Vanessa Sellart - Vicenç Parera - Sílvia Ruano - Óscar Parada - Iván Cedó

Memòria Critical Design Review

1. Objectiu

En el Critical Design Review es crea una aplicació Client/Servidor similar a l'Atenea, pensada per facilitar als estudiants l'accés a informació com els treballs a entregar (*tasks*), els horaris de classe (*timetables*) i les notes obtingudes (*marks*). La idea és desenvolupar una eina pràctica i funcional que permeti gestionar aquesta informació d'una manera senzilla i directa.

2. Servidor

El servidor, programat en NodeJs (JavaScript), s'encarregarà de gestionar les peticions enviades pel client i accedir a la informació de la base de dades MySQL, on es troben les dades sobre els treballs, els horaris i les notes. La comunicació entre el client i el servidor es farà a través del protocol HTTP GET i les dades es retornen en format JSON.

2.1. MySQL - Base de dades

Pel desenvolupament i gestió de la base de dades fem servir MySQL. Ens l'hem instal·lat des de la pàgina web oficial, assegurant-nos de tenir la versió més recent i compatible amb les nostres necessitats. A l'hora de treballar amb les dades, hem preferit utilitzar l'eina MySQL Workbench, ja que ens proporciona una interfície gràfica molt intuïtiva que permet veure de manera visual les taules creades i la seva estructura. Facilitant així el procés de disseny i manteniment de la base de dades.

Inicialment, vam començar creant una taula pels estudiants, i per les altres 3 taules (treballs, notes i horaris) una de cada exclusiva per a cada estudiant. Aquesta configuració, tot i funcional, resultava ineficient i complicava la gestió i escalabilitat del sistema. Per aquest motiu, vam decidir unificar les dades en 3 taules principals: *tasks, marks* i *timetables*. Per tal de mantenir la informació associada a cada estudiant, vam afegir una columna extra, *student_id*, que serveix per identificar a quin estudiant pertany cada registre.

El resultat final és una estructura de quatre taules:

- **Students**: Conté la informació bàsica dels estudiants, incloent-hi el seu UID, que s'utilitza per identificar-los, lligat al nom i cognoms de l'estudiant.
- **Tasks**: Emmagatzema els treballs a entregar, amb el nom de l'entrega, la data i l'assignatura a la qual pertany.
- **Marks**: Registra les notes dels estudiants, amb l'assignatura, el nom del treball o examen i la nota d'aquest.
- **Timetables**: Inclou la informació dels horaris de classe, amb el dia de la setmana, hora, assignatura i aula.

A més, vam haver d'afegir una columna extra a totes les taules (taula_id) que s'autoincrementés per tal de tenir un element únic per a cada línia i així poder-li adjudicar que fos la *primary-key*, ja que si no l'assignàvem no podíem editar la taula.

Totes aquestes taules es troben dins el *schema* anomenat pbe, que hem creat específicament per al projecte, nom de la base de dades que haurem d'incloure en el codi del servidor.

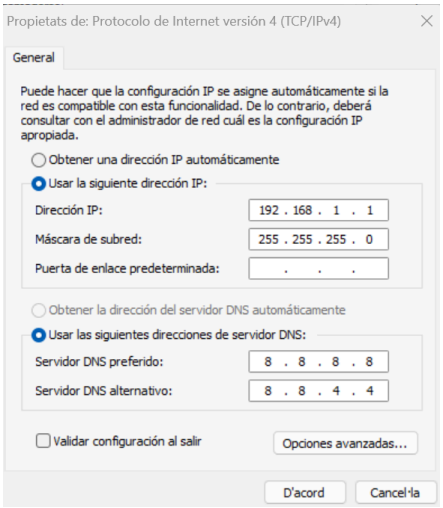
Exemple, amb la taula tasks, de com hem creat les taules:

```
INSERT INTO pbe.tasks (student_id, date, subject, name)
VALUES
--
('060FFBB0','2024-12-03','PBE',"Critical Design Review"),
('060FFBB0','2024-11-29','DSBM',"Estudi Previ 4"),
('060FFBB0','2024-12-11','PBE',"Final Report"),
('060FFBB0','2024-12-02','DSBM',"Practica 4"),
--
('13B67606','2024-12-03','PBE',"Critical Design Review"),
('13B67606','2024-11-29','DSBM',"Estudi Previ 4"),
('13B67606','2024-12-11','PBE',"Final Report"),
('13B67606','2024-12-02','DSBM',"Practica 4"),
--
('B46BE9D0','2024-12-03','PBE',"Critical Design Review"),
('B46BE9D0','2024-11-29','DSBM',"Estudi Previ 4"),
('B46BE9D0','2024-12-11','PBE',"Final Report"),
('B46BE9D0','2024-12-02','DSBM',"Practica 4"),
('B46BE9D0','2024-12-05','RP',"Practica 5"),
--
```

```
CREATE TABLE `pbe`.`tasks` (
    `id_tasks` INT NOT NULL AUTO_INCREMENT,
    `student_id` VARCHAR(45) NOT NULL,
    `date` DATE, -- Se agregó la coma
    `subject` VARCHAR(45) NOT NULL,
    `name` VARCHAR(45) NOT NULL,
    PRIMARY KEY (`id_tasks`),
    INDEX `idxm_student_id` (`student_id` ASC),
    CONSTRAINT `fk_tasks_student`
        FOREIGN KEY (`student_id`)
        REFERENCES `pbe`.`students` (`student_id`)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION
);
```

2.2. IPs

Per implementar l'aplicació Client/Servidor, era imprescindible conèixer les IPs de la connexió tant del client com del servidor, ja que són necessàries per establir la comunicació entre els dos dispositius. Durant el desenvolupament, alguns membres del grup ens vam trobar amb un problema: la connexió de la Raspberry Pi feia que aquesta utilitzés IPs dinàmiques. Això significava que cada vegada que connectàvem la Raspberry, la seva IP canviava, obligant-nos a actualitzar el codi de l'aplicació constantment per adaptar-nos a la nova IP.



