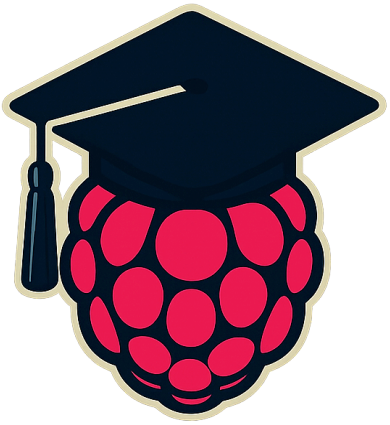


Web Berry

Critical Design Review

Alumnos: Diliara Kavieva, Bruno Enrich, Guangen Wu, Haoyan Chen y Carla Mancera

1. Preparación

Este documento técnico presenta el proceso de desarrollo de una aplicación cliente programada en Ruby y diseñada para ejecutarse en una Raspberry Pi. Su finalidad es establecer comunicación con un servidor para llevar a cabo diferentes acciones, como la lectura de tarjetas mediante un lector RFID, el envío de peticiones HTTP, la interacción con una base de datos y la visualización de información en una pantalla LCD.

Para llevar a cabo este proyecto, se han utilizado distintas librerías de Ruby, entre ellas **gtk3** para construir la interfaz gráfica, **ruby-nfc** para gestionar la lectura RFID y controladores I2C para el manejo del display LCD.

Para saber si nuestra raspberry está actualizada en su última versión hacemos

```
sudo apt update
```

Para el desarrollo del servidor del proyecto, que se encargará de recibir peticiones y responder con información desde la base de datos, se ha optado por utilizar **Node.js** como entorno de ejecución de JavaScript en el backend. Node.js permite ejecutar código JavaScript fuera del navegador, lo cual es ideal para construir aplicaciones del lado del servidor, como APIs REST

```
sudo apt install nodejs npm
```

para verificar el código :

```
node -v   # Versión de Node.js (motor de JavaScript)
npm -v    # Gestor de paquetes para JavaScript
```

2. Cliente

Para crear la aplicación cliente hemos reutilizado nuestro código del puzzle 2 modificando tanto del lcd como del lector nfc.

```javascript
require 'ruby-nfc'

class Rfid
  def initialize
    @readers = NFC::Reader.all
    if @readers.empty?
      puts "No se encontraron lectores NFC."
      exit
    end
  end

  def read_uid
    @readers[0].poll(Mifare::Classic::Tag) do |tag|
      begin
        return tag.uid_hex.upcase
      rescue StandardError => e
        puts "Error al leer la tarjeta: #{e.message}"
        return nil
      end
    end
    nil
  end
end
```

Básicamente es el puzzle 1 donde utilizamos la librería ruby-nfc para leer datos de las tarjetas.

La clase `LCDController` permite gestionar una pantalla LCD conectada por I2C a la Raspberry Pi. Utiliza la librería `i2c/drivers/lcd` para mostrar mensajes en una pantalla de 4 líneas y 20 caracteres por línea.
Incluye un método `printLCD(texto)` que limpia la pantalla y muestra el texto dividido por líneas, respetando las dimensiones del display. Esta clase se utiliza en el proyecto para informar al usuario mediante mensajes como instrucciones de inicio, errores de autenticación o mensajes de bienvenida.

```javascript
require 'i2c/drivers/lcd'

class LCDController
```

```
    def initialize
      @display = I2C::Drivers::LCD::Display.new('/dev/i2c-1', 0x27, rows: 4,
cols: 20)
    end

    def printLCD(texto)
      @display.clear
      # Divide el texto en fragmentos de 20 caracteres
      lineas = texto.scan(/.{1,20}/)
      # Imprime cada lnea hasta un mximo de 4
      lineas.each_with_index do |linea, index|
        @display.text(linea, index) if index < 4
      end
    end

    def printCenter(mensaje)
      # Divide el mensaje en lneas segn saltos de lnea
      lineas = mensaje.split("\n")

      # Centrar cada lnea
      lineas_centradas = lineas.map do |linea|
        espacio = [(20 - linea.length) / 2, 0].max
        " " * espacio + linea
      end

      # Imprimir las lneas centradas
      printLCD(lineas_centradas.join("\n"))
    end
  end
```

En este fragmento de código importamos primero las librerías que se va a utilizar como “gtk3” para la interfaz gráfica, el “thread” para el uso de hilos, “LCDController” y “puzzle1” para manejar la pantalla LCD y Rfid para leer UID. Y “json” y “net/http” que permiten hacer peticiones HTTP y parsear JSON.
Luego definimos constantes para rutas, mensajes de la pantalla y configuración de timeout.
Después creamos un cssprovider que aplica a toda la pantalla GTK que estiliza botones, labels y ventanas.

JavaScript
```
require "gtk3"
require "thread"
require_relative "LCDController"
require_relative "puzzle1"
require "json"
```

```
require "net/http"

CSS_FILE       = "disseny.css"
LOGIN_MESSAGE  = "Please,\nlogin with\nyour card"
AUTH_ERROR_MSG = "Authentication\nerror"
API_BASE       = "http://10.192.40.80:3000"
TIMEOUT_SEC    = 120

def apply_css
  provider = Gtk::CssProvider.new
  provider.load(path: CSS_FILE)
  Gtk::StyleContext.add_provider_for_screen(
    Gdk::Screen.default,
    provider,
    Gtk::StyleProvider::PRIORITY_USER
  )
end
```

