

Web Berry

Final Report

Document: [file name.doc]	Web Berry	
Date: dd/mm/yyyy		
Rev: 01		
Page 2 of 50		

REVISION HISTORY AND APPROVAL RECORD

Revision	Date	Purpose
1	08/05/2025	Creación del FR y reparto de trabajo
2	15/05/2025	Continuación del FR hacer gran parte del trabajo
3	19/05/2025	Última Revisión antes de la entrega

DOCUMENT DISTRIBUTION LIST

Name	E-mail
Haoyan Chen	haoyan.chen@estudiantat.upc.edu
Guangen Wu	guangen.wu@estudiantat.upc.edu
Diliara Kavieva	diliara.kavieva@estudiantat.upc.edu
Bruno Enrich	bruno.enrich@estudiantat.upc.edu
Carla Mancera Iñiguez	carla.mancera@estudiantat.upc.edu
Professor	
Francesc Oller Teijon	francesc.oller@upc.edu

WRITTEN BY: Haoyan Chen Guangen Wu Diliara Kavieva Carla Mancera Date 08/05/2025		REVIEWED AND APPROVED BY:	
Name	Haoyan Chen	Name	Zzzzzzz Wwwwwww
Position	WP Manager	Position	e.g. Project leader

0. CONTENTS

0.	Contents	3
1.	Document scope	4
2.	Project summary	5
3.	Time plan updated	6
4.	System design documentation	7
5.	System implementation documentation	8
6.	System characterization	9
7.	Costs	10
8.	Conclusions	11
9.	Reflection document	12

Document: [file name.doc]	Web Berry	
Date: dd/mm/yyyy		
Rev: 01		
Page 4 of 50		

1. DOCUMENT SCOPE

The scope of this document includes the design, implementation, and integration of the hardware setup (card reader and Raspberry Pi), the backend logic (UID handling, database queries), and the frontend web application that presents the data to the user. All the information this team has learned during the process of this project will be reflected in this document, everything that helped us to conclude it.

This project consists of building a web-based student assistant system that shows users their upcoming tasks, grades, and schedule based on their unique ID (UID). The system uses a card reader to detect the UID when a student taps their card, automatically loading their personalized academic information. Every step that has been taken to achieve the final result, is explained and described in this report.

This document outlines the structure and key components of the Android client for the Course Manager app. It focuses on:

- Clean Architecture layers (UI, Domain, Data, DI)
- Use of Hilt for dependency injection
- Retrofit-based network communication
- Jetpack Compose UI components
- Common development issues and fixed

It is intended for developers working on maintenance or extension of the client.

Document: [file name.doc]	Web Berry	
Date: dd/mm/yyyy		
Rev: 01		
Page 5 of 50		

2. PROJECT SUMMARY

The building process of this project is divided in three main parts: the understanding, the fundamentals and the union.

Understanding:

Once we learned how to use our materials and knew which was our objective, the project was distributed into the two main pillars that hold it. As we had no previous experience making a server or using a Raspberry Pi, through individual and team work we all got familiar with everything needed.

Fundamentals:

A **client**, using NFC technologies and a Raspberry Pi as a computer. Its purpose is to analyze a card or any identifier that may have an ID, subtract the number from it in order to show it in our LCD and provide it to the server, the other main part.

The **server**, which initially was only going to be an “html” web page. After a hard working week the team managed to produce an application for Android devices. It is an educational application that stores student information, including their marks and tasks, customizing the experience depending on the UID previously stored in a Data Base which has all the students information.

Union:

Once both the client and server were functional independently, the final step was to connect them seamlessly. The Raspberry Pi, acting as the NFC client, reads the card’s UID and sends it to the server through a request. The server then processes this UID, retrieves the corresponding student data from the database, and responds with personalized information, such as the timetable, tasks, or marks. This integration transforms the project into a complete and interactive educational system, where each component communicates reliably to deliver a smooth user experience.

Document: [file name.doc]	Web Berry	
Date: dd/mm/yyyy		
Rev: 01		
Page 6 of 50		

3. TIME PLAN UPDATED

During the development process, the project faced significant delays, particularly in the **integration between the client and the server**. Establishing a stable communication channel using HTTP requests from the Raspberry Pi to the Android-based server proved more complex than initially anticipated. Issues related to routing, authentication headers, and request handling caused a considerable deviation from the original timeline.

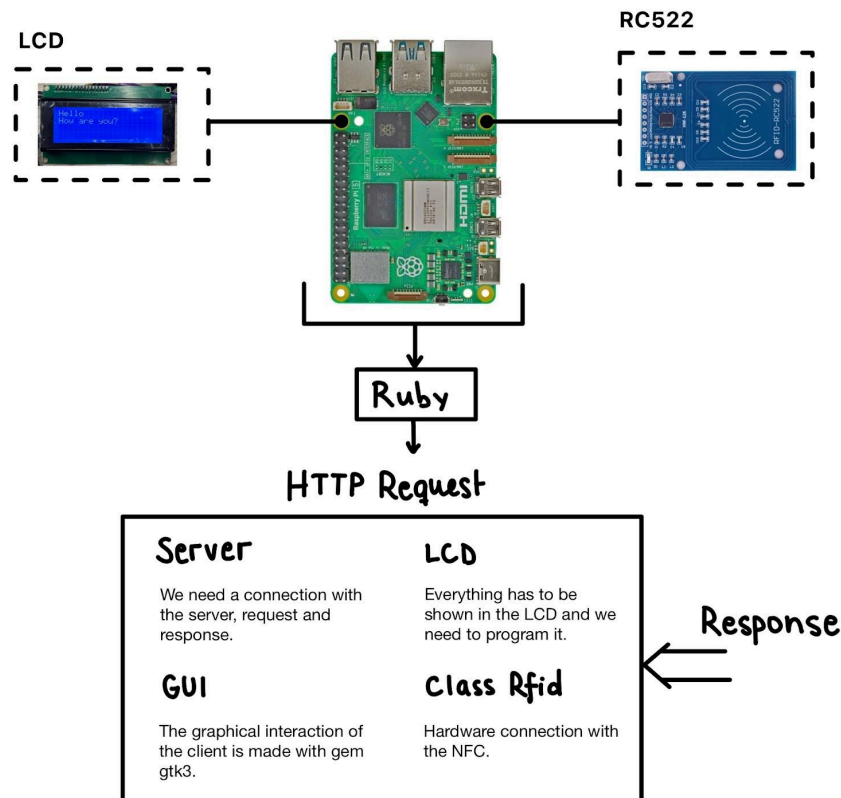
As a result, we had to **conduct numerous tests** to verify the data flow, error responses, and the handling of valid and invalid UIDs. These tests were essential to gradually identify and fix the issues. However, even with these efforts, **the final version did not achieve full stability** in all cases.

Despite this, the **core functionality of the system is operational**: the NFC client can read UIDs and communicate with the server, which returns student-specific data. While not all planned features were implemented or refined, the system serves as a functional prototype and has validated the viability of the overall concept.

