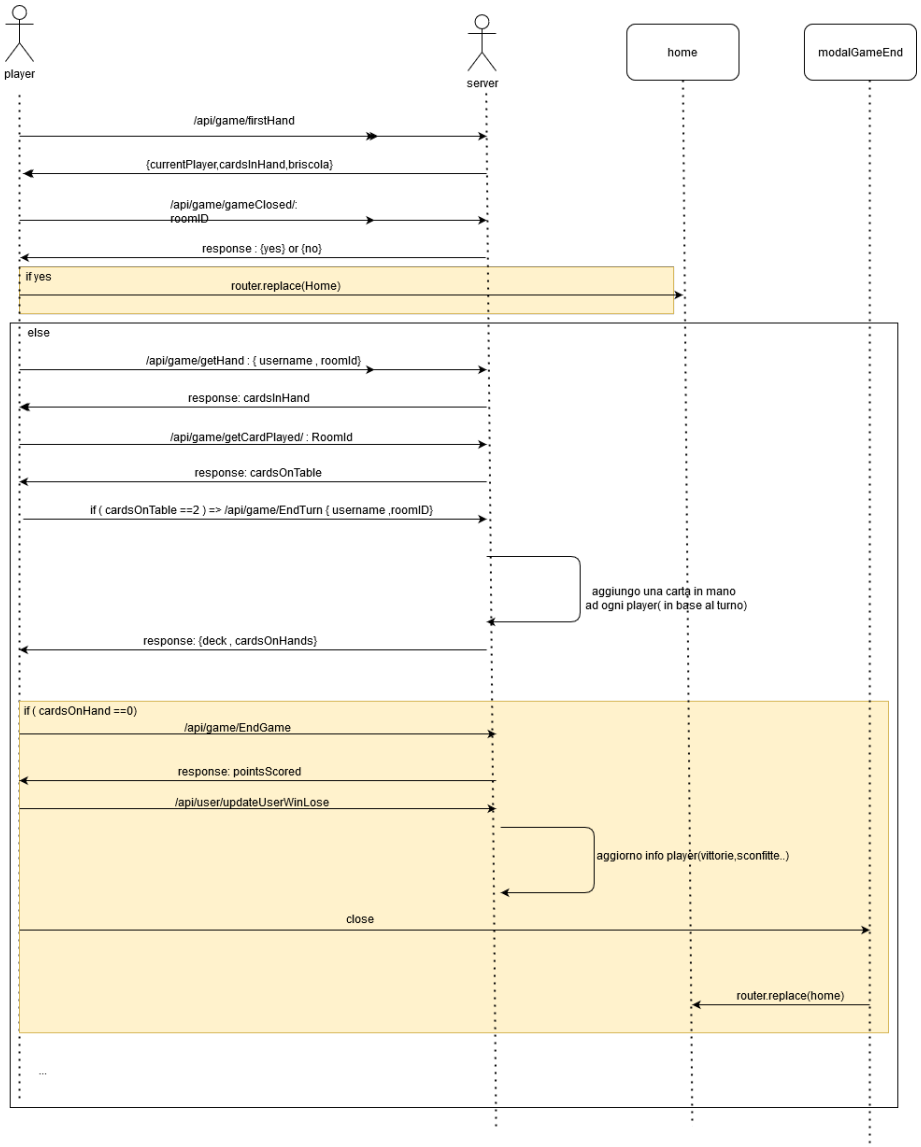

Figure 7: Diagramma di sequenza per la logica di gioco

```
exports.initGame = function(req, res) {
  //aggiorno il db con started
  Lobby.findOneAndUpdate({room_id: req.query.roomId}, {started:
      true}, {new: true}, (err, doc) => {
    if (err) {console.log("Something wrong when updating data!");}
  });

  //creo il deck
  var deck = []
  var deckSeed = ['b', 'c', 's', 'd']
  var i;
  for (i = 1; i < 11; i++)
    for (const seed of deckSeed)
      deck.push(cards = {value: i,seed: seed})

  deck = deck.sort(() => Math.random() - 0.5)
  var randomTurn = Math.floor(Math.random() * req.query.players.
      length)
  var randomBriscola = Math.floor(Math.random() * deck.length)
  var playersJson = []
  var initPoints = []
  req.query.players.forEach((item, i) => {
    playersJson.push(JSON.parse(item))
    initPoints.push({username: JSON.parse(item).user_name,points:
        0})
  });
  var new_Game = new Game({
    "game_id": req.query.roomId,
    "player_joined": playersJson,
    "current_player": playersJson[randomTurn].user_name,
    "deck": deck,
    "cards_on_table": [],
    "briscola": deck.shift(),
    "points_scored": initPoints,
    "end_clicked_count": 0,
    "cards_on_hand": [],
    "chat":[]
  });
  new_Game.save(function(err, game) {res.status(200).send({})})
}
```

Listing 4: Metodo che gestisce l'inizializzazione del gioco

5.4.1 Fase Iniziale

Dopo aver inizializzato la partita entrambi i giocatori pescano 3 carte dal mazzo e la relativa entry nel database viene aggiornata (cards_on_hand).

5.4.2 Fase Centrale

Periodicamente i giocatori controllano le carte sul tavolo di gioco a patto che la partita non sia conclusa o che l'altro giocatore non abbia abbandonato il gioco.

```
1  //prendo le info del tavolo (carte giocate-in mano)
2  getCardsOnTableLoop() {
