```
const Sequelize = require('sequelize');
const {log, biglog, errorlog, colorize} = require("./out");

const {models} = require('./model');


/**
 * Muestra la ayuda.
 *
 * @param rl Objeto readline usado para implementar el CLI.
 */
exports.helpCmd = (socket,rl) => {
    log(socket,"Commandos:");
    log(socket,"  h|help - Muestra esta ayuda.");
    log(socket,"  list - Listar los quizzes existentes.");
    log(socket,"  show <id> - Muestra la pregunta y la respuesta el quiz indicado.");
    log(socket,"  add - Añadir un nuevo quiz interactivamente.");
    log(socket,"  delete <id> - Borrar el quiz indicado.");
    log(socket,"  edit <id> - Editar el quiz indicado.");
    log(socket,"  test <id> - Probar el quiz indicado.");
    log(socket,"  p|play - Jugar a preguntar aleatoriamente todos los quizzes.");
    log(socket,"  credits - Créditos.");
    log(socket,"  q|quit - Salir del programa.");
    rl.prompt();
};


/**
 * Lista todos los quizzes existentes en el modelo.
 *
 * @param rl Objeto readline usado para implementar el CLI.
 */
exports.listCmd = (socket,rl) => {


    models.quiz.findAll()
        .then(quizzes => {
            quizzes.forEach(quiz =>{
                log(socket,`[${colorize(quiz.id, 'magenta')}]: ${quiz.question}`);
            });
        })
        .catch(error =>{
            errorlog(socket,error.message);
        })
        .then(() =>{
            rl.prompt();
        });


};




/**
 * Esta función devuelve una promesa que:
 *  -Valida que se ha introducido un valor para el parámetro.
 *  -Convierte el parámetro en un número entero
 *  Si todo va bien, la promesa se satisface y devuelve el valor de d a usar.*/

  const validateId = id => {
      return new Promise((resolve, reject) => {
          if (typeof id === "undefined" ){
              reject(new Error(`Falta el parámetro <id>.`));
          }else{
              id = parseInt(id);  // coger la parte entera y descartar lo demás
              if (Number.isNaN(id)){
                  reject(new Error(`El valor del parámetro <id> no es un número.`));
              }else{
                  resolve(id);
              }
          }
      });
};


 /**
  * Muestra el quiza indicado
 * @param rl Objeto readline usado para implementar el CLI.
 * @param id Clave del quiz a mostrar.
 */
exports.showCmd = (socket,rl, id) => {
    validateId (id)
        .then(id=> models.quiz.findById(id))
        .then(quiz=>{
            if(!quiz){
                throw new Error( `No existe un quiz asociado al id=${id}.`);
            }
            log(socket,`[${colorize(quiz.id, 'magenta')}]: ${quiz.question} ${colorize('=>', 'magenta')} ${quiz.answer}`);
        })
        .catch(error =>{
            errorlog(socket,error.message);
        })
        .then(()=> {
            rl.prompt();
        });
};

const makeQuestion = (rl, text) => {
    return new Sequelize.Promise((resolve, reject) => {
        rl.question(colorize(text, 'red'), answer => {
          resolve(answer.trim());
        });
    });
};

/**
 * Añade un nuevo quiz al módelo.
 * Pregunta interactivamente por la pregunta y por la respuesta.
```

```
 *
 * Hay que recordar que el funcionamiento de la funcion rl.question es asíncrono.
 * El prompt hay que sacarlo cuando ya se ha terminado la interacción con el usuario,
 * es decir, la llamada a rl.prompt() se debe hacer en la callback de la segunda
 * llamada a rl.question.
 *
 * @param rl Objeto readline usado para implementar el CLI.
 */
exports.addCmd = (socket,rl) => {

   makeQuestion(rl, 'Introduzca una pregunta: ')
       .then(q => {
           return makeQuestion(rl, 'Introduzca la respuesta: ')
               .then(a => {
                   return {question: q, answer: a};
               });
       })
       .then(quiz => {
           return models.quiz.create(quiz);
       })
       .then((quiz) => {
           log(socket,`${colorize('Se ha añadido', 'magenta')}: ${quiz.question} ${colorize('=>', 'magenta')} ${quiz.answer}`);
       })
       .catch(Sequelize.ValidationError, error => {
           errorlog(socket,'El quiz es erróneo:');
           error.errors.forEach(({message}) =>  errorlog(message));
       })
       .catch(error => {
           errorlog(socket,error.message);
       })
       .then(() => {
           rl.prompt();
       });
};


/**
 * Borra un quiz del modelo.
 *
 * @param rl Objeto readline usado para implementar el CLI.
 * @param id Clave del quiz a borrar en el modelo.
 */
exports.deleteCmd = (socket,rl, id) => {
    validateId(id)
        .then(id => models.quiz.destroy({where: {id}}))
        .catch(error => {
            errorlog(socket,error.message);
        })
        .then(() => {
            rl.prompt();
        });
};


/**
 * Edita un quiz del modelo.