La clase MainWindow es la clase que gestiona toda la **interfaz gráfica** de la aplicación usando GTK3.

```
JavaScript
class MainWindow
  def initialize(lcd)
    apply_css //aplicamos css
    @lcd = lcd //guardamos el controlador de la LCD
    @rfid_thread = nil
    @timeout_id  = nil

    @window = Gtk::Window.new("Course Manager")
    @window.set_default_size(500, 200) //tamaño
    @window.signal_connect("destroy") { cleanup_and_quit } //cierre al salir

    show_login //Limpia la ventana y muestra un mensaje tipo "login with
your card". y imprime el texto en la LCD
    Gtk.main //bucle principal
  end
```

En esta parte definimos un método “cleanup_and_quit” que detiene el hilo y sale del bucle gtk. Luego un método “clear_window” que limpia la ventana.
Definimos también la ventana login, en el que muestra el mensaje de login en la LCD. Luego llama al método “start_rfid_read” para empezar la lectura NFC, que crea un hilo auxiliar para imprimir el UID en consola. Finalmente el método “show_reader_error” que actualiza la etiqueta con el mensaje de error y lo pinta de color rojo.

JavaScript

```
  private

  def cleanup_and_quit
    @rfid_thread&.kill
    Gtk.main_quit
  end

  def clear_window
    @window.children.each { |w| @window.remove(w) }
  end

  # — Login screen —
  def show_login
    clear_window
    @lcd.printCenter(LOGIN_MESSAGE)

    @frame = Gtk::Frame.new
    @frame.set_border_width(10)
    @frame.override_background_color(:normal, blue)

    vbox = Gtk::Box.new(:vertical, 5)
    @frame.add(vbox)

    # Aquí creamos directamente el label sin build_label
    @label = Gtk::Label.new("Please, login with your university card")
    @label.override_color(:normal, white)
    @label.set_halign(:center)
    vbox.pack_start(@label, expand: true, fill: true, padding: 10)

    @window.add(@frame)
    @window.show_all

    start_rfid_read
  end

  def start_rfid_read
    @rfid_thread&.kill
    @rfid_thread = Thread.new do
      begin
        rfid = Rfid.new
        uid  = rfid.read_uid
        puts "UID leído: #{uid}"
        GLib::Idle.add { authenticate(uid); false }
      rescue => e
        GLib::Idle.add { show_reader_error(e.message); false }
      end
    end
```

```
  end

  def show_reader_error(msg)
    @label.text = "Reader error: #{msg}"
    @frame.override_background_color(:normal, red)
    @lcd.printCenter("Reader\nerror")
  end
```

Autenticación del usuario mediante UID y la posterior **creación de la pantalla de consultas** (query screen) si el login es exitoso:
Hace una petición HTTP GET al servidor web para consultar si existe el estudiante con ese UID.

JavaScript

```
  # - Authentication -
  def authenticate(uid)
    response =
Net::HTTP.get_response(URI("#{API_BASE}/students?student_id=#{uid}"))
       //verifica s la UID leido es del estudiante
    if response.is_a?(Net::HTTPSuccess) //si es correcta
      data = JSON.parse(response.body) rescue {} //Convertimos en Json
      students = data["students"]//extraemos el array
      if students.is_a?(Array) && !students.empty? //si existe en la lista
        @student_name = students.first["name"] // se guarda el nombre
        show_query_screen //se lanza la pantalla de consultas
      else
        show_auth_error //show error si falla
      end
    else
      show_auth_error
    end
  rescue
    show_auth_error
  end
```

Definimos la variable show_auth_error para que imprima error y que cambie el color del botón . Y empezamos a explicar el Query.

JavaScript

```
  def show_auth_error
    @label.text = "Authentication error, please try again."
    @frame.override_background_color(:normal, red)
    @lcd.printCenter(AUTH_ERROR_MSG) //mensaje en la LCD
  end

  # – Query screen –
  def show_query_screen
    clear_timeout //limpamos
    clear_window
    @frame&.destroy
    @lcd.printCenter("Welcome\n#{@student_name}") //mostramos en la LCD

    vbox = Gtk::Box.new(:vertical, 5)
    @window.add(vbox)

    welcome_label = Gtk::Label.new("Welcome #{@student_name}") //nueva
ventana del cliente y sus configuraciones
    welcome_label.override_color(:normal, white)
    welcome_label.set_halign(:start)
    vbox.pack_start(welcome_label, expand: false, fill: true, padding: 10)

    # Entry + Go //Entrada de texto (Gtk::Entry) para introducir consultas
    hbox = Gtk::Box.new(:horizontal, 5)
    @entry  = Gtk::Entry.new.tap { |e| e.set_placeholder_text("timetables,
tasks, marks") }
//Botón Go, que al hacer clic ejecuta el método perform_query
    @go_btn = Gtk::Button.new(label: "Go")
    @go_btn.signal_connect("clicked") { perform_query }
    hbox.pack_start(@entry, expand: true, fill: true, padding: 0)
    hbox.pack_start(@go_btn, expand: false, fill: false, padding: 0)
    vbox.pack_start(hbox, expand: false, fill: true, padding: 5)
```

Esta parte es para mostrar los datos y definir un botón de logout que vuelve a la ventana de login. En “perform_query” gestiona el flujo del hilo que actualiza gtk. Y el “popular_tree” construye dinámicamente las filas y columnas de la tabla.