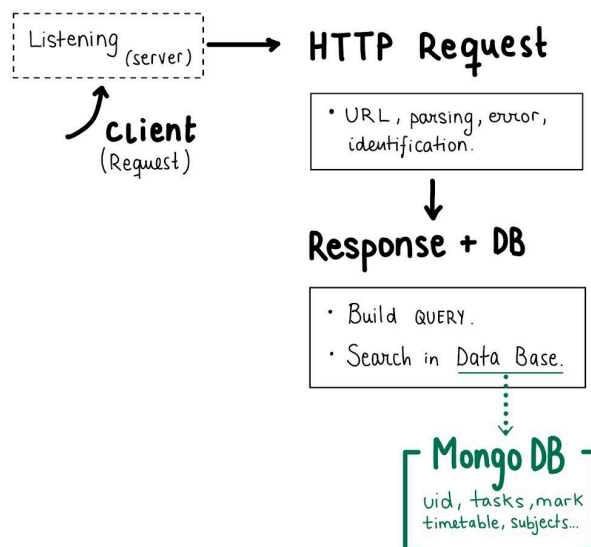


4. SYSTEM DESIGN DOCUMENTATION

Client:

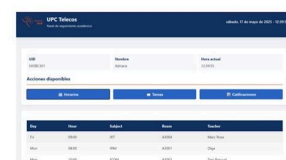


Server:



HTML

- Building of the web.



```
function updateClock() {
  const now = new Date();
  const timeStr = new toLocaleTimeString('en-GB');
  const dateStr = new toLocaleDateString('en-GB', {
    weekday: 'long',
    year: 'numeric',
    month: 'long',
    day: 'numeric'
  });
  const timeElement = document.getElementById('current-time');
  const userTimeElement = document.getElementById('current-time-user');
  if (timeElement) timeElement.textContent = `${dateStr} - ${timeStr}`;
  if (userTimeElement) userTimeElement.textContent = timeStr;
}
setInterval(updateClock, 1000);
updateClock();
```

Document: [file name.doc]	Web Berry	
Date: dd/mm/yyyy		
Rev: 01		
Page 8 of 50		

5. SYSTEM IMPLEMENTATION DOCUMENTATION

- Final schematics / software blocks
- Circuit Layout / user interface screen captures
- Final components list with values
- Circuit / device pictures

SERVIDOR:

JavaScript

```
//PBE_server.js
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({
  origin: '*',
  allowedHeaders: ['Content-Type', 'uid']
}));

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, '0.0.0.0', () => {
  console.log(`Servidor escuchando en http://0.0.0.0:${PORT}`);
});
```


Document: [file name.doc]	Web Berry	
Date: dd/mm/yyyy		
Rev: 01		
Page 9 of 50		

Lo que hemos añadido es:

```
JavaScript
app.use(cors({
  origin: '*',
  allowedHeaders: ['Content-Type', 'uid']
}));

app.use(express.json());
```

cors: Permite que cualquier cliente (desde cualquier origen *) pueda hacer peticiones al servidor, enviando encabezados como **Content-Type** y **uid**.

express.json(): Permite al servidor entender peticiones con cuerpo en formato JSON.

Además ahora nuestro servidor escuchará todas las peticiones de cualquier dispositivo .

```
JavaScript
app.listen(PORT, '0.0.0.0', () => {
  console.log(`Servidor escuchando en http://0.0.0.0:${PORT}`);
});
```

Rutas:

```
JavaScript
//rutas.js
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.getUserById); // 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;
```



Controllers

JavaScript

// [controllers.js](#)

```
const Timestep = require('../models/Timestep');
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.getTimesteps = async (req, res) => {
  const filter = parseQuery(req.query);
  const limit = parseInt(req.query.limit) || null;
  const data = await Timestep.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) => {
```

Document: [file name.doc]	Web Berry	
Date: dd/mm/yyyy		
Rev: 01		
Page 11 of 50		

```

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;
}

```

En [controllers.js](#) hemos mejorado `.authMiddleware` : `console.log` para ver si nos ha llegado uid correctamente. Además nuestro código ahora : extrae el uid de `req.headers`.

1. Busca el estudiante en la BD. Si no existe, devuelve 403 (Prohibido).
2. Adjunta el estudiante a `req.student` para usarlo en rutas posteriores.

Modelos:

```

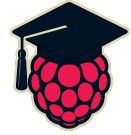
JavaScript
//Student.js
const mongoose = require('mongoose');

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;

```



JavaScript

```
//Timetable.js
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);
```

JavaScript

```
//Task.js
const mongoose = require('mongoose');

const TaskSchema = new mongoose.Schema({
  title: { type: String, required: true },
  description: String,
  subject: String
});

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

JavaScript

```
//Mark.js
const mongoose = require('mongoose');

const MarkSchema = new mongoose.Schema({
  uid: { type: String, required: true, index: true },
  subject: { type: String, required: true },
  value: { type: Number, required: true },
});

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

JavaScript

```
//seed.js
const mongoose = require('mongoose');
const Timetable = require('./models/Timetable');
const Task = require('./models/Task');
const Mark = require('./models/Mark');
```



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

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

async function seedData() {
  // Limpiar datos anteriores
  await Student.deleteMany({});
  await Timetable.deleteMany({});
  await Task.deleteMany({});
  await Mark.deleteMany({});

  // Crear estudiantes
  const students = await Student.insertMany([
    { uid: '5A5BC301', name: 'Adriana' },
    { uid: 'stu002', name: 'Carlos' },
    { uid: 'stu003', name: 'Júlia' }
  ]);

  // Crear horarios (añadiendo más asignaturas de telecos 3º)
  await Timetable.insertMany([
    { day: 'Mon', hour: '08:00', subject: 'IPAV', room: 'A3001', teacher: 'Olga' },
    { day: 'Mon', hour: '10:00', subject: 'ICOM', room: 'A3002', teacher: 'Toni Pascual' },
    { day: 'Tue', hour: '08:00', subject: 'PBE', room: 'A3201', teacher: 'Francesc Oller' },
    { day: 'Wed', hour: '12:00', subject: 'RP', room: 'A3102', teacher: 'Merce' },
    { day: 'Thu', hour: '14:00', subject: 'DSBM', room: 'A3305', teacher: 'Jordi Salazar' },
    { day: 'Fri', hour: '09:00', subject: 'IXT', room: 'A3004', teacher: 'Marc Rosa' }
  ]);

  // Crear tareas (incluyendo nuevas asignaturas)
  await Task.insertMany([
    { title: 'Ejercicios', description: 'Resolver 10 problemas', date: new Date(), subject: 'IPAV', student_uid: '5A5BC301' },
    { title: 'Lectura cap. 3', description: 'Resumen del capítulo', date: new Date(), subject: 'ICOM', student_uid: '5A5BC301' },
    { title: 'Preparar control', date: new Date(), subject: 'PBE', student_uid: 'stu002' },
    { title: 'Proyecto de red', description: 'Diseñar topología', date: new Date(), subject: 'IXT', student_uid: 'stu003' },
  ])
```

Document: [file name.doc]	Web Berry	
Date: dd/mm/yyyy		
Rev: 01		
Page 14 of 50		

```

    { title: 'Hacer exmaen de 2020 problema 3', description: 'Ejercicio de
codificación', date: new Date(), subject: 'DSBM', student_uid: 'stu002' },
    { title: 'Ejercicio de antenas', description: 'Definir tema y objetivos',
date: new Date(), subject: 'RP', student_uid: 'stu003' }
  ]);

  // Crear notas (más asignaturas y estudiantes)
  await Mark.insertMany([
    { subject: 'IPAV', value: 8.5, uid: '5A5BC301' },
    { subject: 'ICOM', value: 7.0, uid: '5A5BC301' },
    { subject: 'PBE', value: 10.0, uid: 'stu002' },
    { subject: 'IXT', value: 9.0, uid: 'stu003' },
    { subject: 'DSBM', value: 6.5, uid: 'stu002' },
    { subject: 'RP', value: 9.5, uid: 'stu003' }
  ]);

  console.log('Datos insertados correctamente');
  mongoose.disconnect();
}

```

Diseño de la página: es una página web llamada **UPC Telecom**. Permite a los estudiantes de la UPC ingresar su identificador (UID) para ver su información académica. Una vez que ingresan, pueden consultar sus **horarios**, **tareas** y **notas** usando botones. La página muestra además la hora actual y los datos del estudiante.