Per solucionar aquest problema i facilitar el treball, vam decidir implementar IPs estàtiques a través d'un pont de xarxa. Aquest procés va consistir en els següents passos:

1. **Configuració del pont de xarxa**:
    - En les propietats del pont de xarxa es van modificar les propietats TCP/IPv4 del mateix per assignar-li una IP estàtica: 192.168.1.1. Aquesta IP serà la que l'ordinador utilitzarà com a punt d'accés.
2. **Configuració de la Raspberry Pi amb IP estàtica**:
    - A la Raspberry, es va modificar el fitxer /etc/network/interfaces per configurar la interfície eth0 amb una IP estàtica. La configuració inclou la següent IP: 192.168.1.2, assegurant que aquesta IP sigui sempre la mateixa.
    - Després de fer aquests canvis, es va reiniciar la Raspberry perquè la configuració tingués efecte.

3. **Comprovació de la configuració**:
    - Per verificar que la IP s'havia assignat correctament al pont de l'ordinador, es va utilitzar la comanda ipconfig a Windows, comprovant que el pont de xarxa tenia la IP 192.168.1.1.
    - A la Raspberry, es va executar la comanda ip addr show eth0 per confirmar que la interfície eth0 tenia assignada la IP estàtica 192.168.1.2.
    - Finalment, es van dur a terme proves de connectivitat entre l'ordinador i la Raspberry mitjançant *pings* en ambdues direccions, comprovant que la comunicació era correcta.

Amb aquesta configuració, vam solucionar el problema de les IPs dinàmiques i vam establir una connexió estable i previsible entre l'ordinador i la Raspberry Pi, cosa que ens va permetre continuar amb el desenvolupament del projecte sense interrupcions.