3    var t = setInterval(() => {
4      if (!this.doLoop || this.gameEnded ) {
5        clearInterval(t)
6      } else {
7        //controllo se l'altro player ha quittato
8        axios.get("/api/game/gameClosed/" + localStorage.roomId).then
             (response => {
9          if(response.data>0 && !this.versusIa){
10           this.gameEnded = true
11           this.doLoop = false
12           Swal.fire({
13             icon: 'warning',
14             html: "L'avversario ha abbandonato la partita",
15             showCancelButton: false,
16             focusConfirm: false,
17             confirmButtonText: 'OK'
18           }).then(response => {
19             axios.get("/api/game/sendClick/" + localStorage.roomId)
                   .then(response => {
20               router.replace({name: 'Home'}, () => router.go())})
21           })
22         }
23       })
24       this.getCardsOnTable()
25     }
26   }, 1000)
27 }
```

Listing 5: Metodo il controllo delle carte in gioco

Durante questa fase i giocatori possono giocare le proprie carte dalla mano, inviando le informazioni sulla carta giocata al server, che si occuperà di aggiornare il database (carte giocate, assegnazione del turno, ecc.) e di effettuare i controlli necessari allo svolgimento del gioco (controllare se entrambi hanno

giocato una carta e in caso notificare la vittoria della mano e aggiornare i punti) attraverso il metodo sendCardPlayed().

5.4.3 Fase finale

Una volta giocate tutte e 40 le carte verrà lanciato il componente Modal-GameEnd in cui verranno visualizzate le informazioni relative alla partita: i punti ottenuti da ogni giocatore, l'esito della partita e l'esperienza guadagnata per salire di livello. Dopo che entrambi i giocatori (o solo uno nel caso contro l'IA) hanno cliccato il tasto "Chiudi" la partita viene chiusa e l'entry viene eliminata dal database.