```

HTML
// design.html
<!DOCTYPE html>
<html lang="es">

<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Panel Escolar UPC Telecom</title>
  <style>
    :root {
      --upc-blue: #003366;
      --upc-light-blue: #0066cc;
      --upc-orange: #ff6600;
      --upc-gray: #f0f2f5;
      --upc-dark: #333333;
    }

    body {
      font-family: 'Segoe UI', 'Roboto', sans-serif;
      background-color: var(--upc-gray);
      color: var(--upc-dark);
      margin: 0;
    }
  </style>

```



```
padding: 0;
line-height: 1.6;
}

.container {
  max-width: 1200px;
  margin: 0 auto;
  padding: 20px;
}

header {
  background-color: var(--upc-blue);
  color: white;
  padding: 20px 0;
  box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);
  margin-bottom: 30px;
}

.header-content {
  display: flex;
  align-items: center;
  justify-content: space-between;
}

.logo {
  display: flex;
  align-items: center;
}

.logo img {
  height: 50px;
  margin-right: 15px;
}

.logo-text h1 {
  margin: 0;
  font-size: 24px;
  font-weight: 700;
}

.logo-text p {
  margin: 0;
  font-size: 14px;
  opacity: 0.9;
}

.user-section {
  display: flex;
  align-items: center;
}
```



```
.card {
  background: white;
  border-radius: 8px;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.05);
  padding: 25px;
  margin-bottom: 25px;
}

.card-title {
  color: var(--upc-blue);
  border-bottom: 2px solid var(--upc-orange);
  padding-bottom: 10px;
  margin-top: 0;
  margin-bottom: 20px;
  font-size: 20px;
}

input {
  padding: 12px 15px;
  border: 1px solid #ddd;
  border-radius: 4px;
  font-size: 16px;
  width: 100%;
  box-sizing: border-box;
  margin-bottom: 15px;
  transition: border 0.3s;
}

input:focus {
  border-color: var(--upc-light-blue);
  outline: none;
  box-shadow: 0 0 2px rgba(0, 102, 204, 0.2);
}

button {
  padding: 12px 20px;
  border: none;
  border-radius: 4px;
  font-size: 16px;
  font-weight: 600;
  cursor: pointer;
  transition: all 0.3s;
}

.btn-primary {
  background-color: var(--upc-orange);
  color: white;
}

.btn-primary:hover {
  background-color: #e65c00;
```




```
}

.btn-secondary {
  background-color: var(--upc-light-blue);
  color: white;
}

.btn-secondary:hover {
  background-color: #005bb5;
}

.action-buttons {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
  gap: 15px;
  margin-top: 20px;
}

.hidden {
  display: none;
}

table {
  width: 100%;
  border-collapse: collapse;
  margin-top: 20px;
  box-shadow: 0 1px 3px rgba(0, 0, 0, 0.1);
}

th {
  background-color: var(--upc-blue);
  color: white;
  padding: 12px 15px;
  text-align: left;
}

td {
  padding: 12px 15px;
  border-bottom: 1px solid #eee;
}

tr:nth-child(even) {
  background-color: #f9f9f9;
}

tr:hover {
  background-color: #f1f1f1;
}

.user-info {
  display: flex;
```



```
        justify-content: space-between;
        margin-bottom: 20px;
    }

    .user-info-item {
        flex: 1;
        padding: 10px;
        background: #f8f9fa;
        border-radius: 4px;
        margin-right: 10px;
    }

    .user-info-item:last-child {
        margin-right: 0;
    }

    .user-info-item strong {
        display: block;
        color: var(--upc-blue);
        margin-bottom: 5px;
    }

    @media (max-width: 768px) {
        .header-content {
            flex-direction: column;
            text-align: center;
        }

        .logo {
            margin-bottom: 15px;
            justify-content: center;
        }

        .action-buttons {
            grid-template-columns: 1fr;
        }

        .user-info {
            flex-direction: column;
        }

        .user-info-item {
            margin-right: 0;
            margin-bottom: 10px;
        }
    }
</style>
</head>

<body>
    <header>
```



```

<div class="container header-content">
  <div class="logo">
    <!-- Logo UPC real -->
    

    <div class="logo-text">
      <h1>UPC Telecom</h1>
      <p>Panel de seguimiento académico</p>
    </div>
  </div>
  <div class="user-section">
    <div id="current-time" style="color: white; font-weight: 600;"></div>
  </div>
</div>
</header>

<main class="container">
  <div id="uid-section" class="card">
    <h2 class="card-title">Identificación</h2>
    <p>Por favor, introduce tu identificador de estudiante UPC</p>
    <input type="text" id="uid-input" placeholder="Ejemplo: stu001" />
    <button id="btnSetUid" class="btn-primary">Acceder al panel</button>
  </div>

  <div id="main-section" class="hidden">
    <div class="card">
      <div class="user-info">
        <div class="user-info-item">
          <strong>UID</strong>
          <span id="uid-display"></span>
        </div>
        <div class="user-info-item">
          <strong>Nombre</strong>
          <span id="name-display"></span>
        </div>
        <div class="user-info-item">
          <strong>Hora actual</strong>
          <span id="current-time-user"></span>
        </div>
      </div>

      <h2 class="card-title">Acciones disponibles</h2>
      <div class="action-buttons">
        <button id="btnTimetables" class="btn-secondary"><img alt="Calendar icon" data-bbox="660 800 675 815"/> Horarios</button>
        <button id="btnTasks" class="btn-secondary"><img alt="Book icon" data-bbox="615 815 630 830"/> Tareas</button>
        <button id="btnMarks" class="btn-secondary"><img alt="Bar chart icon" data-bbox="615 830 630 845"/> Calificaciones</button>
      </div>
    </div>

    <div id="contenedor" class="card"></div>
  </div>

```



```
</div>
</main>

<script>
  // Elementos del DOM
  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 = '';

  // Actualizar reloj
  function updateClock() {
    const now = new Date();
    const timeStr = now.toLocaleTimeString('es-ES');
    const dateStr = now.toLocaleDateString('es-ES', {
      weekday: 'long',
      year: 'numeric',
      month: 'long',
      day: 'numeric'
    });

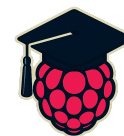
    const timeElement = document.getElementById('current-time');
    const userTimeElement = document.getElementById('current-time-user');

    if (timeElement) timeElement.textContent = `${dateStr} - ${timeStr}`;
    if (userTimeElement) userTimeElement.textContent = timeStr;
  }

  setInterval(updateClock, 1000);
  updateClock();

  // Mostrar sección principal
  function showMain() {
    uidSection.classList.add('hidden');
    mainSection.classList.remove('hidden');
    uidDisplay.textContent = currentUid;
  }

  // Obtener nombre de usuario
  function fetchUserName(uid) {
    fetch(`http://localhost:3000/user/${uid}`, {
      headers: { 'uid': uid }
    })
      .then(res => res.json())
      .then(data => {
        nameDisplay.textContent = data?.name || 'Usuario no encontrado';
      })
      .catch(err => {
```



```
        console.error(err);
        nameDisplay.textContent = 'Error al obtener datos';
    });
}

// Evento para establecer UID
document.getElementById('btnSetUid').addEventListener('click', () => {
    const uid = uidInput.value.trim();
    if (!uid) {
        alert('Por favor, introduce un identificador válido.');
```

return;

```
    }
    currentUid = uid;
    showMain();
    fetchUserName(uid);
});

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

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

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

    // Cabecera
    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.charAt(0).toUpperCase() + header.slice(1);
        filaCabecera.appendChild(th);
    });
    thead.appendChild(filaCabecera);
    tabla.appendChild(thead);

    // 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).toLocaleDateString('es-ES');
} else if (header === 'value' && !isNaN(val)) {
  td.style.fontWeight = '600';
  td.style.color = val >= 5 ? 'green' : 'red';
}

td.textContent = val || '-';
fila.appendChild(td);
});
tbody.appendChild(fila);
});

tabla.appendChild(tbody);

// Limpiar y mostrar
limpiar();
contenedor.appendChild(tabla);
}

// Obtener datos
function fetchData(endpoint) {
  limpiar();
  contenedor.innerHTML = '<p>Cargando datos...</p>';

  fetch(`http://localhost:3000/${endpoint}`, {
    headers: { 'uid': currentUid }
  })
  .then(res => {
    if (!res.ok) throw new Error(`Error ${res.status}: ${res.statusText}`);
    return res.json();
  })
  .then(data => {
    if (!data || data.length === 0) {
      contenedor.innerHTML = `<p>No hay ${endpoint} disponibles.</p>`;
    } else {
      renderTabla(data);
    }
  })
  .catch(err => {
    console.error(err);
    contenedor.innerHTML = `
    <div style="color: red; padding: 15px; background: #ffeeee;
border-radius: 4px;">
      <strong>Error al obtener datos:</strong>
      <p>${err.message}</p>
      <p>Por favor, intenta nuevamente o contacta con soporte.</p>
    </div>`
  })
}
```

Document: [file name.doc]	Web Berry	
Date: dd/mm/yyyy		
Rev: 01		
Page 23 of 50		