2.3. Codi

```JavaScript
const http = require('http');
const mysql = require('mysql2');
const url = require('url'); //necessari per tractar els paràmetres de
consulta

// Configuració de la connexió amb la base de dades
const connection = mysql.createConnection({
  host: '192.168.1.1', // IP de l'ordinador
  user: 'root',         // Usuari de MySQL
  password: '1234',     // Contrasenya de MySQL
  database: 'pbe',      // Base de dades
  port: 3306            // Port de MySQL
});

// Verificar la conexión a la base de datos
connection.connect((err) => {
  if (err) {
    console.error('Error de conexión a la base de datos: ' + err.stack);
    return;
  }
console.log('Conectado a la base de datos con el ID ' +
connection.threadId);
});

//Gestionar sessions d'usuaris
const userSessions = {};
//temps d'inactivitat en mil·lisegons (2 minut)
const SESSION_TIMEOUT = 2*60*1000;
// Crear el servidor HTTP
const server = http.createServer((req, res) => {
```

```
  res.setHeader('Content-Type', 'application/json');
  res.setHeader('Access-control-Allow-Origin', '*');

  const parsedUrl = url.parse(req.url, true); //parsear la URL

    if(req.method === 'GET' && parsedUrl.pathname === '/authenticate'){
        const {uid} = parsedUrl.query; //parsejar el cos de la solicitud per
obtenir uid
        if(!uid){
            res.statusCode = 400;
            res.end(JSON.stringify({error: 'UID es necessari'}));
            return;
        }
        //consultar students per buscar uid
       connection.query(
         'SELECT name FROM students WHERE student_id = ?', //consulta MySQL
          [uid], // Parametre: UID rebut
          (err, results) => {
              if(err){ //control d'errors de consulta
                    console.error('Error al realitzar consulta:', err);
                    res.statusCode = 500; //codi d'error del servidor
                    res.end(JSON.stringify({error:'Error al realitzar la
consulta'}));
              }else if(results.length===0){
                    //si no es troba a la taula
                    res.statusCode = 401; // Codi "no autoritzat"
                    res.end(JSON.stringify({error: 'UID no vàlid'}));
              }else{
                    //si uid vàlid, tornar el nom de l'estudiant
                    userSessions[uid]={
                        name: results[0].name,
                       lastActivity: Date.now() //guarda hora de l'activitat
                    };
                    res.statusCode = 200; // codi d'exit
                    res.end(JSON.stringify({name: results[0].name}));
                }
            }
        );
        //endpoint per realitzar consultes taules concretes
    }else if(req.method === 'GET' && parsedUrl.pathname === '/query'){
        const {table, limit, ...filters} = parsedUrl.query; //parsejar cos
de la solicitud per obtenir la taula
        console.log("URL parseada final:", parsedUrl.href);
        //uid de la sessio
     const uid = Object.keys(userSessions).find((uid) => userSessions[uid]);

        if(!uid){
```

```
            res.statusCode = 600;
          res.end(JSON.stringify({error: 'Sessió no iniciada o caducada'}));
            return;
        }
        userSessions[uid].lastActivity=Date.now(); //actualitzar activitat

        //validar que la taula solicitada és vàlida
        const validTables = {
            tasks: ['date', 'subject', 'name'],
            timetables: ['day', 'hour', 'subject', 'room'],
            marks: ['subject', 'name', 'marks']
        };
        if(!table || !validTables[table]){
            res.statusCode = 400; // solicitud incorrecta
            res.end(JSON.stringify({error: 'Taula no vàlida/necessària'}));
            return;
        }
        //generar la consulta MySQL la taula solicitada filtrant per uid
        const selectedColumns = validTables[table].join(', '); //obtenir les
columnes específiques

        // --------- CONSTRAINTS ---------
      let query = `SELECT ${selectedColumns} FROM ${table} WHERE student_id
= ?`;
        const params = [uid];
        if(Object.keys(filters).length > 0){
            const conditions = [];
            const opMap = {
                gte: '>=',
                gt: '>',
                lte: '<=',
                lt: '<',
                eq: '='
            };

            for( const [key, value] of Object.entries(filters)){
                if(key.includes('[') && key.includes(']')){
                    //treure el camp i l'operador
                    const field = key.split('[')[0];
                    const modifier = key.match(/\[(.+)\]/)[1];
                    const op = opMap[modifier];
                    if(op){
                        conditions.push(`${field} ${op} ?`);
                        params.push(value);
                    }else{
                       console.warn(`Modificador desconocido: ${modifier}`);
                    }
```

```
                } else if (value.toLowerCase()==='now'){
                    if(table === 'timetables'){
                        if(key === 'day'){
                    const currentDay = new Date().toLocaleDateString('en-US',
{weekday: 'short'});
                            conditions.push(`${key} = ?`);
                            params.push(currentDay);
                        } else if (key === 'hour'){
                            // Convertir "now" a hora actual, pero solo con
la hora (sin minutos y segundos)
                           const currentTime = new Date();
                           currentTime.setMinutes(0); //minuts a 0
                           currentTime.setSeconds(0); //segons a 0
                          currentTime.setMilliseconds(0); // milisegons a 0
                           const roundedHour =
currentTime.toTimeString().split(' ')[0];  // Format HH:00:00
                            conditions.push(`${key} = ?`);
                            params.push(roundedHour);
                        }
                    } else if (table === 'tasks'){
                        const currentDate = new
                           Date().toISOString().split('T')[0];
                        conditions.push(`${key} = ?`);
                        params.push(currentDate);
                    } else {
                        console.warn(`Campo "now" no soportado para la tabla
${table}`);
                    }
                }else{
                    conditions.push(`${key} = ?`);
                    params.push(value);
                }
            }
            query += ' AND '+conditions.join(' AND ');
        }

        if(limit){
            query += ' LIMIT ?';
            params.push(parseInt(limit, 10));
        }

//---- Verificacions ----
        console.log("Consulta SQL generada:", query); // Verifica que la
consulta sea correcta
        console.log("Parámetros:", params); // Verifica que los parámetros
se pasen correctamente
```

```
 //consulta a la taula
        connection.query(query,params, (err, results) => {
            if (err){ //control d'errors de consulta
                console.error('Error al realitzar la consulta: ', err);
                res.statusCode = 500; // error del servidor
                res.end(JSON.stringify({error: 'Error al realitzar la
consulta'}));
            }else{
                //tornar les dades de la consulta
                res.statusCode = 200; //codigo de exito
                res.end(JSON.stringify(results));
            }
        });
    }else if(req.method === 'GET' && parsedUrl.pathname === '/logout'){
        const uid = Object.keys(userSessions).find((uid) =>
userSessions[uid]);
        if(!uid){
            res.statusCode = 400;
            res.end(JSON.stringify({ error: 'Sessió no iniciada' }));
            return;
        }
        //esborrar la sessio de l'usuari
            delete userSessions[uid];
            res.statusCode = 200;
            res.end(JSON.stringify({ message: 'Sessió tancada correctament'
}));
        // Ruta para obtener los estudiantes
    } else {
        res.statusCode = 404;
        res.end(JSON.stringify({ error: 'Ruta no encontrada' }));
    }
    });

//Timer per gestionar les sessions
setInterval(() => {
    const now = Date.now();

    for(const uid in userSessions){
        if(now - userSessions[uid].lastActivity > SESSION_TIMEOUT){
            console.log(`Sessió de l'usuari ${uid} ha caducat`);
            delete userSessions[uid];
        }
    }
}, 60 * 1000);   //Comprova l'activitat cada minut

// Escuchar en el puerto 3000
const PORT = 3000;
```

```
server.listen(PORT, '192.168.1.1', () => {
  console.log(`Servidor escuchando en http://192.168.1.1:${PORT}`);
});
```

Connexió amb la base de dades

Per a la comunicació amb la base de dades MySQL, s'utilitza el paquet mysql2, que vam haver de fer servir npm install mysql2 per tal d'instal·lar el paquet al projecte Node.js. La configuració de la connexió inclou els detalls essencials, com l'adreça IP del servidor (192.168.1.1), l'usuari (root), la contrasenya (1234), la base de dades (pbe) i el port de MySQL (3306). Un cop configurada, es comprova la connexió que retorna un error o informa que la connexió ha estat exitosa, incloent-hi el threadId associat. Aquesta connexió es reutilitza al llarg de tot el servidor per realitzar consultes a la base de dades.