JavaScript

```
    # Results TreeView
    @store = Gtk::ListStore.new
    @tree  = Gtk::TreeView.new(@store)
    scrolled = Gtk::ScrolledWindow.new
    scrolled.set_policy(:automatic, :automatic)
    scrolled.add(@tree)
    vbox.pack_start(scrolled, expand: true, fill: true, padding: 5)

    # Logout
    btn_logout = Gtk::Button.new(label: "Logout")
    btn_logout.signal_connect("clicked") do
      clear_timeout
      show_login
    end
    vbox.pack_start(btn_logout, expand: false, fill: false, padding: 10)

    @window.show_all
    start_timeout
  end

  def perform_query
    reset_timeout
    query = @entry.text.strip
    Thread.new do
      begin
        data = fetch_data(query)
        GLib::Idle.add { populate_tree(data); false }
      rescue => e
        GLib::Idle.add { show_error("Error: #{e.message}"); false }
      end
    end
  end
```

La **búsqueda de datos en el servidor** y la **visualización de esos datos en una tabla gráfica (`Gtk::TreeView`)**. También incluye una función para mostrar errores en la interfaz gráfica si algo falla.

JavaScript

```
  def fetch_data(path) //Hacer una petición HTTP al servidor, según el texto
que el usuario ha escrito (por ejemplo: timetables, tasks, marks).
    uri = URI("#{API_BASE}/#{path}")
    res = Net::HTTP.get_response(uri) //get al servidor y lo guarda
    raise "HTTP #{res.code}" unless res.is_a?(Net::HTTPSuccess) //salta
error
    JSON.parse(res.body) //pasa de Json a Array
  end
```

Este método **recibe un array de hashes** con los datos devueltos por el servidor, y **los muestra en una tabla gráfica interactiva (`Gtk::TreeView`)**.

Si es la primera vez, crea automáticamente las columnas según las claves de los datos. Después, llena cada fila con los valores correspondientes.

Si la tabla no tiene columnas, el programa **crea automáticamente una por cada clave del primer elemento** del array recibido. Luego, **rellena la tabla con los datos**, asignando cada valor a su columna correspondiente.

Las columnas se nombran con las claves, usando mayúscula inicial por estética.

JavaScript

```
 def populate_tree(arr)
    @store.clear //limpia las tablas anteriores
    return if arr.empty?

    if @tree.columns.empty?
      keys = arr.first.keys
      @store = Gtk::ListStore.new(*Array.new(keys.size, String))
      @tree.model = @store
      keys.each_with_index do |k, i|
        col = Gtk::TreeViewColumn.new(k.capitalize,
Gtk::CellRendererText.new, text: i)
        @tree.append_column(col)
      end
    end

    arr.each do |row|
      iter = @store.append
```

```
      row.values.each_with_index { |v, i| iter[i] = v.to_s }
    end
  end

  def show_error(msg) // muestra un msg en la interfaz si hay algun error
    dlg = Gtk::MessageDialog.new(
      parent: @window,
      flags:   :modal,
      type:    :error,
      buttons: :close,
      message: msg
    )
    dlg.run
    dlg.destroy
  end
```

Este último bloque de código tiene **dos partes importantes**: el manejo del **temporizador de inactividad (timeout)** y la **definición de colores personalizados** para usar en la interfaz gráfica. También está el arranque final de la aplicación.

JavaScript

```
 def start_timeout //vuelve automaticamente a la pantalla de login
    @timeout_id = GLib::Timeout.add_seconds(TIMEOUT_SEC) do
      show_login
      false
    end
  end

  def reset_timeout //lo cancela y lo vuelve a iniciar
    clear_timeout
    start_timeout
  end

  def clear_timeout //cancela el temporizador si ya esta activo
    GLib::Source.remove(@timeout_id) if @timeout_id
  end
//colores para el GTK
 def blue()  = Gdk::RGBA.new(0, 0, 1, 1)
  def red()   = Gdk::RGBA.new(1, 0, 0, 1)
  def white() = Gdk::RGBA.new(1, 1, 1, 1)
end

lcd = LCDController.new //crea el controlador para la LCD
MainWindow.new(lcd) //inicia la ventana principal
```

Flujo completo:

1. Arranque → CSS → Login UI.
2. Login exitoso → destruye login → ventana principal.
3. Usuario escribe consulta → HTTP GET → parseo JSON → muestra tabla.
4. Inactividad o “Logout” → regresa a login NFC.

Código completo.

JavaScript

```
require "gtk3"
require "thread"
require_relative "LCDController"
require_relative "puzzle1"
require "json"
require "net/http"

CSS_FILE       = "disseny.css"
LOGIN_MESSAGE  = "Please,\nlogin with\nyour card"
AUTH_ERROR_MSG = "Authentication\nerror"
API_BASE       = "http://10.192.40.80:3000"
TIMEOUT_SEC    = 120

def apply_css
  provider = Gtk::CssProvider.new
  provider.load(path: CSS_FILE)
  Gtk::StyleContext.add_provider_for_screen(
    Gdk::Screen.default,
    provider,
    Gtk::StyleProvider::PRIORITY_USER
  )
end

class MainWindow
  def initialize(lcd)
    apply_css
    @lcd = lcd
    @rfid_thread = nil
    @timeout_id  = nil

    @window = Gtk::Window.new("Course Manager")
    @window.set_default_size(500, 200)
    @window.signal_connect("destroy") { cleanup_and_quit }

    show_login
    Gtk.main
  end
```

