

Hardware / Software Project: Timetable display tablet

By:

Arthur Poulain

Yi-Rou Chen

Emma Orain

Ahmet Tomris

Submitted on :

09/06/2025

1.1 The project	4
1.2 Feasibility study.....	4
1.2.1 Technical feasibility:	4
1.2.2 Financial Feasibility:	5
1.2.3 Human Feasibility:.....	5
1.2.4 Energy review:	5
1.2.4.1 Estimated power usage of our components:	5
1.2.4.2 Battery Runtime Estimation:	6
2. Requirements.....	7
2.1 Hardware requirements	7
2.2 Software requirements.....	7
3. Module Design	7
3.1 Hardware	7
3.2 Schedule bot design	10
3.2.1 Introduction:	10
3.2.2 Structure:.....	10
The bot is composed of four Python modules:	10
3.2.2.1 File management module:.....	11
3.2.2.2 Data management module:	11
3.2.2.3 Time conversion module:	11
3.3 Database design.....	11
3.4 Tablet integrated software:.....	12
4. Implementation.....	13
4.1 Schedule bot.....	13
4.1.1 File management functions:	13
4.1.2 Data management functions:	14
4.1.3 Time conversion function:	15
4.2 Database	16
4.3 Tablet software.....	16
4.3.1 Database Module (DatabaseManager)	16
4.3.2.1 Main Interface Controller (RoomUI).....	17
4.3.2.2 Exam Mode Toggle	18

4.3 Side Panel Module (SidePanel)	18
5. Implementation	19
5.1 Transfer of the bot to the server	19
5.2 Integration of the software to the tablet	19
6. Business Plan	20
6.1 Executive Summary	20
6.2 Problem & Need	20
6.3 Target Group	20
6.4 Technical Structure	21
6.4.1 Hardware	21
6.4.2 Software	21
6.5 Key Factors Supporting the Selection of This System	21
6.6 Usage Plan	21
6.6.1 Initial Plan	21
6.6.2 Upcoming Plans	21
6.7 Risk & Solutions	22
6.9 Product Roadmap	22
7. Sources and Support	23

1. System design

1.1 The project

The goal of this project is to create a room reservation display. In a fully connected school, every student must be able to check easily the reservation of the room they have in front of them. In case the room is taken, they must be able to find another one to work on their lessons or projects. That is why a connected display can be a good idea to help students.

1.2 Feasibility study

1.2.1 Technical feasibility:

In this part we will talk about the technical feasibility for this project:

- Data source: The lecture plan is available via iCal.
- Display: A 7-inch color screen will be used to show if the room is available or not, what are the next classes in this room, and if there is an exam in this room or not.
- Connectivity: We need to be able to connect to both Wi-Fi and Ethernet. Both are supported by the Raspberry Pi 4b we are using.
- Power options: The Raspberry Pi 4b will be powered either by PoE, 2 batteries or standard 230V. The most stable solution will be chosen (usually the standard 230V).
- Presence Detection: We will use a motion sensor to detect movement and switch on the screen. The screen will be switched off after 30 seconds of inactivity.
- Software needs:
 - Display schedule and room status will be possible thanks to Python code.
 - Other information like which rooms are available, if it is a PC room or a regular classroom and if we are there is an exam or not in this room. We will not display an HTML website on the screen since it would require using a navigator and we lacked ability in this domain.

Conclusion: It is achievable from a technical point of view. We will be able to fulfill the requirements except for the display of an HTML Website.

1.2.2 Financial Feasibility:

The target budget being 150€ we decided to use the following devices:

- Raspberry Pi 4b: 35€
- PoE HAT: ~20€
- UPS HAT: 25€
- Screen: 50€
- Motion sensor: 3€
- SD card (4GB): 8€
- Various cables: ~10€

Conclusion: From a financial point of view, we can fulfill the requirements.

1.2.3 Human Feasibility:

In this part we will see what skills are needed to do this project:

- Embedded / Hardware development (power, wiring, components)
- Software development (calendar integration, UI display)

Conclusion: Those are skills we possess, and we are enough people to finish this project by the deadline (June 9th, 2025).

1.2.4 Energy review:

We need to see how long the battery will last if we only use it to power the Raspberry Pi and the screen. In this section, we perform an energy analysis using the components of the prototype, which are not the final components.