```

    `;
  });
}

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

</html>

```

Descripción del código de design_web.html:

→ **Su estructura:**

Usa una estructura básica con `<head>` y `<body>`.

En el `<head>` se definen:

- El título de la página.
- Los estilos CSS (colores, tamaños, márgenes, botones, tablas, etc.).

En el `<body>` está el contenido:

- Un **header** con logo, nombre y hora actual.
- Un **formulario para ingresar UID**.
- Una **sección principal oculta** que se muestra después de iniciar sesión.
- Botones para ver horarios, tareas y notas.
- Un contenedor donde se muestran las tablas de datos.

→ Además definimos colores de la UPC en variables CSS (`:root`).

→ Usamos JavaScript para mostrar y actualizar la hora actual cada segundo.

- Se actualizan dos zonas: una en la cabecera, otra en la sección de usuario.

Document: [file name.doc]	Web Berry	
Date: dd/mm/yyyy		
Rev: 01		
Page 24 of 50		

JavaScript

```
function updateClock() {
  const now = new Date();
  const timeStr = now.toLocaleTimeString('es-ES');
  const dateStr = now.toLocaleDateString('es-ES', {
    weekday: 'long',
    year: 'numeric',
    month: 'long',
    day: 'numeric'
  });

  const timeElement = document.getElementById('current-time');
  const userTimeElement = document.getElementById('current-time-user');

  if (timeElement) timeElement.textContent = `${dateStr} - ${timeStr}`;
  if (userTimeElement) userTimeElement.textContent = timeStr;
}

setInterval(updateClock, 1000);
updateClock();
```

→ Al hacer clic en "Acceder al panel":

- Guarda el UID.
- Oculta la sección de ingreso.
- Muestra la sección principal.
- Llama a la función `fetchUserName()` para buscar su nombre desde un servidor local (`localhost:3000`).

JavaScript

```
document.getElementById('btnSetUid').addEventListener('click', () => {
  const uid = uidInput.value.trim();
  if (!uid) {
    alert('Por favor, introduce un identificador válido.');
```

```
    return;
  }
  currentUid = uid;
  showMain();
  fetchUserName(uid);
});

function showMain() {
  uidSection.classList.add('hidden');
  mainSection.classList.remove('hidden');
```


Document: [file name.doc]	Web Berry	
Date: dd/mm/yyyy		
Rev: 01		
Page 25 of 50		

```
uidDisplay.textContent = currentUid;  
}
```

→ Hay 3 botones: horarios, tareas, calificaciones.

Cada botón llama a `fetchData(endpoint)` pasando el nombre del recurso.

Esa función consulta una API local y muestra los datos en tabla.

JavaScript

```
document.getElementById('btnTimetables').addEventListener('click', () =>  
  fetchData('timetables'));  
document.getElementById('btnTasks').addEventListener('click', () =>  
  fetchData('tasks'));  
document.getElementById('btnMarks').addEventListener('click', () =>  
  fetchData('marks'));
```


→ Si recibe datos:

- Crea una tabla con encabezados y filas dinámicas.
- Resalta en verde o rojo las notas según sean ≥ 5 o no.

Si no hay datos:

- Muestra un mensaje de "No hay datos disponibles."

Timatable:



UPC Telecoms
Panel de seguimiento académico

sábado, 17 de mayo de 2025 - 12:39:55

UID
5A5BC301

Nombre
Adriana

Hora actual
12:39:55

Acciones disponibles


Horarios

Tareas

Calificaciones

Day	Hour	Subject	Room	Teacher
Fri	09:00	IXT	A3004	Marc Rosa
Mon	08:00	IPAV	A3001	Olga
Mon	10:00	ICOM	A3002	Toni Pascual

Las notas:



UPC Telecoms
Panel de seguimiento académico

sábado, 17 de mayo de 2025 - 12:40:43

UID
5A5BC301

Nombre
Adriana

Hora actual
12:40:43

Acciones disponibles


Horarios

Tareas

Calificaciones

Uid	Subject	Value	Date
5A5BC301	ICOM	7	17/5/2025
5A5BC301	IPAV	8.5	17/5/2025

Tasks:


UPC Telecos
 Panel de seguimiento académico

sábado, 17 de mayo de 2025 - 12:40:17

UID
 5A5BC301

Nombre
 Adriana

Hora actual
 12:40:17

Acciones disponibles

Horarios

Tareas

Calificaciones

Title	Description	Date	Subject
Ejercicios	Resolver 10 problemas	17/5/2025	IPAV
Lectura cap. 3	Resumen del capítulo	17/5/2025	ICOM
Preparar control	-	17/5/2025	PBE
Proyecto de red	Diseñar topología	17/5/2025	IXT

CLIENT:

After doing CDR documentation we continued trying to fix the errors that we had, focusing mainly on the connection with the server and local host. One of the problems that we had is that we had problems with the local host.

After trying non-stop, we decided to restart part of the project, because we couldn't find the problem, so we made several tests.

For example, **test 6** : His principal function is to print the users name with the uid of the NFC card. We had problems printing the values of the server so, we firstly tried this method.

Test 6:

```

Python
#!/usr/bin/env ruby
require 'net/http'
require 'uri'
require 'json'
require_relative 'puzzle1' # Tu clase Rfid para NFC

# Parámetros (IP y puerto opcionales)
SERVER_IP = ARGV[0] || '172.20.10.3'
SERVER_PORT = (ARGV[1] || 3000).to_i
  
```

Document: [file name.doc]	Web Berry	
Date: dd/mm/yyyy		
Rev: 01		
Page 28 of 50		

```

begin
  # 1) Leer UID por NFC
  puts "Acerca tu tarjeta NFC..."
  uid = Rfid.new.read_uid
  puts "→ UID leída: #{uid}"

  # 2) Fetch user usando /user/:uid
  uri = URI("http://#{SERVER_IP}:#{SERVER_PORT}/user/#{uid}")
  puts "Llamando a #{uri}..."
  res = Net::HTTP.get_response(uri)

  # 3) Mostrar resultado
  if res.is_a?(Net::HTTPSuccess)
    user = JSON.parse(res.body)
    puts "Datos de usuario recibidos:"
    puts "  UID: #{uid}"
    puts "  Name: #{user['name']}"
    # Si hay otros campos en el JSON:
    user.each do |k, v|
      next if k == 'name'
      puts "  #{k.capitalize}: #{v}"
    end
  else
    warn "Error al obtener usuario: #{res.code} #{res.message}"
    warn res.body
  end

rescue => e
  warn "¡Error!: #{e.class} - #{e.message}"
end

```

This test verifies the correct execution of the code. The UID is successfully read, and the URI is correctly formed. The server responds with the corresponding user name using the `getResponse`, confirming proper communication.