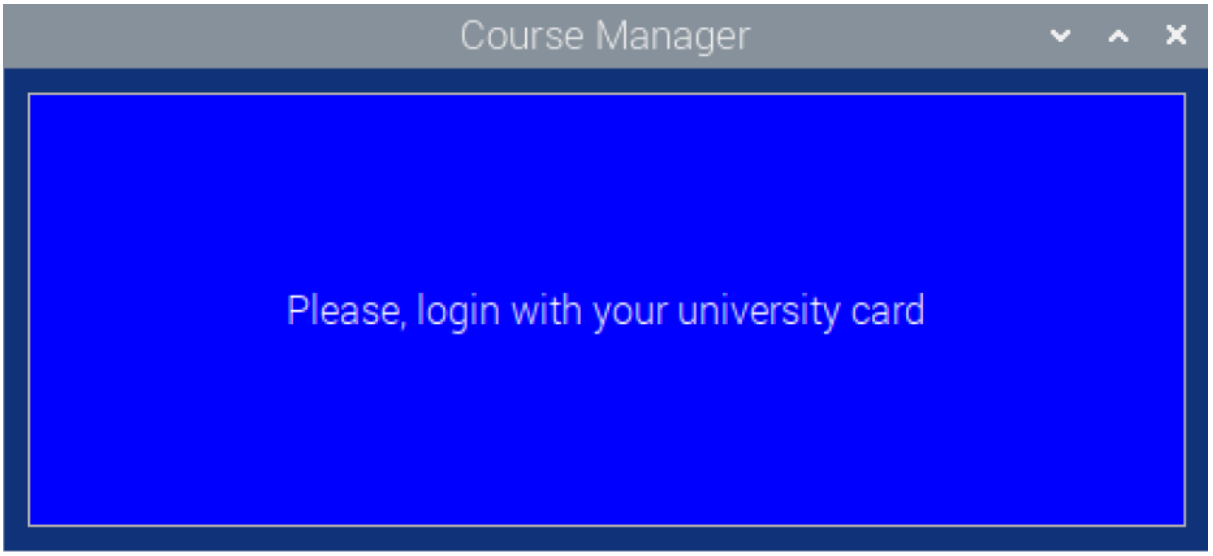
```ruby
private

def cleanup_and_quit
  @rfid_thread&.kill
  Gtk.main_quit
end

def clear_window
  @window.children.each { |w| @window.remove(w) }
end

# – Login screen –
def show_login
  clear_window
  @lcd.printCenter(LOGIN_MESSAGE)

  @frame = Gtk::Frame.new
  @frame.set_border_width(10)
  @frame.override_background_color(:normal, blue)

  vbox = Gtk::Box.new(:vertical, 5)
  @frame.add(vbox)

  # Aquí creamos directamente el label sin build_label
  @label = Gtk::Label.new("Please, login with your university card")
  @label.override_color(:normal, white)
  @label.set_halign(:center)
  vbox.pack_start(@label, expand: true, fill: true, padding: 10)

  @window.add(@frame)
  @window.show_all

  start_rfid_read
end

def start_rfid_read
  @rfid_thread&.kill
  @rfid_thread = Thread.new do
    begin
      rfid = Rfid.new
      uid  = rfid.read_uid
      puts "UID leído: #{uid}"
      GLib::Idle.add { authenticate(uid); false }
    rescue => e
      GLib::Idle.add { show_reader_error(e.message); false }
    end
  end
```

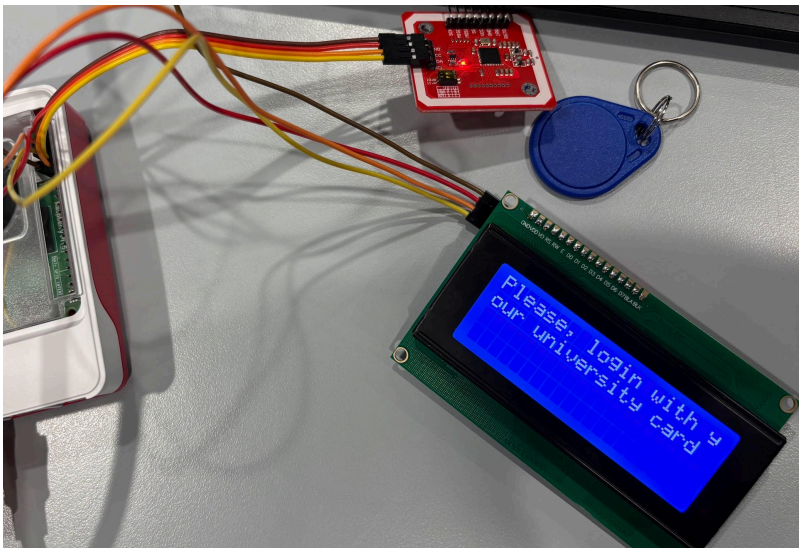
```
  end

  def show_reader_error(msg)
    @label.text = "Reader error: #{msg}"
    @frame.override_background_color(:normal, red)
    @lcd.printCenter("Reader\nerror")
  end

  # – Authentication –
  def authenticate(uid)
    response =
Net::HTTP.get_response(URI("#{API_BASE}/students?student_id=#{uid}"))
    if response.is_a?(Net::HTTPSuccess)
      data = JSON.parse(response.body) rescue {}
      students = data["students"]
      if students.is_a?(Array) && !students.empty?
        @student_name = students.first["name"]
        show_query_screen
      else
        show_auth_error
      end
    else
      show_auth_error
    end
  rescue
    show_auth_error
  end

  def show_auth_error
    @label.text = "Authentication error, please try again."
    @frame.override_background_color(:normal, red)
    @lcd.printCenter(AUTH_ERROR_MSG)
  end

  # – Query screen –
  def show_query_screen
    clear_timeout
    clear_window
    @frame&.destroy
    @lcd.printCenter("Welcome\n#{@student_name}")

    vbox = Gtk::Box.new(:vertical, 5)
    @window.add(vbox)

    welcome_label = Gtk::Label.new("Welcome #{@student_name}")
    welcome_label.override_color(:normal, white)
    welcome_label.set_halign(:start)
    vbox.pack_start(welcome_label, expand: false, fill: true, padding: 10)
```

```
    # Entry + Go
    hbox = Gtk::Box.new(:horizontal, 5)
    @entry  = Gtk::Entry.new.tap { |e| e.set_placeholder_text("timetables,
tasks, marks") }
    @go_btn = Gtk::Button.new(label: "Go")
    @go_btn.signal_connect("clicked") { perform_query }
    hbox.pack_start(@entry, expand: true, fill: true, padding: 0)
    hbox.pack_start(@go_btn, expand: false, fill: false, padding: 0)
    vbox.pack_start(hbox, expand: false, fill: true, padding: 5)

    # Results TreeView
    @store = Gtk::ListStore.new
    @tree  = Gtk::TreeView.new(@store)
    scrolled = Gtk::ScrolledWindow.new
    scrolled.set_policy(:automatic, :automatic)
    scrolled.add(@tree)
    vbox.pack_start(scrolled, expand: true, fill: true, padding: 5)

    # Logout
    btn_logout = Gtk::Button.new(label: "Logout")
    btn_logout.signal_connect("clicked") do
      clear_timeout
      show_login
    end
    vbox.pack_start(btn_logout, expand: false, fill: false, padding: 10)

    @window.show_all
    start_timeout
  end

  def perform_query
    reset_timeout
    query = @entry.text.strip
    Thread.new do
      begin
        data = fetch_data(query)
        GLib::Idle.add { populate_tree(data); false }
      rescue => e
        GLib::Idle.add { show_error("Error: #{e.message}"); false }
      end
    end
  end

  def fetch_data(path)
    uri = URI("#{API_BASE}/#{path}")
    res = Net::HTTP.get_response(uri)
    raise "HTTP #{res.code}" unless res.is_a?(Net::HTTPSuccess)
```