1.2.4.1 Estimated power usage of our components:

- Raspberry Pi 4b: 3.8W (idle) - 6.4W (load)
- Waveshare 7" Touchscreen: 2.5W - 3.5W
- SD Card (32 GB): ~0.5W
- Raspberry Pi Camera Module 3 Wide: 0.5 - 1 W
- UPS HAT (idle power): 0.1-0.2W

Additional Detail:

- Touchscreen: 400–700 mA depending on brightness → ~2.5–3.5W

- SD Card: 50–100 mA \rightarrow $\sim 0.5\text{W}$
- Camera Module 3 Wide: $\sim 0.5\text{--}1.5\text{W}$ depending on activity; we assumed $0.5\text{--}1\text{W}$
- UPS HAT: Consumes $\sim 0.1\text{--}0.2\text{W}$ by itself

Total system power usage:

- Mode Full usage: $\sim 9.5\text{--}11.5\text{W}$
- Mode Idle (screen off): 6W

1.2.4.2 Battery Runtime Estimation:

We use 2 Murata INR18650 (2600 mAh, 3.6V) batteries with the UPS HAT.

We are going to calculate the runtime with theses information:

- For each battery: $3.6\text{ V} \times 2.6\text{ Ah} = 9.36\text{ Wh}$
- For the two batteries: 18.72 Wh in total
- We are assuming around 80% power efficiency from the UPS board and the usable energy around 15 Wh .

With this we can estimate the runtime for the 2 modes:

- Mode Full usage (10.5W): $\sim 1.4\text{h}$ ($\sim 84\text{mins}$)
- Mode Idle (screen off): $\sim 2.5\text{h}$ ($\sim 150\text{mins}$)

Conclusion: Our analysis suggests that the system is well suited to handle scenarios involving power interruptions lasting several hours.

2. Requirements

2.1 Hardware requirements

All the different parts must cost less the 150 €. The screen needs to be big enough so we can read it, but not too much to limit energy consumption. So, we are going to use a 7-inch screen. The possibility to be powered by 230V, PoE (Power over Ethernet) and battery is required. The system needs to go to power save mode if no one is in front of the screen. For connectivity, the system must be able to connect to the ethernet and Wi-Fi. All the data must be stored on a server, with 10Mo of storage, Python and MySQL installed.

2.2 Software requirements

The system must be able to read a time schedule and display the different reservations of the room on it. If the room is already reserved, the screen must show where the nearest available room is.

The tablet must be able to stay ready and availed at every moment without consuming too much energy. It must wake up only when a human is near, to see if the room is available. The system must be actualized almost in real time to be sure that every change on the time schedule will be shown (ideally every 5 minutes). To do all of this, the software on the server must take the schedule file, extract the data, and send everything to the database. The software inside the tablet must take the content of the database every 5 minutes and refresh the schedule showing.

3. Module Design

3.1 Hardware

First, we must choose a controller. We need Wi-Fi and Ethernet ports. We also need a controller that can show a program on a screen. The simplest way to have all these features is to use a Raspberry Pi. It allows us to use a Linux environment and to install easily software and all its dependencies.

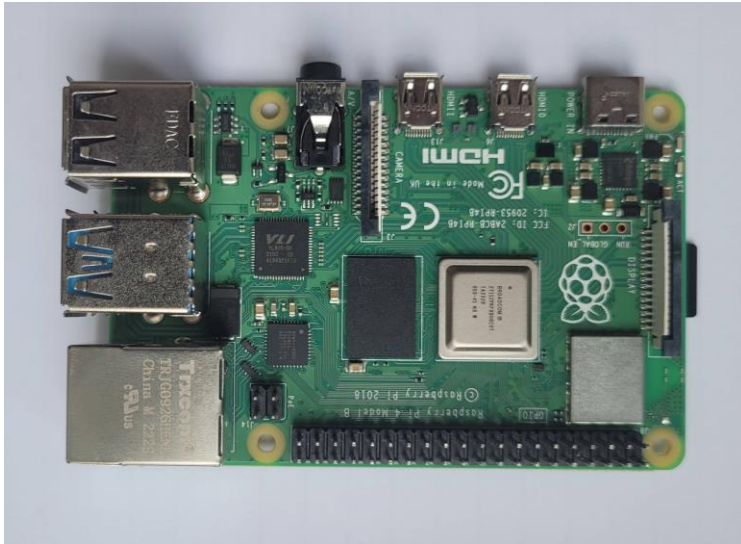


Figure 1: Raspberry Pi Model B

For the display, to allow students to with the screen, we need a touchscreen.