Autentificar a l'usuari

Quan un client envia una sol·licitud GET al camí /authenticate, el servidor espera rebre un uid (identificador únic d'usuari) com a paràmetre. Aquest uid s'utilitza per consultar la taula students i comprovar si existeix a la base de dades.

Si el uid no és vàlid, es retorna un error (401 Unauthorized), però si és vàlid, es crea una sessió per a l'usuari. En lloc de necessitar el uid en futures peticions, el servidor l'emmagatzema en l'objecte userSessions. Això simplifica les interaccions posteriors, ja que el client només ha de gestionar la sessió i no reenviar el uid.

Sessió d'usuari amb Timer

Un cop l'usuari s'autentica, es crea una sessió associada al seu uid. Aquesta sessió conté informació com el nom de l'usuari i l'hora de l'última activitat. El servidor gestiona aquestes sessions amb un timer que s'executa cada minut. Si una sessió roman inactiva durant més de 2 minuts (SESSION_TIMEOUT), s'elimina automàticament. Això garanteix que les sessions no ocupin memòria innecessàriament i protegeix contra accessos indeguts.

Consulta a les taules (Query)

Quan un client vol consultar una taula específica (com tasks, marks o timetables), envia una sol·licitud GET al camí /query. El servidor valida que l'usuari tingui una sessió activa. En lloc de requerir el uid en la petició, el servidor utilitza el uid emmagatzemat a la sessió per filtrar les dades corresponents a l'usuari.

Cada taula només conté certes columnes rellevants:

- tasks: date, subject, name
- timetables: day, hour, subject, room
- marks: subject, name, marks

El servidor selecciona dinàmicament les columnes i genera una consulta SQL que retorna només les dades de l'usuari. Això assegura que cap estudiant pugui accedir a informació d'altres usuaris.

Filtrar les taules (Constraints)

Per fer les consultes més personalitzades, es poden aplicar filtres a les dades. Aquests filtres es defineixen amb modificadors com: gte (més gran o igual), lte (menor o igual), eq (igual), etc.

Per exemple, es pot obtenir informació de tasques a partir d'una data (date[gte]=2023-11-01) o horaris en un dia específic (day=Mon). També es pot usar el valor especial now per obtenir horaris de l'hora actual o tasques de la data d'avui, que si es tracta de la taula de tasques, l'interpreta com una data i a la taula d'horaris com el dia de la setmana.

Amb aquesta estructura, el servidor permet consultes personalitzades i segures, garantint que l'usuari només pugui accedir a la seva pròpia informació.