```

pi@guang:~/PBE/Cliente $ ruby test6.rb
Acerca tu tarjeta NFC...
→ UID leída: 5A5BC301
Llamando a http://172.20.10.3:3000/user/5A5BC301...
Datos de usuario recibidos:
  UID: 5A5BC301
  Name: Adriana

```

Document: [file name.doc]	Web Berry	
Date: dd/mm/yyyy		
Rev: 01		
Page 29 of 50		

This confirms that the server and database are correctly processing the request. Therefore, any remaining issues are likely due to errors in the client-side code or the request path being used.

In the next part of the test, we attempt to print the timetable data associated with a specific user from the server. The goal is to ensure that the timetable is being printed correctly. Previously, several errors occurred due to type mismatches between variables (e.g., strings vs. integers), which could have caused failures. This test helps confirm that the server and database are functioning properly and that data types are handled correctly on the client side.

Test 8:

```

Python
require 'net/http'
require 'uri'
require 'json'
require_relative 'puzzle1'

SERVER_IP   = '172.20.10.3'
SERVER_PORT = 3000

def fetch_data(endpoint)
  uri      = URI("http://#{SERVER_IP}:#{SERVER_PORT}#{endpoint}")
  response = Net::HTTP.get_response(uri)
  JSON.parse(response.body) if response.is_a?(Net::HTTPSuccess)
end

def display_timetable(timetables)
  puts "\n HORARIO ACADÉMICO"
  puts "-" * 50

  # Agrupar por día
  by_day = timetables.group_by { |t| t['day'] }

  # Mostrar para cada día de lunes a viernes
  %w[Mon Tue Wed Thu Fri].each do |day|
    next unless by_day[day]

    puts "\n#{day_to_spanish(day)}:"
    puts "-" * 30

    by_day[day]
      .sort_by { |t| t['hour'] }
      .each do |entry|
        puts " #{entry['hour']} | #{entry['subject']}"
        puts " #{entry['room']} | #{entry['teacher']}"
        puts "-" * 30
      end
    end
  end
end

```



```
end

def day_to_spanish(day)
  {
    'Mon' => 'Lunes',
    'Tue' => 'Martes',
    'Wed' => 'Miércoles',
    'Thu' => 'Jueves',
    'Fri' => 'Viernes'
  }[day] || day
end

begin
  puts " Acerca tu tarjeta NFC..."
  uid = Rfid.new.read_uid
  puts " UID: #{uid}"

  # Obtener datos básicos
  user = fetch_data("/user/#{uid}")
  puts "\n Estudiante: #{user['name']}" if user

  # Obtener horario académico
  timetables = fetch_data("/timetables")
  display_timetable(timetables) if timetables

  # Obtener calificaciones
  marks = fetch_data("/user/#{uid}/marks")
  if marks&.any?
    puts "\nCalificaciones:"
    marks.each do |m|
      puts "   #{m['subject']}: #{m['value']}"
    end
  else
    puts "\nNo se encontraron calificaciones"
  end

rescue => e
  puts " Error: #{e.message}"
end
```

```
pi@guang:~/PBE/Ciente $ ruby t
Acerca tu tarjeta NFC...
UID: 5A5BC301

Estudiante: Adriana

HORARIO ACADÉMICO

Lunes:
-----
08:00 | IPAV
A3001 | Olga
-----
10:00 | ICOM
A3002 | Toni Pascual
-----

Martes:
-----
08:00 | PBE
A3201 | Francesc Oller
-----
```

```
Miércoles:
-----
12:00 | RP
A3102 | Merce
-----

Jueves:
-----
14:00 | DSBM
A3305 | Jordi Salazar
-----

Viernes:
-----
09:00 | IXT
A3004 | Marc Rosa
-----
```

We can see that the execution is correct: using this URI, the server successfully returns the timetable data, including the time, subject, class, and teacher's name.

test9

And now we are making a test9 that will print the marks of X student with X UID.

This part we really had problems, because we could reach the marks, because in the server it was a private path, not public, because we needed the UID of the student for the path for getting the marks that we wanted.

```
Python
require 'net/http'
require 'uri'
require 'json'
require_relative 'puzzle1' # Tu librería NFC

SERVER_IP = '172.20.10.3'
SERVER_PORT = 3000
```

Document: [file name.doc]	Web Berry	
Date: dd/mm/yyyy		
Rev: 01		
Page 32 of 50		

```

def fetch_data(endpoint, params = {})
  uri = URI("http://#{SERVER_IP}:#{SERVER_PORT}/#{endpoint}")
  uri.query = URI.encode_www_form(params) if params.any?
  response = Net::HTTP.get_response(uri)
  JSON.parse(response.body) if response.is_a?(Net::HTTPSuccess)
end

begin
  # 1. Leer UID del NFC
  puts "Acerca tu tarjeta NFC..."
  uid = Rfid.new.read_uid
  puts " UID: #{uid}"

  # 2. Obtener datos del estudiante
  student = fetch_data("/students/#{uid}")
  puts "\n Estudiante: #{student['name']}" if student

  # 3. Obtener notas
  marks = fetch_data("/user/#{uid}/marks", { uid: uid })
  if marks.any?
    puts "\nCalificaciones:"
    marks.each { |m| puts "  #{m['subject']}: #{m['value']}" }
  else
    puts "\nNo se encontraron calificaciones"
  end

  rescue => e
    puts " Error: #{e.message}"
end

```

After reviewing the server code , we changed several conditions and the path, this way we can reach the student mark, otherwise there would no be a possible way to enter

```

pi@guang:~/PBE/Cliente $ ruby test9.rb
Acerca tu tarjeta NFC...
UID: 5A5BC301

Calificaciones:
ICOM: 7
IPAV: 8.5

```


Document: [file name.doc]	Web Berry	
Date: dd/mm/yyyy		
Rev: 01		
Page 33 of 50		

Código Cliente Mejorado:

Mejoras implementadas respecto al código anterior

1. Separación en clases auxiliares

Antes: lectura NFC y peticiones HTTP estaban dentro de MainWindow.

Ahora:

RfidReader: gestiona lectura NFC en hilo con callbacks.

HttpRequest: realiza petición GET y parsea JSON en hilo propio.

→ Mejora: modularidad, reutilización de código y limpieza estructural.

2. Gestión de errores centralizada

Errores controlados desde dentro de RfidReader y HttpRequest.

Comunicados al hilo principal mediante GLib::Idle.add.

→ Mejora: robustez y seguridad de la interfaz gráfica.

3. Interfaz de consulta genérica

Se permite consultar cualquier tabla (timetables, tasks, marks) desde un solo campo.

El TreeView se genera dinámicamente según los datos JSON recibidos.

→ Mejora: escalabilidad y reducción de código repetido.

4. Limpieza de interfaz y control de sesión

Función clear_window para eliminar widgets anteriores.

Temporizador de inactividad con funciones específicas (start_timeout, reset_timeout, clear_timeout).

→ Mejora: interfaz limpia y control de sesión más fiable.

5. Reintento automático ante error

Si falla la lectura NFC o la autenticación, se reintenta automáticamente tras 2 segundos.

→ Mejora: experiencia de usuario más fluida, sin necesidad de intervención manual.

Conclusión

Se ha rediseñado el cliente para seguir principios de diseño modular, control de errores, y escalabilidad. Esto mejora el mantenimiento, facilita futuras ampliaciones y garantiza una ejecución más robusta.



Python

```
require "gtk3"
require_relative "LCDController"
require_relative "Rfid"
require "json"
require "net/http"
```

```
CSS_FILE      = "diseny.css"
LOGIN_MSG     = "Please,\nlogin with\nyour card"
AUTH_ERR_MSG  = "Authentication\nerror"
API_BASE      = "http://10.192.40.80:3000"
TIMEOUT_SEC   = 120
```

Aplica CSS global

```
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
```

Encapsula la lectura de NFC en un solo hilo y handler

```
class RfidReader
  def initialize(on_success, on_error = nil)
    @on_success = on_success
    @on_error    = on_error
    @thread      = nil
  end

  def start
    stop
    @thread = Thread.new do
      begin
        uid = Rfid.new.read_uid
        GLib::Idle.add { @on_success.call(uid); false }
      rescue => ex
        GLib::Idle.add { @on_error&.call(ex); false }
      end
    end
  end

  def stop
    @thread&.kill
    @thread = nil
  end
end
```

Encapsula petición HTTP GET y parseo JSON

```
class HttpRequest
```



```
def initialize(path, on_complete)
  @path          = path
  @on_complete   = on_complete
  @thread        = Thread.new { perform }
end

def perform
  begin
    uri = URI("#{API_BASE}#{@path}")
    res = Net::HTTP.get_response(uri)
    raise "HTTP #{res.code}" unless res.is_a?(Net::HTTPSuccess)
    json = JSON.parse(res.body)
    GLib::Idle.add { @on_complete.call(json, nil); false }
  rescue => ex
    GLib::Idle.add { @on_complete.call(nil, ex); false }
  end
end

class MainWindow
  def initialize(lcd)
    apply_css
    @lcd          = lcd
    @rfid_reader  = 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

  private

  def cleanup_and_quit
    @rfid_reader&.stop
    Gtk.main_quit
  end

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

  # - Pantalla de login NFC -
  def show_login
    clear_timeout
    clear_window
    @lcd.printCenter(LOGIN_MSG)

    @frame = Gtk::Frame.new
```



```
@frame.set_border_width(10)
@frame.override_background_color(:normal, blue)
@window.add(@frame)