Figure 2: Touchscreen Waveshare 7 inches

Power management needs to be done by a single module who chooses what power source it needs to use. Here we use a UPS Hat that allows us to connect the Raspberry to the sector power supply and at the same time to the batteries. The component chooses which source he uses by checking the stability of the different sources (the sector is always the most stable). When the sector is used, the batteries are automatically charged.

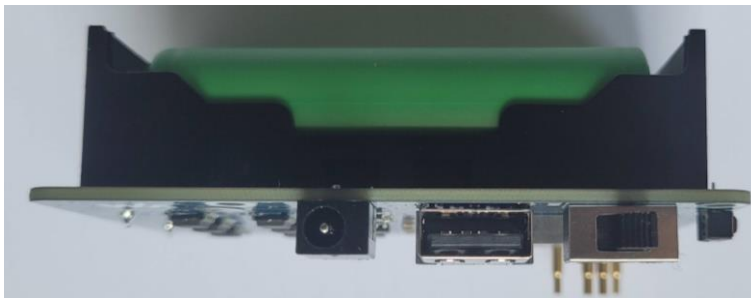


Figure 3 and 4: UPS Hat

Our last component is a camera, that will only be used to detect if someone is in front of the screen or not.

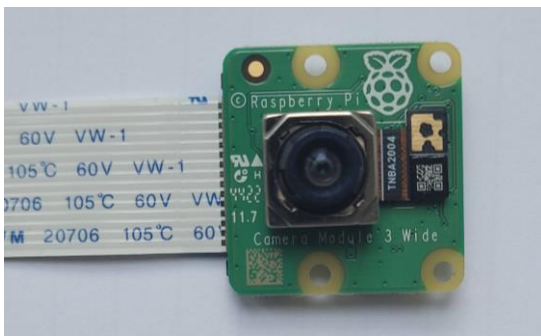


Figure 5: Raspberry camera

3.2 Schedule bot design

3.2.1 Introduction:

The program first fetches the calendar URL from the database. It then empties the "lessons" table to begin with a clean slate and adds new data. It then fetches the .ics file containing the lecture schedule, parses, and gets all event entries. Events that are relevant (with date filtering) are then pushed into the database. Finally, the local .ics file is erased to take space and maintain cleanliness. In the lecture plan, the data are displaying this way:

```
BEGIN:VEVENT
LAST-MODIFIED:20250429T120637Z
CREATED:20250204T090954Z
DTSTART:20250627T083000Z
DTSTAMP:20250429T120637Z
DTEND:20250627T100000Z
UID:a63aeb86-72f5-48a1-8371-3fa6b5efc785
SUMMARY:Klausur Mathematik 2 (90 min.) [Taraca\, R.]
LOCATION:N002\, Hörsaal\, N001\, Hörsaal\, TWI24-1
CATEGORIES:Prüfung
END:VEVENT
```

Figure 6: Example of an event in the .ics file

We can see that every piece of information is recognized by a special tag. We will pay attention to a special tag: “END:VCALENDAR” because it represents the end of the calendar and the moment the bot is supposed to stop reading information. We only need to retrieve the following data:

- Creation date (CREATED)
- Last modified date (DTSTAMP)
- Start date (DTSTART)
- End date (DTEND)
- Location (LOCATION)
- ID (UID)
- Summary (SUMMARY)
- Category (CATEGORIES)

3.2.2 Structure:

The bot is composed of four Python modules:

- The main program that connects to the database retrieves the URL and calls for every function. It will be executed every 5 minutes.
- The file management program that contains all the functions related to the file
- The data management program that contains all the functions related to the data we will get from the lecture that we are going to send to the database
- The time conversion file contains only one function that allows us to convert the time so that it can be understood by the database.

3.2.2.1 File management module:

This module is composed of 3 different functions:

- `download_file`: This function is in charge of downloading the lecture plan.
- `open_file`: This function is in charge of opening the lecture plan and getting the content of it.
- `delete_file`: This function is in charge of deleting the file to avoid using too much space on the server.

3.2.2.2 Data management module:

This module is composed of 3 different functions:

- `unfold_lines`: This function ensures that no line is split in two, preventing errors in the code.
- `data_analyse`: This function will analyze the data by adding them to a list (one list for each element of an event).
- `sent_data`: This function sorts out events into two groups: those we need (events occurring between yesterday and the end of the calendar) and those we don't need (events before yesterday). Once sorted, it sends the relevant data to the database.