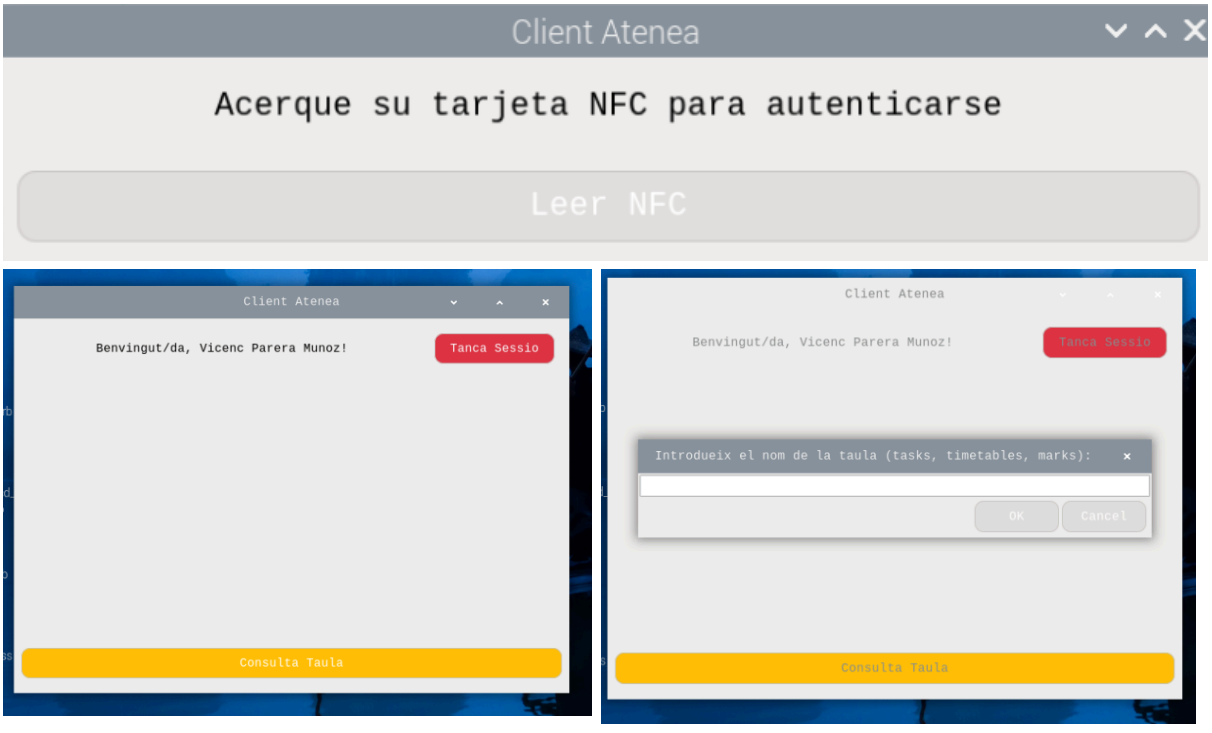
3. Client

El client tracta d'una aplicació gràfica desenvolupada en Ruby per a una Raspberry Pi, que serà la interfície que utilitzaran els estudiants. Per accedir al sistema, es farà servir un lector de targetes que identificarà cada estudiant amb el seu UID. També es mostraran missatges de *login* o d'error a una pantalla LCD per fer el procés més intuïtiu.

Per fer el client hem dissenyat una interfície amb gtk3 que té les següents pantalles:

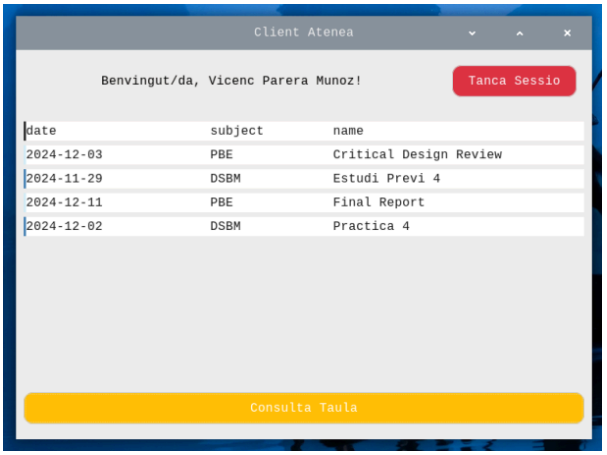
Autenticació:

Selecció:



Selecció de Filtres:

Mostratge de Taules:



El funcionament és senzill, simplement pases la targeta, i enviem el codi del NFC al servidor, el qual ens retorna el nom d'usuari de la persona en qüestió.

En cas de rebre un nom d'usuari, significa que la validació es correcta, i entrem a la finestra de selecció.

Un cop a la finestra de selecció podrem pulsar a consultar taula i escriure el nom de la taula que desitgem. Tot seguit s'obrirà una segona finestra preguntant quins filtres vols aplicar. si no en vols posar cap, prem "ok", la taula consultada juntament amb els filtres s'envien al servidor i aquest retorna les dades sol·licitades.

Quan el client reb les dades les mostra en un llistat de Text views. En cas que no hi hagi prou espai, el separador es converteix en "scrollable".

Pel que fa al format, hem posat cada botò d'un color diferent per poderlos diferenciar facilment. També hem posat la font en monospace perquè quedi més arreglada i ho hem desat tot en un fitxer .css, el qual cridem des del codi.

Pel que fa a la lcd, simplement hem executat un lcd print inicial que indica el passar la targeta, i altre amb un missatge de Benvingut #{name} un cop el servidor ha comprovat qui som.

Tant la connexió al servidor com la lectura del nfc les hem fet mitjançant fils auxiliars, ja que ho requereixen.

3.1. Codi Connexió

Aquesta primera versió del client en Ruby té com a objectiu validar la connexió amb el servidor i oferir una interfície interactiva mitjançant el terminal per autenticar usuaris i consultar dades. El codi permet que l'usuari introdueixi el seu UID per autenticar-se, realitzant una sol·licitud al servidor amb el camí /authenticate. Si l'autenticació és correcta, es retorna el nom de l'estudiant i s'estableix una sessió. Per simplificar la interacció, l'UID no cal ser enviat en cada sol·licitud posterior, ja que queda gestionat internament.

El client ofereix la funcionalitat de consultar dades de taules específiques com tasks, timetables i marks, permetent afegir filtres personalitzats. Aquests filtres segueixen el format camp=valor i es poden especificar múltiples condicions com intervals de dates o valors específics. Les consultes s'envien al servidor utilitzant el camí /query amb els paràmetres corresponents. Les respostes del servidor són processades en format JSON, mostrant els resultats o missatges d'error de manera clara.

Per a garantir un funcionament eficient, totes les sol·licituds HTTP GET es realitzen en fils auxiliars. Això permet que el client continuï operant sense bloquejar-se mentre espera la resposta del servidor. En cas d'errors de connexió o altres problemes, assegurant la robustesa de l'aplicació. Aquesta versió inicial facilita la validació del sistema, establint una base sòlida per desenvolupar una interfície gràfica en futures iteracions.