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

@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.show_all
@rfid_reader = RfidReader.new(method(:authenticate),
method(:show_reader_error))
@rfid_reader.start
end

def show_reader_error(ex)
  @label.text = "Reader error: #{ex.message}"
  @frame.override_background_color(:normal, red)
  @lcd.printCenter("Reader\nerror")
  # reintentar
  GLib::Timeout.add_seconds(2) { @rfid_reader.start; false }
end

# - Autenticación genérica vía HttpRequest -
def authenticate(uid)
  HttpRequest.new("/students?student_id=#{uid}", lambda do |data, err|
    if err || !data["students"].is_a?(Array) || data["students"].empty?
      show_auth_error
    else
      @student_name = data["students"].first["name"]
      @uid = uid
      show_query_screen
    end
  end)
end

def show_auth_error
  @label.text = "Authentication error, try again"
  @frame.override_background_color(:normal, red)
  @lcd.printCenter(AUTH_ERR_MSG)
  # reintentar NFC
  GLib::Timeout.add_seconds(2) { @rfid_reader.start; false }
end

# - Pantalla de consulta genérica -
def show_query_screen
  clear_window
  @lcd.printCenter("Welcome\n#{@student_name}")
end
```



```
vbox = Gtk::Box.new(:vertical, 5)
vbox.margin = 10
@window.add(vbox)

# Título
label = Gtk::Label.new("Welcome #{@student_name}")
label.override_color(:normal, white)
vbox.pack_start(label, expand: false, fill: false, padding: 5)

# Entrada de tabla
@entry = Gtk::Entry.new
@entry.set_placeholder_text("timetables, tasks o marks")
@entry.signal_connect("activate") { perform_query }
vbox.pack_start(@entry, expand: false, fill: true, padding: 5)

# ScrolledWindow para resultados
@sw = Gtk::ScrolledWindow.new
@sw.set_policy(:automatic, :automatic)
@sw.set_vexpand(true)
vbox.pack_start(@sw, expand: true, fill: true, padding: 5)

# Logout
logout = Gtk::Button.new(label: "Logout")
logout.signal_connect("clicked") { show_login }
vbox.pack_start(logout, expand: false, fill: false, padding: 5)

@window.show_all
start_timeout
end

# - Ejecuta Query genérico -
def perform_query
  reset_timeout
  table = @entry.text.strip
  path = "/user/#{@uid}/#{table}"
  HttpRequest.new(path, lambda do |data, err|
    if err
      show_error(err.message)
    else
      populate_tree(data)
    end
  end)
end

# - Pinta el TreeView genérico -
def populate_tree(arr)
  @sw.remove(@sw.child) if @sw.child
  return if arr.nil? || arr.empty?

  store = Gtk::ListStore.new(*Array.new(arr.first.size, String))
```



```
tree = Gtk::TreeView.new(store)
arr.first.keys.each_with_index do |k,i|
  col = Gtk::TreeViewColumn.new(k.capitalize, Gtk::CellRendererText.new,
text: i)
  tree.append_column(col)
end
arr.each do |row|
  iter = store.append
  row.values.each_with_index { |v,i| iter[i] = v.to_s }
end
@sw.add(tree)
tree.show_all
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 automático -
def start_timeout
  clear_timeout
  @timeout_id = GLib::Timeout.add_seconds(TIMEOUT_SEC) {
    show_login; false
  }
end

def reset_timeout
  clear_timeout; start_timeout
end

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

# - Helpers color -
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 de la app
lcd = LCDController.new
MainWindow.new(lcd)
```

Document: [file name.doc]	Web Berry	
Date: dd/mm/yyyy		
Rev: 01		
Page 39 of 50		

Listas de mejoras que hemos realizado :

1. `authenticate` → la crida a `get_response` ha d'estar fora del fil principal.
2. Revisió: els **threads** són mecanismes o artefactes, igual que `GLib::Idle.add`. Haurien d'estar encapsulats dins les classes `RfidReader` i `HttpRequest`. El **parsing del JSON** també s'hauria de fer dins d'aquestes classes. Això permet que hi hagi un **únic fil per connexió HTTP**, una **única crida GET** i un **únic parseig de la resposta JSON**.
3. No s'hauria de particularitzar el codi per a cada taula ni al servidor ni al client. Al servidor, s'hauria de fer un `parse_query(req)` genèric (excepte pel criteri d'ordenació de MySQL). Al client, la taula s'hauria de dibuixar de forma genèrica a partir del paràmetre rebut (una llista de diccionaris).
4. Les classes `RfidReader` i `HttpRequest` s'haurien de programar de forma genèrica, acceptant paràmetres i un **handler** o funció de retorn per gestionar el resultat.
5. Tractament simple dels errors de parseig de consultes al servidor: construir la **query SQL (SELECT)** de forma genèrica i simple. Si no té sentit, serà la pròpia base de dades la que rebutjarà la petició.

Android Client Documentation

1. Overall Architecture

The client follows a Clean Architecture approach with the following layers:

- **UI (Presentation):** Login, Home, and Result screens.
- **Domain:** use cases (
 - `LoginUseCase`
 - `QueryTableUseCase`
).
- **Data:**
 - **Remote:** `CourseApi` (Retrofit interface).

Document: [file name.doc]	Web Berry	
Date: dd/mm/yyyy		
Rev: 01		
Page 40 of 50		

- Repository: `CourseRepository`.
- Dependency Injection (DI): `NetworkModule` using Hilt.

2. Data / Remote Layer

CourseApi

Python

```
interface CourseApi {
    @GET("{table}")
    suspend fun queryTable(
        @Path("table") table: String,
        @QueryMap filters: Map<String, String>
    ): JSONArray
}
```

- `queryTable(table, filters)`: Sends a GET request to `BASE_URL/{table}` with query parameters.

3. Data / Repository Layer

CourseRepository

Python

```
class CourseRepository @Inject constructor(
    private val api: CourseApi
) {
    suspend fun query(
        table: String,
        filters: Map<String, String>
    ): JSONArray = api.queryTable(table, filters)
}
```

- `query(table, filters)`: Calls the API and returns the resulting `JSONArray`.

4. Domain / Use Cases

LoginUseCase

Document: [file name.doc]	Web Berry	
Date: dd/mm/yyyy		
Rev: 01		
Page 41 of 50		

Python

```
class LoginUseCase @Inject constructor() {
    suspend operator fun invoke(
        username: String,
        studentId: String
    ): ResultState {
        // Perform authentication logic (dummy or real)
    }
}
```

- **invoke(username, studentId):** Executes the authentication logic and returns a **ResultState**.

QueryTableUseCase

Python

```
class QueryTableUseCase @Inject constructor(
    private val repository: CourseRepository
) {
    suspend operator fun invoke(
        table: String,
        filters: Map<String, String>
    ): ResultState {
        val jsonArray = repository.query(table, filters)
        return ResultState.Ready(jsonArray)
    }
}
```

- **invoke(table, filters):** Orchestrates data fetching and wraps the result in a **ResultState**.

5. DI / Network Module

NetworkModule

Python

```
@Module
@InstallIn(SingletonComponent::class)
object NetworkModule {
    @Provides
```

Document: [file name.doc]	Web Berry	
Date: dd/mm/yyyy		
Rev: 01		
Page 42 of 50		

```

@Singleton
fun provideOkHttpClient(): OkHttpClient =
    OkHttpClient.Builder()
        .connectTimeout(Duration.ofSeconds(10))
        .readTimeout(Duration.ofSeconds(15))
        .addInterceptor(HttpLoggingInterceptor().apply {
            level = HttpLoggingInterceptor.Level.BODY
        })
        .build()

@Provides
@Singleton
fun provideRetrofit(client: OkHttpClient): Retrofit =
    Retrofit.Builder()
        .baseUrl("https://api.example.com/")
        .client(client)