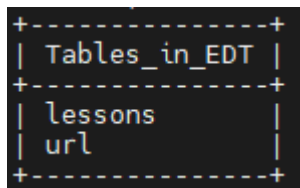
3.2.2.3 Time conversion module:

This module is only composed of 1 function:

- `convert_datetime`: This function will analyze the time we give in argument, and it will transform it in a date that can be understood by the database (YYYY-MM-DD HH:MM:SS).

3.3 Database design

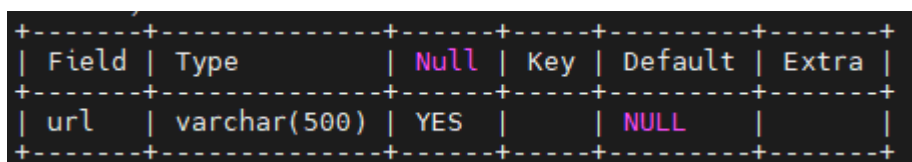
The database for this project is basic. We created a database with only two tables.



Tables_in_EDT
lessons
url

Figure 7: different tables

One table named “URL” is composed of only one column, which has the same name as the table. It allows the user to change the URL of the schedule in case of changes.



Field	Type	Null	Key	Default	Extra
url	varchar(500)	YES		NULL	

Figure 8: Table URL

The main table of the database is named “lessons”: it has all the information we can extract from the schedule file such as the beginning of a lesson or its name. The primary key is a unique ID. In the file, each lesson has a unique ID, even if the lessons are cancelled or moved.

Field	Type	Null	Key	Default	Extra
creationDate	datetime	YES		NULL	
modificationDate	datetime	YES		NULL	
startDate	datetime	YES		NULL	
endDate	datetime	YES		NULL	
location	varchar(250)	YES		NULL	
ID	varchar(250)	NO	PRI	NULL	
summary	varchar(1000)	YES		NULL	
category	varchar(250)	YES		NULL	

Figure 9: Table lessons

3.4 Tablet integrated software:

The core objective of this system is to display real-time classroom availability. To achieve this, the interface and logic were decomposed into modular components, each with clearly defined responsibilities and interfaces. The main modules are:

- **Database Management Module:** Handles communication with the MySQL database, including fetching the current class schedule, upcoming classes, and available rooms.
- **Main UI Module (RoomUI):** The central visual controller that manages layout, screen updates, and conditional UI changes (e.g., exam mode, normal display, idle status).
- **Side Panel Module:** Offers interactive options, such as finding alternative rooms and viewing course details.
- **Overlay & Stacked Layout Module:** Implements popup-style panels and layered UI transitions.
- **Timed Refresh Module:** Uses Qt's timer to periodically refresh the display, ensuring up-to-date data.

4. Implementation

4.1 Schedule bot

In this part we are going to review all the functions of the bot, what libraries they need, how they work, what are the input and output of every of them. All these functions will be called in the main program. Every of these functions will send an error message if there is any problem with the code. All the codes were coded in Python.

4.1.1 File management functions:

In this file we are going to use the following libraries:

- **requests:** (Used to download files, access APIs, etc.) allows the program to download, open, read, close and delete a file.
- **os:** (deleting files, check file paths, etc.)

download_file: This function will have as input the URL of where we are supposed to download the file, and as output a variable “check” that would take the value 1 if everything went well and 0 if there is something wrong in the code, and the name of the file. In this function, we will first download the lecture plan file. Then, we will rename the file with the name “Alle_Raume.ics”. Thereafter, we will open the file in “write mode”. Finally, we will save the file, print the path of the file and send the variable “check” and the file name.

open_file: This function takes the name of the file as input and returns two outputs: the variable “check” and the file content. We will download the content of the file "Alle_Raume.ics" and then store it in the variable “content”. In this part of the code, we will first open the file in “read mode”. Then, we will store the content of this file in the variable “content”. And finally, we will send the variables “check” and “content”.

delete_file: This function takes also the name of the file as input and will return “check”. We will use the command “os.remove(file_name)” to delete the file from the storage.