 *
 * Hay que recordar que el funcionamiento de la funcion rl.question es asíncrono.
 * El prompt hay que sacarlo cuando ya se ha terminado la interacción con el usuario,
 * es decir, la llamada a rl.prompt() se debe hacer en la callback de la segunda
 * llamada a rl.question.
 *
 * @param rl Objeto readline usado para implementar el CLI.
 * @param id Clave del quiz a editar en el modelo.
 */
exports.editCmd = (socket,rl, id) => {
    validateId(id)
        .then(id => models.quiz.findById(id))
        .then(quiz => {
            if (!quiz){
                throw new Error(`No existe un quiz asociado al id= ${id}.`);
            }

            process.stdout.isTTY && setTimeout(() => {rl.write(quiz.question)}, 0);
            return makeQuestion(rl, 'Introduzca la pregunta : ')
                .then(q => {
                    process.stdout.isTTY && setTimeout(() => {rl.write(quiz.answer)}, 0);
                    return makeQuestion( rl, 'Introduzca la respuesta: ')
                        .then(a => {
                            quiz.question = q;
                            quiz.question = a;
                            return quiz;
                        });
                });
        })
        .then(quiz => {
            return quiz.save();
        })
        .then(quiz => {
            log(socket,`Se ha cambiado el quiz ${colorize(quiz.id, 'magenta')} por: ${quiz.question} ${colorize('=>', 'magenta')} ${quiz.answer}`);

        })
        .catch(error => {
            errorlog(socket,error.message);
        })
        .then(() => {
            rl.prompt();
        });
};


/**
 * Prueba un quiz, es decir, hace una pregunta del modelo a la que debemos contestar.
 *
 * @param rl Objeto readline usado para implementar el CLI.
 * @param id Clave del quiz a probar.
 */
exports.testCmd = (socket,rl, id) => {
    validateId(id)
        .then(id => models.quiz.findById(id))
        .then(quiz => {
            if (!quiz) {
```

```
                throw new Error(`No existe un quiz asociado al id= ${id}.`);
            }
            return makeQuestion(rl, `${quiz.question}? `)
                    .then(a => {
                    if (a.toLocaleLowerCase().trim() === quiz.answer.toLocaleLowerCase()) {
                        log(socket,'Su respuesta es correcta.');
                        biglog(socket,'Correcta', 'green');
                    } else {
                        log(socket,'Su respuesta es incorrecta.');
                        biglog(socket,'Incorrecta', 'green');
                    }
                });
        })
        .catch(error => {
            errorlog(socket, error.message);
        })
        .then(() => {
            rl.prompt();
        });
}

/**
 * Pregunta todos los quizzes existentes en el modelo en orden aleatorio.
 * Se gana si se contesta a todos satisfactoriamente.
 *
 * @param rl Objeto readline usado para implementar el CLI.
 */
exports.playCmd = (socket,rl) => {
    let numCorrectos = 0;
    let left = [];
    models.quiz.findAll()
        .then(quizzes => {
            quizzes.forEach(quiz => {
                left.push(quiz);
            });
            const runOne = () => {
                if (left.length == 0) {
                    log(socket,'No hay nada más que preguntar.', 'green');
                    log(socket,'Fin del juego. Aciertos:' + `${numCorrectos}`);
                    biglog(`${numCorrectos}`, 'magenta');
                    rl.prompt();
                } else {
                    let index = Math.floor(Math.random()*left.length);
                    let randomId = left[index];
                    return makeQuestion(rl, `${randomId.question}? `)
                        .then(a => {
                            if (a.toLocaleLowerCase().trim() === randomId.answer.toLocaleLowerCase().trim()) {
                                numCorrectos = numCorrectos + 1;
                                left.splice(index, 1);
                                log(socket,'CORRECTO -  Lleva ' + `${numCorrectos}` + ' aciertos');
                                runOne();
                            } else {
                                log(socket,'INCORRECTO');
                                log(socket,'Fin del juego. Aciertos ' + `${numCorrectos}`);
                                biglog(socket,`${numCorrectos}`, 'magenta');
                            }
                        })
                        .catch(error => {
                            errorlog(socket,error.message);
                        })
                        .then(() => {
                            rl.prompt();
                        });

                }
            }
            runOne()
        });

    };
    ////////////////////////////////////////////////////////////////////////////////////////////////////////




        ///////////////////////////////////////////////////////////////////////////////////////////




    /**
     * Muestra los nombres de los autores de la práctica.
     *
     * @param rl Objeto readline usado para implementar el CLI.
     */
    exports.creditsCmd = (socket,rl) => {
        log(socket,'Autor de la práctica:');
        log(socket,'Gonzalo Ignacio Hidalgo García', 'green');
        //log('feat: Carita', 'green');
        rl.prompt();
    };


    /**
     * Terminar el programa.
     *
     * @param rl Objeto readline usado para implementar el CLI.
     */
    exports.quitCmd = (socket,rl) => {
        rl.close();
        socket.end();
    };
```