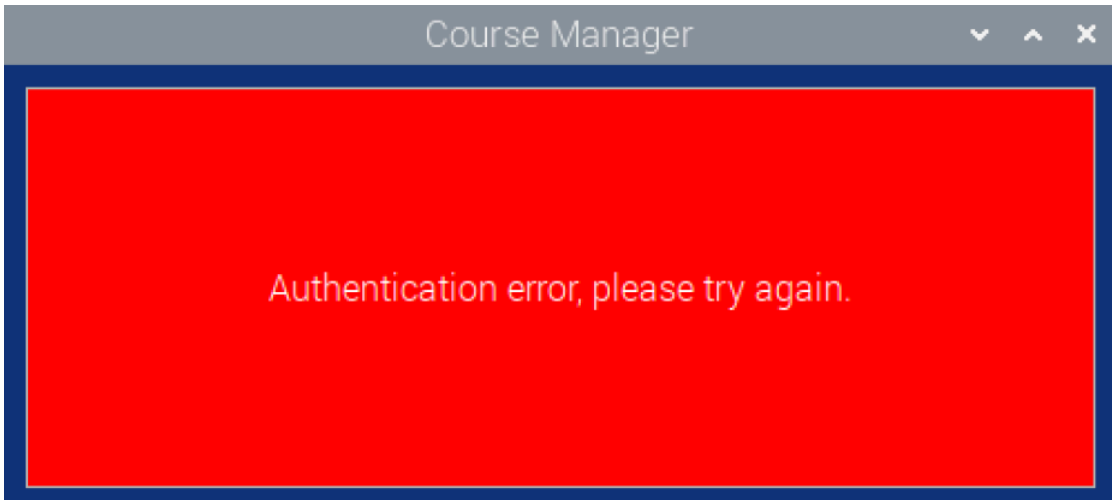
```
    JSON.parse(res.body)
  end

  def populate_tree(arr)
    @store.clear
    return if arr.empty?

    if @tree.columns.empty?
      keys = arr.first.keys
      @store = Gtk::ListStore.new(*Array.new(keys.size, String))
      @tree.model = @store
      keys.each_with_index do |k, i|
        col = Gtk::TreeViewColumn.new(k.capitalize,
Gtk::CellRendererText.new, text: i)
        @tree.append_column(col)
      end
    end

    arr.each do |row|
      iter = @store.append
      row.values.each_with_index { |v, i| iter[i] = v.to_s }
    end
  end

  def show_error(msg)
    dlg = Gtk::MessageDialog.new(
      parent: @window,
      flags:   :modal,
      type:    :error,
      buttons: :close,
      message: msg
    )
    dlg.run
    dlg.destroy
  end

  # — Timeout management —
  def start_timeout
    @timeout_id = GLib::Timeout.add_seconds(TIMEOUT_SEC) do
      show_login
      false
    end
  end

  def reset_timeout
    clear_timeout
    start_timeout
  end
```