4.1.2 Data management functions:

In this file we are going to use the following libraries:

- **re**: Used for pattern matching in strings — like extracting dates, times, or cleaning data
- **convert_time**: Converts the time that was in string into a time object.
- **datetime**: To represent and manipulate dates and times
- **time_delta datetime**: to do time arithmetic (e.g., adding 30 minutes).
- **pytz**: Time zone handling.

unfold_lines: This function takes the content as the input and will return also the content as the output but modified so there is only one command by line in the content. It will make sure that we do not have this kind of line anymore.

This function finds all line breaks followed by a space or tab and joins them back into a single continuous line.

```
BEGIN:VEVENT
LAST-MODIFIED:20241218T132736Z
CREATED:20231012T073315Z
DTSTART:20251029T083000Z
DTSTAMP:20241218T132736Z
DTEND:20251029T093000Z
UID:a01e1e0f-91b6-4271-8644-8d1e93f59dd8
SUMMARY:Besprechung Studiengang Informatik [Balogh\, E.\, Schmidt\, C.\,
Judt\, A.\, Hoff\, A.\, Spandl\, Ch.\, Schneider\, J.\, Lau\, H.]
LOCATION:H108b\, Besprechungsraum
CATEGORIES:Sonstiger Termin
END:VEVENT
```

Figure 10: Example of a line that is cut in two

data_analyse: This function needs the content as input and will return the variable "check", one list by element of the event (date of creation, date of the last modification, date of start, date of end, location, ID, summary, category) and the number of events. To do this we will scan the content from the first line "BEGIN:VEVENT" to the end of the file. We will only need to read the first part of the line (the part before the first ":"). This is why we are going to cut every line in two parts (before and after the ":") and compare this first part with the usual tags we have and store it in the correct list:

```
def data_analyse(content):
    while index < len(lines) and lines[index] != "END:VEVENT":
        elements = lines[index].split(":",1)
        if len(elements) != 2:
            print("Ligne mal formatee à index {index} : {lines[index]}")
        # By looking at the first part of the line we can determine the type of information that and store it
        if elements[0] == "BEGIN" or elements[0] == "LAST-MODIFIED":
            pass #there is no relevant information about the event
        elif elements[0] == "DTSTAMP":
            date_modified = convert_time.convert_datetime(elements[1])
            last_modified.append(date_modified)
        elif elements[0] == "CREATED":
            date_created = convert_time.convert_datetime(elements[1])
            creation_date.append(date_created)
        elif elements[0] == "DTSTART" or elements[0] == "DTSTART;TZID=Europe/Berlin":
            start_date = convert_time.convert_datetime(elements[1])
            date_start.append(start_date)
        elif elements[0] == "DTEND" or elements[0] == "DTEND;TZID=Europe/Berlin":
            end_date = convert_time.convert_datetime(elements[1])
            date_end.append(end_date)
        elif elements[0] == "UID":
            event_ID.append(elements[1])
        elif elements[0] == "SUMMARY":
            summary.append(elements[1])
        elif elements[0] == "LOCATION":
            location.append(elements[1])
        elif elements[0] == "CATEGORIES":
            categories.append(elements[1])
        else:
            #there is an information that is not important for us
            pass
    index += 1
```

Figure 11: Code of the function "data_analyse"

sent_data: This function needs the variable "conn", "cursor", all the list with the data of the event and finally the number of events. And as output this function will send the variable "check". Events are sent

to the database in batches of 100. To filter them, we first define the date for "yesterday" and compare it against each event. If an event occurs on or after that date, it is added to the batch for database insertion.

```
“INSERT INTO lessons (creationDate, modificationDate, startDate, endDate, location, ID, summary,
category)                                VALUES                                (%s,%s,%s,%s,%s,%s,%s,%s)”
```

4.1.3 Time conversion function:

In this part we will need the following libraries:

- **datetime**: To represent and manipulate dates and times.
- **pytz**: Time zone handling.