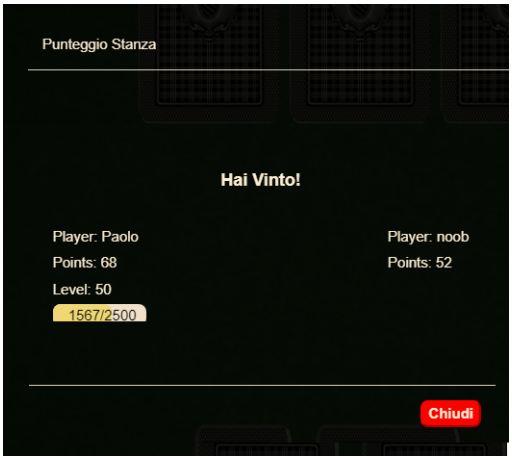

Figure 8: Schermata di fine gioco

5.4.4 Giocatore VS IA

Nel caso delle partite contro l'IA, prima della fase iniziale viene aggiunto al database un giocatore IA attraverso il metodo addIA. Una volta in partita, l'IA controlla periodicamente che sia il suo turno, e se lo è gioca una carta, che verrà scelta in base al livello di difficoltà deciso al momento di creazione della lobby. Sono presenti tre diversi livelli di difficoltà e il loro comportamento è deciso nel seguente modo:

- facile: ha il 50% di probabilità di giocare la carta "giusta", valutata nel seguente modo: se inizia l'IA lancerà il metodo getNotOptimalLoopEasy;

```
function getNotOptimalLoopEasy(doc){
  //gioco prima uno scartino non di briscola
  cardIndex = getWasteIndex(doc,false)
  //se non lo ho allora lancio una figura
  if(cardIndex ==-1)
  cardIndex = getFigureIndex(doc,false)
  //se non ho figure o scartini, allora cerco tra le briscole
  //(true indica la briscola)
  if(cardIndex ==-1)
  cardIndex = getWasteIndex(doc,true)
  if(cardIndex ==-1)
  cardIndex = getFigureIndex(doc,true)
  //se non trovo nulla  allora lancio la prima carta in mano
  if(cardIndex ==-1)
  cardIndex =0

  return cardIndex
}
```

Listing 6: Metodo la carta da giocare dall'IA facile se è lei ad iniziare

```
else {
  //se ha giocato un carico, rispondo con una briscola
  if(cardPlayer1.value =='1' || cardPlayer1.value =='3'){
    cardIndex = getBriscolaIndex(doc)
    if(cardIndex == -1)
    cardIndex = getNotOptimalLoopEasy(doc)
  } else {
    //cerco di rispondere con il carico della carta scelta
    cardIndex=getCaricoFromCard(doc,cardPlayer1)
    if(cardIndex == -1)
    //cerco di rispondere con una carta superiore dello stesso
         seme
    cardIndex=getUpperFromCard(doc,cardPlayer1)
    //se non ho una carta maggiore, faccio il solito loop
        scartino -> figura ->scartino briscola -> figura
        Briscola -> prima carta
    if(cardIndex == -1 )
    cardIndex = getNotOptimalLoopEasy(doc)
  }
  if(cardIndex == -1)
    cardIndex = getNotOptimalLoopEasy(doc)
  chosencard={username:doc.cardsOnHand[1].username , card: doc
      .cardsOnHand[1].cards.splice(cardIndex,1)[0]}
```

Listing 7: Metodo che gestisce la carta giocata se l'IA deve rispondere

- difficile: risponde alla stessa maniera della difficoltà facile ma con il 100% di probabilità di fare la scelta giusta; altrimenti, se deve giocare la prima carta, giocherà una carta del seme più giocato che non sia quello di briscola.

- god: risponde nello stesso modo della difficoltà difficile, ma se giocherà la prima carta, lo farà tenendo conto delle carte in mano all'avversario, in questo modo:

```
1  function getFirstCardGod(doc){
2    var cardsPlayer1 = doc.cardsOnHand[0].cards
3
4    var briscola = cardsPlayer1.filter(card => card.seed == doc.
         briscola.seed)
5    var carico =  cardsPlayer1.filter(card => ((card.value=='1'
         || card.value == '3')&& card.seed != doc.briscola.seed))
6    var cardIndex = -1
7    //se avversario non ha briscole, lancio carico o punti
8    if(briscola.length == 0){
9      cardIndex=doc.cardsOnHand[1].cards.findIndex(card => ( (
           card.value=='1' && card.seed != doc.briscola.seed)  ||
            (card.value == '3' && card.seed != doc.briscola.seed
           ) ))
10   if(cardIndex == -1)
11     cardIndex = doc.cardsOnHand[1].cards.findIndex(card => ((
           parseInt(card.value)>=8) && card.seed != doc.briscola.
           seed ))
12   }else {
13     //invio il seme che non ha l'avversario
14     cardIndex = getNotSeedP1(doc)
15     if(cardIndex == -1){
16       //se avversario ha un carico , non butto quel seme
17       if(carico.length>0)
18         cardIndex = getNoCaricoP1(doc,carico)
19     }
20   }
21   if(cardIndex == -1)
22     cardIndex= getFirstCardHard(doc)
23 return cardIndex
24 }
```