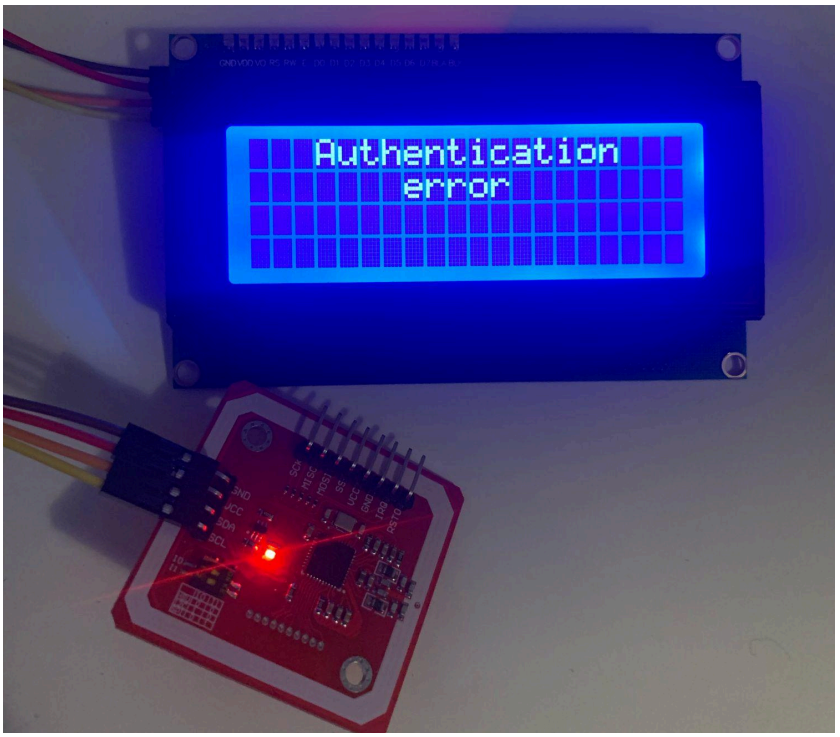
```
  def clear_timeout
    GLib::Source.remove(@timeout_id) if @timeout_id
  end

  # — Color helpers —
  def blue()  = Gdk::RGBA.new(0, 0, 1, 1)
  def red()   = Gdk::RGBA.new(1, 0, 0, 1)
  def white() = Gdk::RGBA.new(1, 1, 1, 1)
end

# Arranque
lcd = LCDController.new
MainWindow.new(lcd)
```





```
pi@guang:~/PBE/Cliente $ ruby test5.rb
UID leído: 5A5BC301
```





3. Servidor

Decidimos separar el servidor en dos partes principales, las programadas en JavaScript:

- PBE_server.js
- rutas.js
- controllers.js
- modelos: Student.js, Mark.js, Timetable.js, Task.js

Y el diseño de la web HTML con su propio lenguaje:

- design_web.html

El objetivo principal de esta parte del proyecto es traducir las query's en objetos de JavaScript, es decir, con nuestro cliente Ruby enviamos nuestras peticiones o consultas, query's, al servidor. Este recibe las peticiones de parte del cliente y se las envía a la base de datos MongoDB con tal de recibir aquello que el cliente está pidiendo, todo esto según los objetos de JavaScript que han sido traducidos.

1. PBE_server.js

Descripción: Este es el documento encargado de arrancar nuestro servidor y de conectar con la base de datos.

- Configura CORS y JSON parsing.
- Conexión a MongoDB
- Router centralizado (`routes/rutas.js`).

JavaScript

```
const express = require('express');
const cors = require('cors');
const mongoose = require('mongoose');

// Conexión a MongoDB
mongoose.connect('mongodb://localhost:27017/pbe', { useNewUrlParser: true,
useUnifiedTopology: true })
  .then(() => console.log('Conectado a MongoDB'))
  .catch(err => console.log('Error al conectar con MongoDB', err));

const app = express();
const PORT = 3000;

// Middleware
app.use(cors()); // Habilitar CORS para aceptar peticiones desde otros
orígenes
app.use(express.json()); // Para que el backend entienda las peticiones con
JSON

// Rutas
const apiRoutes = require('./routes/rutas');
app.use('/', apiRoutes);

// Arrancar servidor
app.listen(PORT, () => {
  console.log(`Servidor escuchando en http://localhost:${PORT}`);
```

```
});
```

---

2. routes/rutas.js

Descripción: Declaración de rutas públicas y protegidas con middleware. Según la autentificación que se encarga de hacer nuestro servidor, primero el cliente se identifica y a partir de aquí puede acceder a los dos tipos de rutas, privadas y públicas, descritas en este documento.

- Separación clara: rutas públicas antes de `authMiddleware`, rutas protegidas después.
- Exposición de endpoints

```javascript
const express = require('express');
const router = express.Router();
const controller = require('../controllers/controllers');

// Rutas públicas
router.get('/timetables', controller.getTimetables);
router.get('/tasks', controller.getTasks);
router.get('/user/:uid', controller.getUserByUid); // Obtener nombre públicamente

// Autenticación para las siguientes rutas
router.use(controller.authMiddleware);

// Rutas protegidas
router.get('/marks', controller.getMarks);
router.get('/me', controller.getMe); // Obtener UID y nombre del usuario autenticado

module.exports = router;
```

---

3. controllers/controllers.js

Descripción: Forma el algoritmo de organización, a partir del UID, controla el

horario, las tareas y las notas del alumno, es decir, aquí encontramos cómo funcionará el servidor y sus funciones propias.

- Controladores asíncronos con Mongoose, con la librería MongoDB.
- parseQuery genérico para transformar filtros de URL.