        .addConverterFactory(Json.asConverterFactory(MediaType.get("application/json")))
        .build()

@Provides
@Singleton
fun provideCourseApi(retrofit: Retrofit): CourseApi =
    retrofit.create(CourseApi::class.java)
}

```

- **provideOkHttpClient():** Configures `OkHttpClient` with timeouts and a logging interceptor.
- **provideRetrofit(client):** Builds a `Retrofit` instance with Kotlinx Serialization.
- **provideCourseApi(retrofit):** Creates the `CourseApi` implementation.

6. UI / Screens and ViewModels

LoginViewModel

Python

```

@HiltViewModel
class LoginViewModel @Inject constructor(

```

Document: [file name.doc]	Web Berry	
Date: dd/mm/yyyy		
Rev: 01		
Page 43 of 50		

```

        private val loginUseCase: LoginUseCase
    ) : ViewModel() {
        fun onLogin(username: String, studentId: String) =
            viewModelScope.launch {
                val result = loginUseCase(username, studentId)
                // Update UI state based on result
            }
    }

```

- **onLogin(username, studentId):** Triggers the login use case and updates the UI state.

ResultViewModel

Python

```

@HiltViewModel
class ResultViewModel @Inject constructor(
    private val queryTableUseCase: QueryTableUseCase
) : ViewModel() {
    fun loadResults(table: String, filters: Map<String, String>) =
        viewModelScope.launch {
            val result = queryTableUseCase(table, filters)
            // Update UI state with fetched data
        }
}

```

- **loadResults(table, filters):** Invokes the query use case and updates the UI state.

Key Composables

- **LoginScreen(viewModel: LoginViewModel):** Renders the login UI and calls **onLogin**.
- **HomeScreen():** Displays table selection and filter inputs.
- **ResultScreen(viewModel: ResultViewModel):** Shows a list of results using a **ResultCard** composable.

7. Application Class

CourseManagerApp

Document: [file name.doc]	Web Berry	
Date: dd/mm/yyyy		
Rev: 01		
Page 44 of 50		

Python

```
@HiltAndroidApp
class CourseManagerApp : Application()
```

- **CourseManagerApp:** Entry point for Hilt dependency injection.

Final Notes:

- Extend use cases with error handling and loading states.
- Map raw `JsonArray` to strongly-typed data models for better maintainability.
- Add unit tests for each layer (Domain, Data, UI).

8. Encountered Errors and Resolutions

Below is a summary of the errors encountered during development and the corresponding fixes:

1. Duplicate Kotlin Plugin Declaration

- **Error:** Plugin with id 'org.jetbrains.kotlin.android' was already requested
- **Cause:** Applying Kotlin plugin both via version catalog (`alias(libs.plugins.kotlin.android)`) and direct `kotlin("android")`.
- **Solution:** Remove the duplicate direct declaration and rely solely on the version catalog.

2. Compose Compiler vs Kotlin Version Mismatch

- **Error:** This version (1.3.2) of the Compose Compiler requires Kotlin version 1.7.20 but you appear to be using Kotlin version 1.9.23.
- **Cause:** BOM defaulted to an old compiler incompatible with Kotlin 1.9.x.
- **Solution:** Align versions by using Compose BOM 2024.03.00 (composer-compiler 1.6.1) with Kotlin 1.9.22; force `kotlinCompilerExtensionVersion` in `composeOptions`.

3. Missing org.jetbrains.kotlin.plugin.compose Plugin

- **Error:** Plugin with id 'org.jetbrains.kotlin.plugin.compose' was not found for Kotlin 1.9.23.
- **Cause:** Compose compiler plugin only required for Kotlin ≥2.0.

Document: [file name.doc]	Web Berry	
Date: dd/mm/yyyy		
Rev: 01		
Page 45 of 50		

- **Solution:** Remove `kotlin-compose` plugin entry when using Kotlin 1.9.x.

4. JavaPoet NoClassDefFoundError in Hilt AggregateDeps

- **Error:** `java.lang.NoClassDefFoundError: com.squareup.javapoet.ClassName`
- **Cause:** Hilt-compiler (2.50) required JavaPoet ≥1.13, but classpath had older version.
- **Solution:** Exclude old Javapoet from `kapt` and inject `kapt("com.squareup:javapoet:1.13.0");` add JavaPoet to `buildscript` classpath in `settings.gradle.kts` for plugin.

5. HiltAggregateDepsDebug Task Failure

- **Error:** `AggregateDepsTask$WorkerAction 'java.lang.String com.squareup.javapoet.ClassName.canonicalName()'`
- **Cause:** The Hilt Gradle plugin itself lacked JavaPoet in its classpath at plugin load.
- **Solution:** In `settings.gradle.kts`, add:

Python