Listing 8: Metodo che gestisce la giocata della prima carta dall'IA God

5.4.5 Uscita dalla partita

Nel caso in cui un giocatore decida di uscire prematuramente dalla partita (cliccando l'apposito pulsante o il tasto back del browser) la partita viene interrotta ed eliminata dal database, notificando l'altro giocatore dell'evento.

5.5 Struttura delle rotte lato server

Le API esposte dal server sono organizzate sulla base del dominio su cui agiscono. Di seguito è presentata la specifica tecnica delle API.

UserAuth				
Metodo	**Rotta**	**Parametri**	**Risposta**	**Descrizione**
POST	/api/login	mail: String pass: String	statusCode	Controlla le credenziali di accesso
POST	/api/register	mail: String pass: String username: String	statusCode	Gestisce la registrazione controllando se esistono duplicati
GET	/session/id	//	session.userID	Risponde con lo username dell'utente loggato
GET	/api/session/logout	//	statusCode	Gestisce il logout

Lobby				
Metodo	**Rotta**	**Parametri**	**Risposta**	**Descrizione**
GET	/api/room	//	lobby:Lobby	restituisce tutte le lobby esistenti
POST	/api/room	userId:String playerCount:Number password: String iaDifficult: String	lobby:Lobby	gestise creazione di una nuova lobby
POST	api/lobby/join	idRoom:Number username:String	//	aggiorna il database inserendo il giocatore nella lobby specificata
GET	/api/lobby/info/ :roomId	roomId:Number	playersInfo: Player[] started:bool	Ritorna le informazioni sui player all'interno della room specificata e se questa è partita
GET	/api/lobby/close/ :lobbyId	lobbyId: Number	statusCode	Chiude la lobby specificata
POST	/api/lobby/leave/	playerJoined :Player lobbyId: Number	statusCode	Gestisce l'uscita dalla lobby
POST	/api/room/ matchPass/	password: String idRoom: Number	lobby: Lobby	Controlla il corretto inserimento della password per entrare in una lobby
POST	/api/user/ updateUserBackground/	username: String newImage: String	statusCode	Aggiorna lo sfondo giocatore
POST	/api/user/ updateUsertable/	username: String newImage: String	statusCode	Aggiorna il tavolo da gioco
POST	/api/user/ updateUserCardBack/	username: String newImage: String	statusCode	Aggiorna lo sfondo delle carte

POST	/api/user/ updateUserImg/	username: String newImage: String	statusCode	Aggiorna l'immagine profilo del giocatore
GET	/api/user/ refreshUserInfo/ :username	username: String	player: Player	Restituisce le informazioni del player inserito

Game				
Metodo	**Rotta**	**Parametri**	**Risposta**	**Descrizione**
POST	/api/game/initGame	roomId: Number players: Player[]	statusCode	Inizializza la partita
POST	/api/game/firstHand	roomId: Number username: String	cards: Card[] currentPlayer: String briscola: Card	Pesca tre carte e ritorna la briscola e il giocatore di turno
POST	/api/game/ sendCardPlayed	cardSelectedIndex: Number username: String roomId: Number	player di turno	Invia al server la carta giocata e gestisce la logica di gioco per il turno
GET	/api/game/ getCardPlayed/:roomId	roomId: Number	cardsOnTable: Card[] currentPlayer: String deckLength: Number	Recupera le informazioni sul gioco relative al turno attuale
POST	/api/game/getHand	roomId: Number username: String	cardsInHand: Card[]	Restituisce le carte in mano del giocatore
POST	/api/game/endTurn	roomId: Number username: String	cards: Card[]	Conclude il turno e restituisce le carte aggiornate