JavaScript

```
const Timetable = require('../models/Timetable');
const Task = require('../models/Task');
const Mark = require('../models/Mark');
const Student = require('../models/Student');

// Middleware de autenticación
exports.authMiddleware = async (req, res, next) => {
  const uid = req.headers['uid'];
  if (!uid) return res.status(401).json({ error: 'Falta UID' });

  const student = await Student.findOne({ uid });
  if (!student) return res.status(403).json({ error: 'UID no registrado' });

  req.student = student;
  next();
};

// (público)
exports.getTimetables = async (req, res) => {
  const filter = parseQuery(req.query);
  const limit = parseInt(req.query.limit) || null;
  const data = await Timetable.find(filter)
    .sort({ day: 1, hour: 1 })
    .limit(limit);
  res.json(data);
};

// (público)
exports.getTasks = async (req, res) => {
  const filter = parseQuery(req.query);
  const tasks = await Task.find(filter).sort({ date: 1 });
  res.json(tasks);
};
//  (protegido)
exports.getMarks = async (req, res) => {
  const filter = { student_uid: req.student.uid, ...parseQuery(req.query) };
  const marks = await Mark.find(filter).sort({ subject: 1 });
  res.json(marks);
```

```
};

// GET /me (usuario autenticado)
exports.getMe = (req, res) => {
  res.json({ uid: req.student.uid, name: req.student.name });
};

// GET /user/:uid (público)
exports.getUserByUid = async (req, res) => {
  const { uid } = req.params;
  const student = await Student.findOne({ uid });
  if (!student) return res.status(404).json({ error: 'Usuario no encontrado'
});
  res.json({ name: student.name });
};

// para convertir parámetros de consulta a formato MongoDB
function parseQuery(query) {
  const result = {};
  for (const key in query) {
    if (key === 'limit') continue;
    if (key.includes('[')) {
      const [field, op] = key.split(/\[|\]/);
      const value = query[key] === 'now'
        ? (field === 'date' ? new Date() : undefined)
        : query[key];
      if (!result[field]) result[field] = {};
      result[field][`$${op}`] = value;
    } else {
      result[key] = query[key];
    }
  }
  return result;
}
```

“for (const key in query)” – Se recorre cada clave del objeto query.
“const value = query[key] === 'now'
? (field === 'date' ? new Date() : undefined)
: query[key];” – si el valor es “now” y el campo es “date”, se reemplaza por “new Date” (fecha actual). En otros casos, se deja el valor tal cual

---

4. models/*.js

Student.js

```javascript
const mongoose = require('mongoose');

const studentSchema = new mongoose.Schema({
  uid: { type: String, required: true, unique: true },
  name: { type: String, required: true }
});

const Student = mongoose.model('Student', studentSchema);

module.exports = Student;
```

Timetable.js

```javascript
const mongoose = require('mongoose');

const TimetableSchema = new mongoose.Schema({
  day: String,
  hour: String,
  subject: String,
  room: String,
  teacher: String
});

module.exports = mongoose.model('Timetable', TimetableSchema);
```

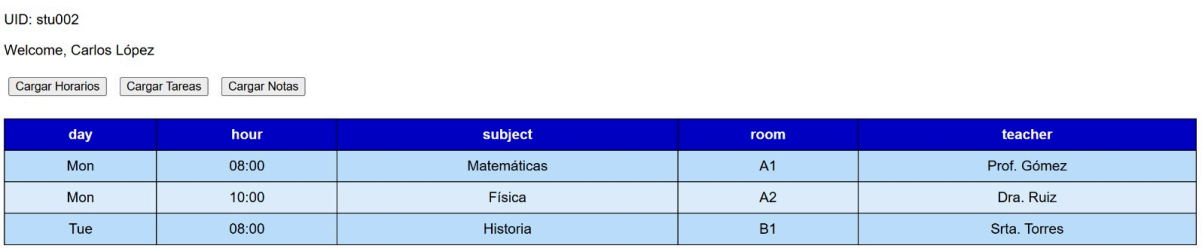
Task.js

```javascript
const mongoose = require('mongoose');

const TaskSchema = new mongoose.Schema({
  title: { type: String, required: true },
  description: String,
  date: { type: Date, required: true },
  subject: String,
  student_uid: String // Para asociar tareas a un estudiante
});
```

```
module.exports = mongoose.model('Task', TaskSchema);
```

Mark.js

```
JavaScript
const mongoose = require('mongoose');

const MarkSchema = new mongoose.Schema({
  subject: String,
  value: Number,
  student_uid: String
});

module.exports = mongoose.model('Mark', MarkSchema);
```



day	hour	subject	room	teacher
Mon	08:00	Matemáticas	A1	Prof. Gómez
Mon	10:00	Física	A2	Dra. Ruiz
Tue	08:00	Historia	B1	Srta. Torres

Esto sería un ejemplo del servidor con unas base de datos de ejemplo como quedaría el resutaldo.

---

design_web.html + JS Frontend

Descripción: Interfaz HTML+JS para interfaz y mostrar tablas.

- Eliminación de campos técnicos (`_id`, `__v`).
- Diseño CSS.

Problemas encontrados

```
function fetchUserName(uid) {
  fetch(`http://localhost:3000/user/${uid}`,
```