conversion_datetime: This function will take as input the date in string and will have as output the date in the format imposed by the database which is the format: YYYY-MM-DD HH:MM:SS. The date is originally in RFC 5545 timestamp format (20251029T083000Z for example or Europe/Berlin: 20250407T160000).

```
from datetime import datetime #to represent and manipulate dates and times.
import pytz #Time zone handling.

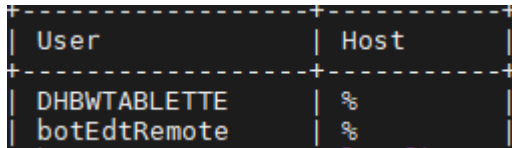
def convert_datetime(date_str):
    # this function take a time in ISO format (ex: '2024-06-01T15:00:00Z') and return the time in GMT+1 or UTC+2 (depending if we use the
    try:
        # Conversion from str to datetime
        if date_str.endswith("Z"):
            utc_time = datetime.strptime(date_str, "%Y%m%dT%H%M%S") #convert into an object
            utc_zone = pytz.utc # it give us the time zone
            utc_time = utc_zone.localize(utc_time) #the time is now in GMT+0

            # convert in timezone Europe/Paris which is the same as in Germany
            paris_zone = pytz.timezone('Europe/Paris')
            local_time = utc_time.astimezone(paris_zone)
            return local_time.strftime('%Y-%m-%d %H:%M:%S') #this return the date and time in a format that the database will understand
        else:
            local_time = datetime.strptime(date_str, "%Y%m%dT%H%M%S")
            paris_zone = pytz.timezone('Europe/Paris')
            local_time = paris_zone.localize(local_time)
            return local_time.strftime('%Y-%m-%d %H:%M:%S') #this return the date and time in a format that the database will understand
    except Exception as e:
        print(f"Error convert time : {e}")
        return None
```

Figure 12: Code of the function “convert_datetime”

4.2 Database

First, we need to download the language we are using for the database and all its dependencies. Here we chose the most common one: MySQL. After the installation we created a database using MySQL language and defined two users: one for the server and one for the tablet. Since the database is on the same server (or at least the same private network), we can set the user so he can just be used on this network (for security reasons). Same for the table of user.



User	Host
DHBWTABLETTE	%
botEdtRemote	%

Figure 13: the table of user

When the users are ready to use, we need to create the two tables and add every needed column inside the tables. When the database is ready, the bot and tablet can use it as they want.

4.3 Tablet software

To build a robust and scalable classroom status display system, we chose **Python with PyQt5** as the GUI framework and **MySQL** for backend data management. The system is organized into modular components, each responsible for a specific task. Below is a breakdown of the implementation.

4.3.1 Database Module (DatabaseManager)

- Technology: mysql.connector
- Responsibilities:
 - Connects to a MySQL database and retrieves real-time schedule data.
 - Provides clean methods (**get_schedule_for_room()** and **get_currently_busy_rooms()**) for other modules to use.
 - Provides clean methods for other modules to use:
 - **get_schedule_for_room()**: input the room number, get today's schedule of the room.
 - **get_currently_busy_rooms()**: no need for an additional input parameter as it will automatically query the database for the classroom that is currently in session
- Reason:
 - Separating data logic from UI improves maintainability and debugging.
 - Wrapping raw SQL logic inside dedicated methods increases readability and allows easier expansion (e.g., error logging or data validation).

4.3.2.1 Main Interface Controller (RoomUI)

- Technology: QWidget, QStackedLayout, QTimer, QVBoxLayout, QPropertyAnimation
- Responsibilities:
 - Acts as the root container for the entire UI.
 - Handles screen scaling, layout initialization, page switching, and update timing.
 - Reacts to schedule changes by conditionally updating the screen content.
- Key Features:
 - QStackedLayout Composition:
RoomUI uses a stacked layout to layer multiple panels:
 - Base UI
 - Touch overlay (dark transparent screen for modal behavior)
 - Slide-in SidePanel
 - MenuPanel for additional options
 - Dynamic Layout Switching:
Function **refresh_ui()** can detect when the room is available, in use, or in an exam, and dynamically switches the UI state:
 - Normal Mode: Shows current & upcoming courses with red footer.
 - Exam Mode: Focuses on displaying exam information in red background with large text.
 - Idle Mode: Shows room as available with a green footer.
 - Live Updates with QTimer:
A QTimer triggers a full interface refresh every second, ensuring timely updates based on the system clock.
 - Screen Responsiveness:
At launch, it detects actual screen size and applies a scaling factor (SCALE) to all dimensions and fonts via helper functions.
 - Animation for Side Panel Slide-in/Slide-out
Implement a smooth animation to show or hide the side panel with an overlay effect. Use QPropertyAnimation to animate the side panel's position from the right edge of the screen (off-screen) into view, or slide it back out when hiding.
 - Show: Display the side panel and overlay, bringing them to the front.
 - Hide: Hide behind the side panel and overlay once the animation is completed, ensuring the panel remains hidden after the slide-out.