POST	/api/game/ endGame/	roomId: Number	points: Points	Gestisce la fine del gioco e restiyuisce i punti fatti
GET	/api/game/ gameClosed/ :roomId	roomId: Number	endClicked Count: Number	Restituisce il numero di giocatori che ha chiuso la partita
GET	/api/game/ sendClick/ :roomId	roomId: Number	game: Game	Incrementa il numero di giocatori che ha chiuso la partita
GET	/api/game/ gameClosed/:roomId	roomId: Number	endClicked Count: Number	Restituisce il numero di giocatori che ha chiuso la partita e gestisce l'eliminazione della lobby e del game

UserInfo				
Metodo	**Rotta**	**Parametri**	**Risposta**	**Descrizione**
POST	/api/user/ updateUserWinLose	userWin: Boolean username: String	statusCode	Aggiorna le informazioni utente relative alle vittorie e sconfitte
POST	/api/user/ refreshUserLevel	points: Number username: String	user: User	Aggiorna i punti e il livello del player
GET	/api/user/getLead	//	users: User[]	Ritorna i primi dieci user in classifica
GET	/api/user/ userInfo/:username	username: String	playerInfo: User	Ritorna le informazioni dell'utente

IA				
Metodo	**Rotta**	**Parametri**	**Risposta**	**Descrizione**
POST	/api/game/addIA	roomId: Number	statusCode	Inizializza l'IA e la inserisce in partita
POST	/api/game/IA/firstHand	roomId: Number	statusCode	Pesca le carte dell'IA e aggiorna il database
POST	/api/game/IA/ sendCardIA	roomId: Number	endTurn: Boolean nextTurn: String	Gestisce la carta giocata dall'IA
POST	/api/game/IA/ clearTable/	roomId: Number	statusCode	Svuota il tavolo se l'IA ha giocato per ultima

Chat				
Metodo	**Rotta**	**Parametri**	**Risposta**	**Descrizione**
POST	/api/game/sendMessage	roomId: Number username: String icon: String message: String	chat: Chat	Invia il messaggio nella room specificata
GET	/api/game/ getMsgGame/:roomId	roomId: Number	chat: Chat	Ritorna i messaggi inviati nella partita specificata
POST	/api/user/ sendGlobalMessage	username: String icon: String message: String	chat: Chat	Invia il messaggio nella chat globale
GET	/api/user/ getGlobalMsg/	//	chat: Chat	Ritorna gli ultimi messaggi inviati nella chat globale

6 Test

Le funzionalità del'applicativo sono state testate sia su browser Chrome che su browser Firefox allo scopo di garantire la portabilità. In particolare, sono state testate funzionalità complesse in real-time per verificare il loro funzionamento in entrambi i browser.

6.1 Usability Test

Il sistema, una volta completato, è stato fatto testare da diverse categorie di utenti, con vari livelli di esperienza nel gioco, allo scopo di testare vari aspetti dell'applicativo e capire se sono presenti elementi migliorabili in futuro. Agli utenti sono stati sottoposti vari task (gli stessi per entrambi) con l'obiettivo di portarli a termine in autonomia quando possibile:

- **Task 1:** Registrazione al sito
- **Task 2:** Logout
- **Task 3:** Login
- **Task 4:** Entrata in una lobby con password precedentemente creata da un membro del team di sviluppo
- **Task 5:** Uscita dalla lobby
- **Task 6:** Creazione di una lobby contro l'IA con difficoltà a scelta
- **Task 7:** Svolgimento della partita

Di seguito viene riportato un sunto dei risultati ottenuti.