Al principio hemos usado http://localhost:3000/me, intentando devolver el

nombre, pero nos salía error. Al conectarse directamente a uid, nos empezó a devolver el nombre.

```
<!DOCTYPE html>
<html lang="es">

<head>
  <meta charset="UTF-8" />
  <title>Prueba del Backend Escolar</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 20px;
    }

    input,
    button {
      margin: 5px;
    }

    .hidden {
      display: none;
    }

    table {
      border-collapse: collapse;
      width: 100%;
      margin-top: 20px;
    }

    th,
    td {
      border: 1px solid #000;
      padding: 8px;
      text-align: center;
    }

    th {
      background-color: #0000cc;
      color: white;
      text-transform: lowercase;
    }

    tr:nth-child(even) {
      background-color: #d9f0ff;
    }
```

```
      tr:nth-child(odd) {
        background-color: #b3e0ff;
      }

      #contenedor {
        margin-top: 20px;
      }
    </style>
  </head>

  <body>
    <div id="uid-section">
      <label for="uid-input">Ingrese su UID:</label>
      <input type="text" id="uid-input" placeholder="Escribe tu UID" />
      <button id="btnSetUid">Aceptar UID</button>
    </div>

    <div id="main-section" class="hidden">
      <p>UID: <span id="uid-display"></span></p>
      <p>Welcome, <span id="name-display"></span></p>
      <button id="btnTimetables">Cargar Horarios</button>
      <button id="btnTasks">Cargar Tareas</button>
      <button id="btnMarks">Cargar Notas</button>
      <div id="contenedor"></div>
    </div>


    <script>
      const uidSection = document.getElementById('uid-section');
      const mainSection = document.getElementById('main-section');
      const uidInput = document.getElementById('uid-input');
      const uidDisplay = document.getElementById('uid-display');
      const nameDisplay = document.getElementById('name-display');
      const contenedor = document.getElementById('contenedor');
      let currentUid = '';

      function showMain() {
        uidSection.classList.add('hidden');
        mainSection.classList.remove('hidden');
        uidDisplay.textContent = currentUid;
      }
```

```
    function fetchUserName(uid) {
      fetch(`http://localhost:3000/user/${uid}`, {
        headers: {
          'uid': uid
        }
      })
        .then(res => res.json())
        .then(data => {
          if (data && data.name) {
            nameDisplay.textContent = data.name;
          } else {
            nameDisplay.textContent = 'usuario no encontrado';
          }
        })
        .catch(err => {
          console.error(err);
          nameDisplay.textContent = 'Error al obtener el nombre';
        });
    }


  document.getElementById('btnSetUid').addEventListener('click', () => {
    const uid = uidInput.value.trim();
    if (!uid) {
      alert('Por favor, escribe un UID.');
      return;
    }
    currentUid = uid;
    showMain();
    fetchUserName(uid);
  });

  function limpiar() {
    contenedor.innerHTML = '';
  }



function renderTabla(data) {
  if (!Array.isArray(data) || data.length === 0) {
    contenedor.innerHTML = '<p>No hay datos disponibles.</p>';
    return;
  }

  const tabla = document.createElement('table');
  const thead = document.createElement('thead');
  const tbody = document.createElement('tbody');

  // Encabezados
  const headers = Object.keys(data[0]).filter(key => key !== '_id' && key !== '__v');
  const filaCabecera = document.createElement('tr');
  headers.forEach(header => {
    const th = document.createElement('th');
    th.textContent = header;
    filaCabecera.appendChild(th);
  });
  thead.appendChild(filaCabecera);
```

```
        // Filas
        data.forEach(item => {
          const fila = document.createElement('tr');
          headers.forEach(header => {
            const td = document.createElement('td');
            let val = item[header];
            if (header === 'date' && val) {
              val = new Date(val).toISOString().split('T')[0]; // Solo fecha
            }
            td.textContent = val;
            fila.appendChild(td);
          });
          tbody.appendChild(fila);
        });

        tabla.appendChild(thead);
        tabla.appendChild(tbody);
        contenedor.appendChild(tabla);
      }


      function fetchData(endpoint) {
        limpiar();
        fetch(`http://localhost:3000/${endpoint}`, {
          headers: { 'uid': currentUid }
        })
          .then(res => res.json())
          .then(data => {
            renderTabla(data);
          })
          .catch(err => {
            console.error(err);
            contenedor.innerHTML = '<p>Error al conectar con el servidor.</p>';
          });
      }

      document.getElementById('btnTimetables')
        .addEventListener('click', () => fetchData('timetables'));
      document.getElementById('btnTasks')
        .addEventListener('click', () => fetchData('tasks'));
      document.getElementById('btnMarks')
        .addEventListener('click', () => fetchData('marks'));
    </script>
  </body>

  </html>
```

Bases de Datos

En este proyecto hemos utilizado **MongoDB** para almacenar toda la información de estudiantes, horarios, tareas y notas. La base de datos se llama **pbe** y se conecta desde el servidor Node.js a través de la librería `mongoose`.

Diseño e Implementación

Se han definido cuatro modelos principales en la carpeta `/models`:

- **Student.js:** guarda `uid` (identificador NFC) y `name` (nombre del estudiante).

- **Task.js:** contiene `title`, `description`, `date`, `subject` y `student_uid`.
- **Timetable.js:** almacena horarios de clases (`day`, `hour`, `subject`, `room`, `teacher`).
- **Mark.js:** registra notas (`subject`, `value`, `student_uid`).

Cada uno de estos modelos se estructura en un esquema Mongoose para permitir operaciones CRUD directas contra MongoDB.

Desde el servidor (`PBE_server.js`) se realiza la conexión a la base de datos:

```javascript
mongoose.connect('mongodb://localhost:27017/pbe', { useNewUrlParser: true, useUnifiedTopology: true })
```

Seed de Datos

Para preparar datos de prueba, se desarrolló un script `seed.js` que:

- Elimina documentos previos de las colecciones `students`, `tasks`, `marks` y `timetables`.
- Inserta estudiantes de prueba con `uid` conocidos (`abc123`, `def456`).
- Asocia tareas, horarios y notas correspondientes a estos estudiantes.

Ejemplo de inserción en el `seed.js`:

```javascript
await Student.insertMany([
  { uid: 'abc123', name: 'Juan Pérez' },
  { uid: 'def456', name: 'María García' }
]);
```

Con este procedimiento, cada vez que se ejecuta:

```
Unset
node seed.js
```

se asegura que la base de datos esté limpia y configurada para pruebas inmediatas.

Funcionalidad soportada

La base de datos proporciona soporte para:

- **Autenticación NFC:** validación de estudiantes mediante `uid`.
- **Consultas:** recuperación de tareas, horarios y notas filtradas por campos (`subject`, `date`, `mark`, etc.).
- **Respuestas JSON:** los datos recuperados son enviados al cliente para ser mostrados en la interfaz gráfica.

Toda la interacción con MongoDB se realiza de forma asíncrona usando `async/await`.