4.3.2.2 Exam Mode Toggle

- Technology: Keyword matching & UI component logic
- Responsibilities:
 - Detects the presence of the keyword "KLAUSUR" in the current course title.
 - When triggered, switches to a red-background exam mode:
 - Hides upcoming schedule and footer.
 - Displays exam name prominently in the center.
- How it works:
 - Uses **.upper()** and **split("KLAUSUR")** to extract the actual exam name.
 - Applies **setText()**, layout adjustments, and **.hide()** on unrelated widgets for a clean focus view.

4.3 Side Panel Module (SidePanel)

- Technology: QWidget, QVBoxLayout, QScrollArea, QPushButton, dynamic card layouts.
- Responsibilities:
 - Displays contextual information such as:
 - Available rooms nearby
 - Alternative spaces
 - Course descriptions
 - Allows user interaction via touch or button clicks.
- UI Structure:
 - QScrollArea allows scrolling when the list of available rooms is too long.
 - Room suggestions are displayed as individual QFrame-based cards for clarity.
 - The top section includes navigation (e.g., back/close buttons) and filters.
- User Flow:
 - The side panel is typically opened when a user taps the "Find another space" button.
 - It overlays the main display without destroying or reinitializing the base UI.
 - On close, it disappears cleanly, revealing the original display state underneath.
- Reason using QStackedLayout here?
 - It allows layering the side panel above the base screen without rebuilding layouts.
 - Also supports modals (like dimmed overlays) with a shared layout context.

5. Implementation

5.1 Transfer of the bot to the server

Importing the bot that we build on a computer is not that easy. We need to create an environment where the bot can work. Here before importing the bot, we downloaded python 3 and all the needed dependencies (mysql-connector-python, pitzrequests).

When it's done, we need to create a script that launches the bot at the start of the server. It must be done smoothly so in case of network failure or server shutdown; everything can be restarted easily. To upload itself, we need to use an FTP like FileZila.

5.2 Integration of the software to the tablet

The problem here is the same as the bot. Before downloading anything to the tablet, we need to be sure that all the dependencies are downloaded. When it's done, we also need to be sure that the software will launch at the start of the tablet. So, we write the same script as the one we use on the server, and we just change the path of the main file of the software. We use a Raspberry, so for the transfer we can just use a USB stick.

6. Business Plan

6.1 Executive Summary

In educational institutions and offices, it is often not possible to instantly learn whether a room is available or not. This leads to loss of time, conflicting reservations, and inefficient use of space. We set out to solve this very concrete problem we experienced at the DHBW Ravensburg campus. We developed a display system that can work integrated with the RaPla system already used on our campus, provides real-time information, is energy-friendly, and does not impose any extra load on the user. The system works with a Raspberry Pi, a 7" touchscreen, and a camera module. The camera detects whether there is someone in front of the screen and wakes up the screen, when necessary, but no image is recorded. This provides a structure that is fully compliant with the GDPR. The aim is to make room usage transparent and efficient, not only in classrooms but also in all common areas.

6.2 Problem & Need

At DHBW Ravensburg, all classroom reservations are managed centrally through the RaPla system. However, this system is only accessible online, which leads to several practical issues in daily use:

- The current occupancy status of a room is not visible at the door.
- Finding an available nearby room can be time-consuming and inconvenient.

This project was initiated to address these shortcomings by adding a missing user interface to the existing system.

Our solution enables users to access real-time room information directly at the door, making it easier to identify whether a room is occupied or available without having to check online.

6.3 Target Group

The primary target of this project is educational institutions that use digital room reservation systems like ours. While the solution was initially developed for use on our own campus, its modular structure allows for easy adaptation in other schools or institutions.

Potential areas of application include:

- Universities and other higher education institutions
- High schools with frequent classroom changes
- Libraries and individual or group study zones
- Meeting and project rooms intended for shared use
- Coworking spaces and open-plan offices

The first phase involves piloting the system within DHBW Ravensburg. Based on user feedback, the system will be further refined. Only after this testing phase will broader implementation or sharing with other institutions be considered.