3.2. Codi Interfície

```JavaScript
require 'gtk3'
require 'json'
require 'net/http'
require 'uri'
require 'i2c/drivers/lcd'
require_relative 'puzzle1'

# Retiro la variable lcd de la part global i la defino dins de la classe

BASE_URL = 'http://192.168.1.1:3000'

def fetch_data(path, params = {})
  uri = URI("#{BASE_URL}#{path}")
  uri.query = URI.encode_www_form(params) unless params.empty?
  begin
    response = Net::HTTP.get_response(uri)
    if response.code.to_i == 200
      JSON.parse(response.body)
    else
      puts "Error del servidor: #{response.code} - #{response.body}"
```

```
      nil
    end
  rescue StandardError => e
    puts "Error al connectar amb el servidor: #{e.message}"
    nil
  end
end

class SimpleClientApp
  def initialize
    @window = Gtk::Window.new('Client Atenea')
    @window.set_size_request(800, 600)
    @window.signal_connect('destroy') { Gtk.main_quit }

    @vbox = Gtk::Box.new(:vertical, 10)
    @vbox.margin = 10
    @window.add(@vbox)

    # Crear la instància de la pantalla LCD dins de la classe
    @lcd = I2C::Drivers::LCD::Display.new('/dev/i2c-1', 0x27, rows=4,
cols=20)

    # Carregar els estils des d'un fitxer CSS extern
    load_css_from_file('styles.css')

    # Mostrar la pantalla d'autenticació
    show_authentication_screen

    @window.show_all
  end

   def show_authentication_screen
  @vbox.children.each(&:destroy) # Limpia la interfaz

  # Etiqueta de bienvenida inicial
  label = Gtk::Label.new('Acerque su tarjeta NFC para autenticarse')
  @vbox.pack_start(label, expand: false, fill: false, padding: 10)

  # Botón para iniciar la lectura NFC
  nfc_button = Gtk::Button.new(label: 'Leer NFC')
  nfc_button.signal_connect('clicked') do
    uid = read_uid
    if uid
      authenticate_user(uid)
    else
      show_error('No se pudo leer el UID. Intente nuevamente.')
    end
```

```
  end
  @vbox.pack_start(nfc_button, expand: false, fill: false, padding: 10)

  @window.show_all
end

def read_uid
    begin
      reader = Rfid.new
      uid = reader.read_uid
      puts "UID leído: #{uid}" # Debugging opcional
      uid
    rescue StandardError => e
      puts "Error leyendo NFC: #{e.message}"
      nil
    end
  end

  def show_main_interface(name)
    @vbox.children.each(&:destroy)

    # Crear un contenidor horitzontal per al text de benvinguda i el botó de
logout
    header_box = Gtk::Box.new(:horizontal, 5)

    # Mostrar un missatge de benvinguda a la pantalla LCD
    @lcd.clear
    @lcd.text('    Welcome', 1)
    @lcd.text("#{name}!", 2)

    welcome_label = Gtk::Label.new("Benvingut/da, #{name}!")
    header_box.pack_start(welcome_label, expand: true, fill: true, padding:
10)

    logout_button = Gtk::Button.new(label: 'Tanca Sessio')
    logout_button.set_name('logout_button')
    logout_button.signal_connect('clicked') { logout }
    header_box.pack_start(logout_button, expand: false, fill: false,
padding: 10)

    @vbox.pack_start(header_box, expand: false, fill: false, padding: 10)

    @text_views_box = Gtk::Box.new(:vertical, 2)
    scrolled_window = Gtk::ScrolledWindow.new
    scrolled_window.add(@text_views_box)
    scrolled_window.set_policy(:automatic, :automatic)
    @vbox.pack_start(scrolled_window, expand: true, fill: true, padding: 10)
```

```
    query_button = Gtk::Button.new(label: 'Consulta Taula')
    query_button.set_name('query_button')
    query_button.signal_connect('clicked') { query_table }
    @vbox.pack_start(query_button, expand: false, fill: false, padding: 10)

    @window.show_all
  end

  def authenticate_user(uid)
    result = fetch_data('/authenticate', { uid: uid })
    if result && result['name']
      show_main_interface(result['name'])
    else
      show_error_dialog('Error d\'autenticacio. Torna-ho a intentar.')
    end
  end

  def query_table
    table = prompt('Introdueix el nom de la taula (tasks, timetables,
marks):')
    return unless table

    filter_string = prompt('Introdueix els filtres (exemple:
subject=Math&date[gte]=2023-01-01):')
    filters = parse_filters(filter_string)

    params = { table: table }.merge(filters)
    result = fetch_data('/query', params)

    if result.is_a?(Array) && !result.empty?
      populate_text_views(result)
    elsif result.is_a?(Array) && result.empty?
      show_error_dialog("No hi ha dades disponibles per a la taula
#{table}")
    else
      show_error_dialog('Error: resposta inesperada')
    end
  end

  def populate_text_views(data)
    @text_views_box.children.each(&:destroy)

    # Càlcul de longitud màxima per columna
    columns = data.first.keys
    column_widths = columns.map { |col| [col.to_s.length, *data.map { |row|
row[col].to_s.length }].max }
```

```
    # Afegir capçaleres com el primer TextView
    headers = columns.each_with_index.map { |col, i|
col.to_s.ljust(column_widths[i]) }.join("\t\t")
    add_text_view(headers, 'header', 0)

    # Afegir les dades fila a fila
    data.each_with_index do |row, index|
      row_data = columns.each_with_index.map { |col, i|
row[col].to_s.ljust(column_widths[i]) }.join("\t\t")
      style_class = index.even? ? 'row_even' : 'row_odd'
      add_text_view(row_data, style_class, index + 1)
    end

    @window.show_all
  end

  def add_text_view(text, style_class, row_index)
    buffer = Gtk::TextBuffer.new
    buffer.text = text

    text_view = Gtk::TextView.new
    text_view.buffer = buffer
    text_view.editable = false
    text_view.cursor_visible = false
    text_view.set_name(style_class)
    @text_views_box.pack_start(text_view, expand: false, fill: false,
padding: 2)
  end

  def parse_filters(filter_string)
    return {} if filter_string.nil? || filter_string.empty?

    filters = {}
    filter_string.split('&').each do |filter|
      key, value = filter.split('=')
      filters[key.strip] = value.strip if key && value
    end
    filters
  end

  def show_error_dialog(message)
    dialog = Gtk::MessageDialog.new(
      parent: @window,
      flags: :destroy_with_parent,
      type: :error,
      buttons: :close,
```

```
      message: message
    )
    dialog.run
    dialog.destroy
  end

  def logout
    show_authentication_screen
  end

  def prompt(message)
    dialog = Gtk::Dialog.new(
      title: message,
      parent: @window,
      flags: :destroy_with_parent,
      buttons: [[Gtk::Stock::OK, :ok], [Gtk::Stock::CANCEL, :cancel]]
    )
    entry = Gtk::Entry.new
    dialog.child.add(entry)
    dialog.child.show_all

    response = dialog.run
    input = entry.text.strip
    dialog.destroy
    response == :ok && !input.empty? ? input : nil
  end

  def load_css_from_file(file_path)
    return unless File.exist?(file_path)
    provider = Gtk::CssProvider.new
    provider.load_from_path(file_path)
    Gtk::StyleContext.add_provider_for_screen(
      Gdk::Screen.default,
      provider,
      Gtk::StyleProvider::PRIORITY_USER
    )
  end
end

Gtk.init
SimpleClientApp.new
Gtk.main
```