Utente A: giovane con poca esperienza nel gioco Questo utente è abbastanza esperto nell'uso di dispositivi elettronici ed applicativi web, ed è riuscito agilmente a completare tutti i task richiesti con semplicità. Vista la sua poca esperienza nel gioco ha scelto la difficoltà facile, trovando adeguata la sfida.
Volendo testare anche gli altri livelli di IA il team ha fatto giocare all'utente una partita anche a livello difficile, che si è rilevata essere effettivamente ostica per il livello di esperienza dell'utente.
L'utente ha evidenziato due consigli che secondo lui possono migliorare l'esperienza di gioco:

- Rendere più immediata la comprensione del giocatore di turno
- Introdurre delle animazioni per evidenziare chi è il giocatore che vince la mano

Utente B: anziano molto esperto nel gioco Questo utente non è molto avvezzo all'utilizzo di applicativi web e dispositivi elettronici ma vanta una grande esperienza nel gioco della briscola. L'utente ha riscontrato difficoltà ad eseguire il task 2 poiché non è riuscito subito ad individuare la sezione i cui effettuare il logout, ma dopo che è stato guidato nel profilo è riuscito nell'intento. Anche nel task 4 ci sono state delle complicazioni perché l'utente effettuava un click singolo anziché uno doppio. Nel task 5 l'utente anziché premere il tasto apposito o il tasto back del browser ha premuto il tasto X del browser, consigliandoci di rendere più chiara l'icona o con l'inserimento di una scritta apposita.
Nel gioco ha rilevato inizialmente le stesse problematiche dell'utente A, ma una volta consigliato è riuscito a giocare senza troppi problemi. L'utente ha scelto la difficoltà God, perdendo la prima partita di pochi punti, cosa che ha fatto sì che il team lo abbia fatto giocare di nuovo, ottenendo una vittoria alla terza partita, seppur con un scarto di pochi punti.
L'utente si è dimostrato soddisfatto dall'intelligenza artificiale, che è riuscita a batterlo nonostante la sua esperienza.

7 Deployment

- Clonare il repository: https://github.com/LorenzoSimoncini2/Brisconline.git
- Dalla cartella principale del progetto avviare il server: node app.js
- Collegarsi al sito: http://localhost:3000/

8 Conclusioni

I risultati dei test effettuati hanno portato il team alle seguenti considerazioni:

- L'applicazione si comporta come da previsioni e il funzionamento generale percepito dagli utenti è stato ritenuto in linea con le aspettative.
- Per quanto riguarda la user interface in futuro si potrebbe puntare a migliorare la leggibilità di alcuni elementi e ad introdurre delle animazioni.
- Dal lato delle funzionalità, sicuramente una delle priorità è l'aggiunta della modalità 2vs2; altri aspetti emersi dal confronto con gli utenti sono l'introduzione di una chat privata, un sistema di amicizie e inviti, implementazione delle emote da inviare durante le partite, un ampliamento degli elementi customizzabili e l'ottenimento di alcuni con modalità speciali (ad esempio al raggiungimento di un certo numero di partite vinte o completate).
- Come aspetti tecnici da inserire si può prevedere la conferma dell'account tramite e-mail e la modifica delle informazioni dell'utente (tra cui la password) con eventuale conferma e-mail.

Commenti finali

A posteriori, l'utilizzo del tool Vue CLI avrebbe portato ad avere un codice meglio organizzato, più leggibile ed estendibile, tuttavia considerando ciò che abbiamo visto a lezione e il tempo limitato a disposizione per lo sviluppo si è deciso di optare per la versione classica di Vue. L'idea del progetto è nata poiché si era alla ricerca di un modo per giocare a briscola online che fosse semplice ed immediato, con la possibilità di allenarsi anche contro un'IA nel caso in cui gli amici non fossero presenti.

References

[1] Sweet Alert. https://sweetalert2.github.io/.

[2] Crypto-js. https://www.npmjs.com/package/crypto-js.