```
buildscript {
    repositories { google(); mavenCentral() }
    dependencies { classpath("com.squareup:javapoet:1.13.0") }
}
```

6.

KAPT Configuration Errors

- **Error:** `:app:kaptDebugKotlin FAILED`
- **Cause:** Annotation processing environment missing JavaPoet.
- **Solution:** Ensure `kapt("com.squareup:javapoet:1.13.0")` is declared and old versions excluded.

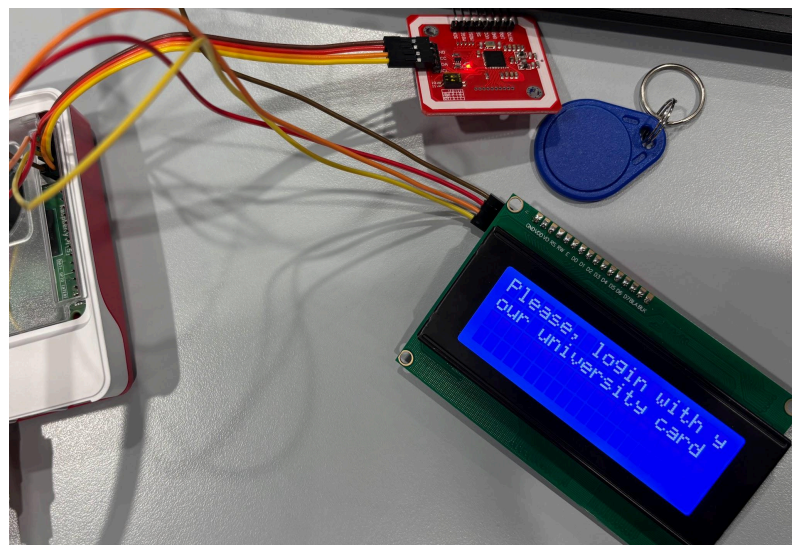
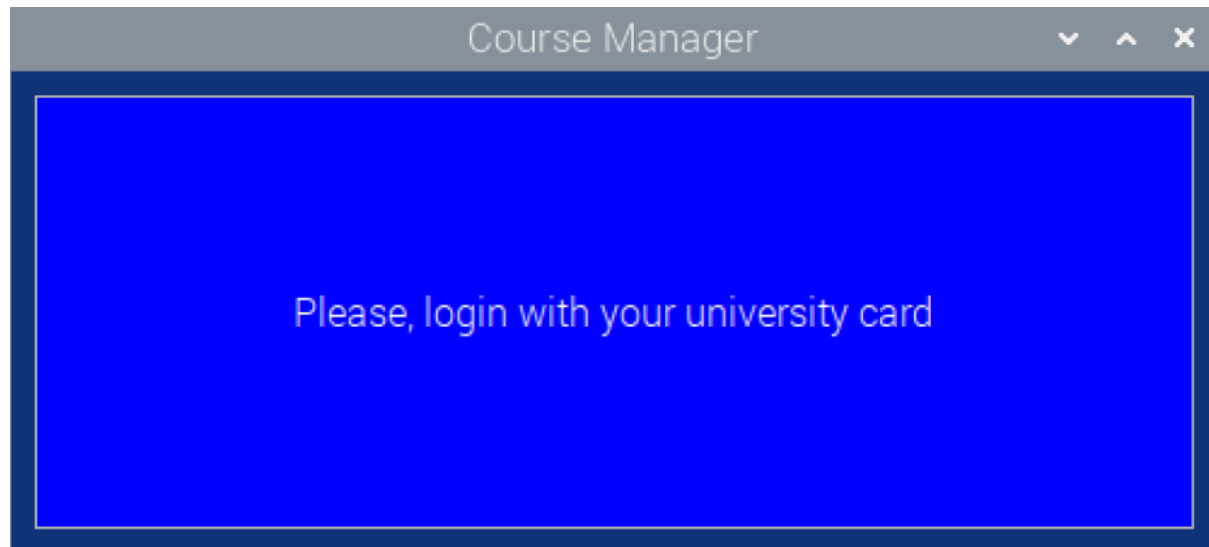
7. Repository Resolution for Javapoet

- **Error:** `Could not find com.squareup:javapoet:1.15.0.`
- **Cause:** Non-existent version requested.
- **Solution:** Use the latest available version `1.13.0` published in Google Maven.

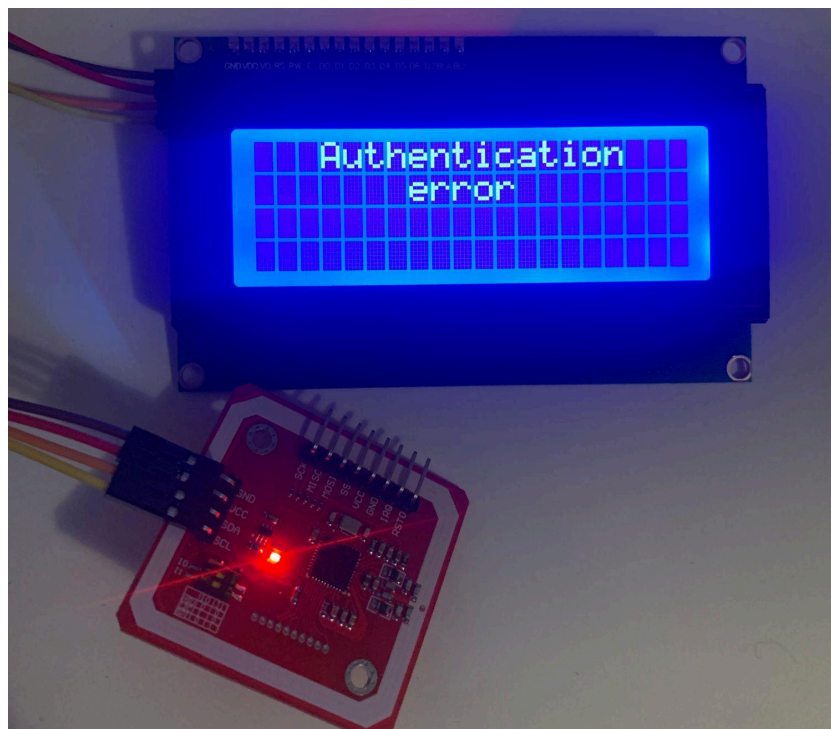
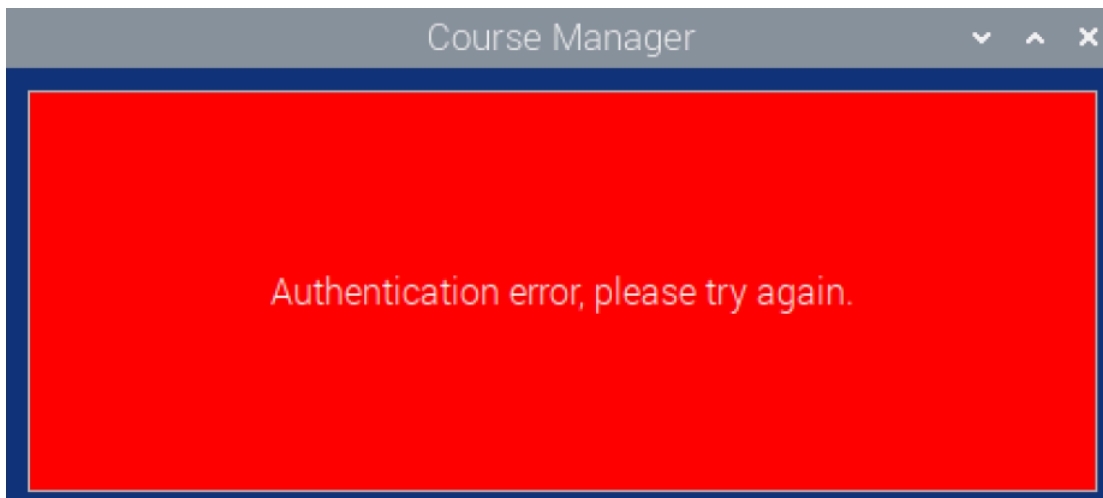


6. SYSTEM CHARACTERIZATION

Cliente :



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



Las ejecuciones de la web está en apartados anteriores, y luego las ejecuciones de android lo haremos en persona en la presentación de final report.

7. COSTS

This section presents an estimate of the project expenses, taking into account physical components, software utilities, and non-material resources like development time. The cost analysis is divided into the following categories:

1. Hardware Expenses

These refer to the tangible elements required to assemble and operate the system.

Item	Qty.	Unit Price (€)	Total (€)
Raspberry Pi 5B	4	75.00	300.00
LCD Display (HD44780)	1	9.00	9.00
Elechouse NFC)	1	7.40	12.00
Breadboard	1	6.00	6.00

Total Hardware Cost: €327.00

2. Software Resources

The software tools used throughout the project were all open source or freely available, which helped minimize costs.

Total Software Cost: €0.00

Some additional cost would be that during the project we had some peripheral devices that somehow that have been broken, so we needed an Elechouse and LCD display extra so the additional cost would be LCD+ Elechouse = 9+12 = 21€

Additional cost = 21€

Document: [file name.doc]	Web Berry	
Date: dd/mm/yyyy		
Rev: 01		
Page 49 of 50		

8. CONCLUSIONS

This project has been a great learning experience where we combined hardware and software to build a working system. We started by learning how to use the Raspberry Pi, NFC readers, and LCD screens, and then moved on to developing both the client and the server side. Even though we had no previous experience with many of these tools, we were able to create a functional prototype.

The biggest challenge was connecting the client and the server. We had many problems with the requests, the routes, and getting the data from the database, which caused several delays. In the end, the connection worked in a basic way, and the system could read the UID from a card and show personalized information from the server.

Working as a team helped us solve problems faster and learn from each other. We also learned the importance of planning, testing often, and communicating clearly to avoid confusion and delays.

Even though not everything went as planned, we are happy with the final result. We managed to build something functional and learned a lot about real-world development. This project gave us useful skills for future academic or professional work.

Document: [file name.doc]	Web Berry	
Date: dd/mm/yyyy		
Rev: 01		
Page 50 of 50		

9. REFLECTION DOCUMENT

1. Things That Could Have Been Done Better by the Team

Although we achieved the core objectives, several internal aspects could have been improved:

- Better **early-stage planning** and clearer role distribution would have helped avoid delays later on.
- **Communication** could have been more consistent, especially when tasks overlapped.
- More focus on documentation from the start would have streamlined final reporting and debugging.

2. Performance as a Work Team

Despite the challenges, the team showed strong commitment and adaptability:

- We collaborated well during problem-solving phases, especially when facing technical issues with the client-server connection.
- Everyone took initiative when needed, and peer support played a key role in overcoming setbacks.
- The learning curve was steep, but we managed to divide knowledge and help each other understand unfamiliar tools.

3. Self-Assessment

Our original goal, as stated in the team constitution document, was to aim for a high-quality, functional prototype with shared responsibility and consistent progress.

Although not all features were completed as initially planned, the final product demonstrates solid technical implementation and teamwork. We estimate our performance as a **7.5 to 8 out of 10**, taking into account the difficulties overcome, our learning process, and the degree of functionality achieved.

4. Aspects of the Subject That Could Have Been Improved

While the project offered valuable learning opportunities, certain areas could have benefited from additional support:

- More detailed initial guidelines or templates would have helped clarify expectations early on.
- A clearer timeline with key deadlines or checkpoints could have improved time management across all teams.