Aquest codi implementa una aplicació gràfica utilitzant la llibreria GTK3 en Ruby, amb l'objectiu de crear una interfície d'usuari per a un sistema que combina autenticació mitjançant targetes NFC i consulta de dades des d'un servidor remot. La interfície gràfica es gestiona a través de la classe ‘SimpleClientApp’, que defineix una finestra principal amb una disposició vertical (‘Gtk::Box’) on

s'afegeixen els elements visuals de forma dinàmica segons les necessitats del programa. Al llarg del codi, es fan servir tècniques per mantenir l'interfície modular i flexible, com ara esborrar els components existents abans d'afegir-ne de nous.

En l'inici de l'aplicació, es mostra una pantalla d'autenticació que convida l'usuari a apropar una targeta NFC per validar la seva identitat. Aquesta pantalla inclou una etiqueta descriptiva i un botó que inicia la lectura del UID de la targeta. Quan l'usuari fa clic al botó, es crida al mètode 'read_uid', que intenta llegir el codi únic de la targeta mitjançant un lector NFC definit a través de la classe Rfid. Si la lectura té èxit, es crida al mètode 'authenticate_user' per validar el UID amb el servidor. En cas contrari, es mostra un missatge d'error.

Després de l'autenticació, si el servidor retorna un nom associat al UID, l'aplicació mostra la interfície principal, on es dona la benvinguda a l'usuari tant a la pantalla LCD com a la finestra. La integració de la pantalla LCD és un detall interessant, ja que utilitza el mòdul 'I2C::Drivers::LCD' per escriure missatges personalitzats, com ara 'Welcome' seguit del nom de l'usuari autenticat. Aquesta combinació de la interfície física (pantalla LCD) i gràfica (finestra GTK) crea una experiència rica i interactiva.

A més, la interfície principal inclou un botó per fer consultes a taules de la base de dades remota. L'usuari pot introduir el nom de la taula i filtres específics per personalitzar la consulta. Els resultats es mostren de manera tabular dins d'una àrea de desplaçament ('Gtk::ScrolledWindow') amb diferents estils per a files parells i imparells.

3.3. Codi Interfície - Lector Per Teclat

Hi ha un lector que funciona de forma diferent als altres, ja que funciona com si fos un teclat. Per a poder llegir les dades correctament: hem de gestionar el client de manera diferent. En primer lloc, necessitem una classe, la qual hem anomenat UID, que gestioni el UID que llegim, ja que el volem en hexadecimal i el lector ens el llegeix en decimal.

```JavaScript
class UID
    attr_accessor :id

    def initialize(id = nil)
        @id = id
    end

    def hex_uid
        return nil unless id && id =~ /^\d+$/

        decimal_id = id.to_i
        bytes_array = [decimal_id].pack('L').bytes.reverse
        decimal_array = bytes_array.pack('C*').unpack('L')
        hex_string = decimal_array[0].to_s(16).upcase
        hex_string
    rescue StandardError => e
```

```
        warn "Error converting UID to hex: #{e.message}"
        nil
    end
end
```

Després, a la part de client necessitem gestionar un event de tipus 'key-press-event' que detecta quan una tecla es premuda. La forma que té el teclat d'escriure les dades es número a número fins que, un cop ha escrit tots els números, escriu la paraula 'Return'. De tal manera, la lectura escrita pel lector sería: 'XXXXXXXXXReturn', on les 'X' son els dígits del UID. Per a poder separar els dígits, passar-los a hexadecimal i fer la autenticació, hem dissenyat la funció 'handle_keypress'.