6.4 Technical Structure

6.4.1 Hardware

The system is composed of the following Hardware devices:

- Raspberry Pi 4b
- 7" touch screen
- Camera Module 3 Wide (used as a motion sensor)
- UPS HAT + 2 batteries (18650)

6.4.2 Software

The system includes the following software components:

- Rapla iCal reading and parsing codes developed in Python
- A clean and user-friendly screen design built with a Tkinter interface
- Auxiliary functions such as "Find another available space"
- Update possibility via network connection when necessary

6.5 Key Factors Supporting the Selection of This System

The following points outline why this project represents an effective solution to the problem we are facing:

- Shows the lecture plan in real time from the RaPla calendar
- GDPR compliant since the camera is only use as a motion sensor
- Energy friendly
- Easy to install, low cost

6.6 Usage Plan

6.6.1 Initial Plan

The initial phase of this project involves testing the completed prototype in a classroom at DHBW Friedrichshafen. Following the trial, we will gather feedback from both students and teachers regarding their experience and suggestions for improvement. Their input will then be carefully considered to refine and enhance the project.

6.6.2 Upcoming Plans

In this part we will see the future expectations for our system which are the following ones:

- Monitor user experience through feedback from students and teachers
- Enhance the software based on collected input
- Add extra functionalities if required (e.g., detailed reservation display with QR codes)
- Provide thorough documentation, including an installation-ready manual and supporting visuals

6.7 Risk & Solutions

The table below presents an overview of the potential risks identified in the project alongside the corresponding preventive actions we have taken to address and minimize these risks.

Risk	Taken / Measure to be Taken
GDPR and privacy concerns	Camera only detects presence, does not process images
Hardware supply problems	Alternative: Same system works with Pi Zero 2 W
Device damage or screen failure	Easy replacement thanks to modular design
Incompatibility with calendar system	RaPla → iCal compatible, all structures supported

Figure 14: Table of risk and measures taken

6.9 Product Roadmap

The following diagram illustrates the product roadmap for this project.

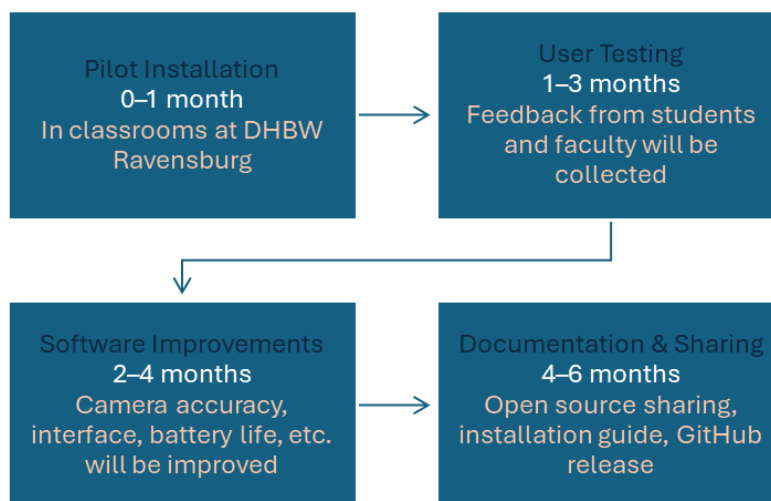


Figure 15: Product roadmap

7. Sources and Support

In this section, we will review the sources consulted and the support utilized throughout the project.

Sources:

- Raspberry Pi Foundation. (n.d.) *Raspberry Pi 4 Model B*. Retrieved June 1, 2025, from <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/>
- Geerling, J. (2020, July 8). Raspberry Pi 4 power and heat testing. Jeff Geerling Blog. Retrieved June 1, 2025, from <https://www.jeffgeerling.com/blog/2020/raspberry-pi-4-power-and-heat-testing>
- Tom's Hardware. (2020, July 30). *Best microSD cards for Raspberry Pi 4*. Retrieved June 2, 2025, from <https://www.tomshardware.com/reviews/raspberry-pi-microsd-card-performance,5977.html>
- Raspberry Pi Ltd. (2023). *Camera Module 3 Product Brief*. Retrieved June 8, 2025, from <https://www.raspberrypi.com/products/camera-module-3/>
- Pi Supply. (n.d.) *PiJuice – A portable power platform for the Raspberry Pi*. GitHub. Retrieved June 2, 2025, from <https://github.com/PiSupply/PiJuice>
- Waveshare. (n.d.) *UPS HAT for Raspberry Pi*. Retrieved June 2, 2025, from https://www.waveshare.com/wiki/UPS_HAT

Support:

- ChatGPT: Used for translation and text and rephrasing