JavaScript
```
entry = Gtk::Entry.new
entry.signal_connect("key-press-event") do |widget, event|
    handle_keypress(widget, event)
end

def handle_keypress(widget, event)
    key_val = event.keyval
    key_str = Gdk::Keyval.to_name(key_val)

    if key_str == "Return"
        authenticate_user(uid.hex_uid)
        self.uid.id.clear
    else
        self.uid.id += key_str
    end

    true
end
```

Resumidament, el que fa es agafar cada carácter escrit pel lector, afegir-los al ID d'una instància de UID i, un cop detectat el 'Return', passa el valor guardat a 'self.uid.id' a hexadecimal i fa l'autenticació amb aquest valor.

3.4. Codi CSS

Aquest full d'estils CSS defineix una aparença per a una interfície d'usuari bàsica però funcional. L'objectiu principal és proporcionar una experiència visual clara i consistent, destacant diferents elements com botons i zones de text. A continuació, es detalla cada part del disseny:

- Estil global: S'utilitza una tipografia monoespaiada (monospace) per a tots els elements, assegurant una estètica uniforme. És a dir, tot el text està alineat i uniforme.

- Botons: Tots els botons tenen cantonades arrodonides (border-radius: 12px) i un espai intern entre el contingut d'un element (com text o imatges) i la seva vora generós, “padding”, (10px 20px), amb text en blanc per garantir una bona visibilitat.

  Cada tipus de botó està codificat per colors de fons:

  - Verd (#28a745) per a autenticació.
  - Vermell (#dc3545) per a tancament de sessió.
  - Groc (#ffc107) per a consultes.

  Un efecte “hover”, un canvi interactiu que ocorre quan l'usuari passa el cursor del ratolí sobre un botó en aquest cas. Concretament reduïm lleugerament l’opacitat indicant la interacció.

- Zones de de text (TextViews): Les zones de text tenen un contorn suau (1px solid #ccc), un fons clar (#f9f9f9) i un padding mínim per a la llegibilitat. Hi ha estils especials per a:

  - Encapçalaments: Fons fosc (#2a2a2a) amb text blanc en negreta (font-weight: bold)

  - Files alternes: Diferenciació visual de les files parells (#d3ecf8, blau cel) i senars (#4682b4, blau fosc) amb text blanc.

Considerem que aquest estil s'adapta a interfícies que requereixen claredat i organització, com la nostra a l’hora de mostrar taules de dades ogestionar autenticació i consultes. Donem especial atenció a la usabilitat y claretat, utilitzant contrastos efectius i feedback visual.

Notacions

- **border-radius: 12px:** Arrodoneix les cantonades d'un element. En aquest cas, el valor 12px significa que les cantonades serán 12 píxels d'alt o ample

- **padding: 10px 20px:** Afegeix espai intern (padding) dins de l'element.10px amunt i avall.20px a l'esquerra i dreta

- **button:hover { opacity: 0.8; }:** Canvia l'opacitat del botó quan el cursor passa per sobre (hover). Opacity: 0.8 fa que el botó sigui lleugerament més transparent

- **background-color: #28a745:** Estableix el color de fons de l'element amb el valor #28a745, que és un to verd

- **background-color: #dc3545:** Cambia el color de fons a #dc3545, que es un to vermell.

- **background-color: #ffc107:** Cambia el fons al color #ffc107, que es un to groc.

- **border: 1px solid #ccc:** Afegeix una vora d' 1 píxel al voltant de l' element. El solid indica que serà continu (sense línies puntejades) i #ccc és un color gris clar.

- **padding: 2px: Afegeix un petit espai interior de 2 píxels disn del element**

- **background-color: #f9f9f9:** Defineix un fons de color #f9f9f9, que es un gris molt clar, quasi blanc

- **background-color: #2a2a2a**: Cambia el fons al color #2a2a2a, un gris molt fosc.

- **font-weight: bold:** Fa que el text sigui en negreta o més gruix
- **background-color: #d3ecf8**: Defineix un fons de color #d3ecf8, que es un to blau celest clar.
- **background-color: #4682b4:** Defineix un fons de color #4682b4, que es un to blau fosc

Totes aquestes paletes de colors, estructura, fonts i canvis visuals es poden trobar a la documentación oficial de GTK3 (Gtk – 3.0)

```
Python
/* Fonts generals */
* {
  font-family: monospace;
}

/* Botons */
button {
  border-radius: 12px;
  padding: 10px 20px;
  color: white;
}

button:hover {
  opacity: 0.8;
}

button#auth_button {
  background-color: #28a745; /* Verd */
}

button#logout_button {
  background-color: #dc3545; /* Vermell */
}

button#query_button {
  background-color: #ffc107; /* Groc */
}

/* TextViews */
textview {
  border: 1px solid #ccc;
  padding: 2px;
  background-color: #f9f9f9;
}

textview#header {
  background-color: #2a2a2a; /* Fosc */
  color: white;
```

```
  font-weight: bold;
}

textview#row_even {
  background-color: #d3ecf8; /* Blau cel */
}

textview#row_odd {
  background-color: #4682b4; /* Blau fosc */
  color: white;
}
```