

POLITECHNIKA WROCŁAWSKA  
WYDZIAŁ ELEKTRONIKI

---

KIERUNEK: Informatyka (INF)  
SPECJALNOŚĆ: Inżynieria Internetowa (INT)

**PROJEKT NA ZAJĘCIA SYSTEMÓW  
WBUDOWANYCH**

System do zarządzania zadaniami.

**AUTOR:**  
Sebastian Wilgosz (nr indeksu: 195963), Rafal  
Sztandera (nr indeksu: 200774)

**PROWADZĄCY:**  
dr. mgr inż. Marek Woda

**OCENA PRACY:**



# Spis treści

<b>1 Wstęp</b>	<b>3</b>
1.1 Cel projektu . . . . .	3
1.1.1 Opis aplikacji . . . . .	3
1.1.2 Definicje . . . . .	4
1.1.3 Techniki wspomagające zarządzanie sobą w czasie . . . . .	4
1.2 Opis problemu . . . . .	5
1.2.1 Wymagania . . . . .	6
1.2.2 Ograniczenia . . . . .	7
<b>2 Rozwiązanie</b>	<b>9</b>
2.1 Koncepcje związane z projektowanym systemem . . . . .	9
2.1.1 Model systemu rozproszonego . . . . .	9
2.1.2 Platforma mikroprocesorowa . . . . .	10
2.2 Tworzenie interaktywnej aplikacji . . . . .	14
2.3 Aplikacja serwerowa . . . . .	16
2.3.1 Instalacja i uruchomienie aplikacji serwerowej . . . . .	17
<b>3 Dostarczone API</b>	<b>19</b>
3.1 Obsługa błędów . . . . .	19
3.1.1 Forbidden . . . . .	19
3.1.2 Not found . . . . .	20
3.1.3 Unprocessable entity (422) . . . . .	20
3.1.4 Invalid (unauthorized) (401) . . . . .	20
3.1.5 Invalid (406) . . . . .	21
3.2 Sesje logowania . . . . .	21
3.2.1 Logowanie . . . . .	21
3.3 Zarządzanie listami . . . . .	22
3.3.1 Pobranie list zadań: . . . . .	22

3.3.2	Pobranie pojedynczej listy zadań: . . . . .	23
3.3.3	Tworzenie listy . . . . .	24
3.4	Zarządzanie zadaniami . . . . .	24
3.4.1	Pobranie zadań danej listy: . . . . .	24
3.4.2	Pobranie pojedynczego zadania: . . . . .	25
3.4.3	Tworzenie zadania . . . . .	26
3.5	Synchronizacja . . . . .	27
<b>4</b>	<b>Wnioski</b>	<b>29</b>

# Rozdział 1

## Wstęp

### 1.1 Cel projektu

Celem projektu jest stworzenie systemu mającego wspomagać zarządzaniem czasem i zadaniami przy użyciu platformy mikroprocesorowej, łącza sieciowego oraz aplikacji serwerowej.

Projekt obejmuje zaprojektowanie i stworzenie autonomicznego urządzenia wspomagającego zarządzanie czasem, serwisu internetowego udostępniającym te same funkcjonalności w tym tworzenie i prezentację zadań oraz stworzenie i wdrożenie metody synchronizacji danych pomiędzy urządzeniem, a serwisem webowym.

Platforma mikroprocesorowa będzie docelowo wykorzystywana w środowisku domowym jako element wyposażenia. Powinna być łatwo dostępna, dawać możliwość korzystania z niej bez użycia klawiatury, myszy i osobnego ekranu.

Dokument zawiera opis zakresu prac, wymagania użytkowe i funkcjonalne stawiane systemowi, projekt rozwiązania, analizę dostępnych technologii i motywację wyboru konkretnych w celu przeprowadzenia implementacji rozwiązania. W dokumencie proponujemy autorskie rozwiązanie dotyczące wersjonowania i synchronizacji danych pomiędzy aplikacją tworzoną na platformie mobilnej (system wbudowany), oraz aplikacją internetową. Wnioski zawierają analizę trafności decyzji podjętych na etapie projektowania i implementacji rozwiązania pod kątem jakości, prędkości działania i dostosowania technologii do konkretnego rozwiązania.

#### 1.1.1 Opis aplikacji

Elementem projektu, który go wyróżnia wśród dostępnych rozwiązań jest oprogramowanie. Podstawową funkcją którą realizuje, jest układanie list zadań do wyko-

nania. Funkcja wspomagająca wykonywanie zadań bazuje na technice Pomodoro. Więcej o niej w sekcji 1.1.3. Zadania posiadają parametry: nazwa, wartość określająca przynależność do jednej z list, złożoność (parametr szacowany pod kątem techniki Pomodoro) oraz priorytet (parametr wynikający z techniki związanej z matrycą Eisenohwera (więcej w sekcji 1.1.3)).

### 1.1.2 Definicje

W ramach projektu zamierzamy stworzyć dwie aplikacje. Pierwsza z nich będzie aplikacją internetową, druga będzie aplikacją na platformę mikroprocesorową. W celu ułatwienia opisu przyjmujemy, że termin *urządzenie* odnosić się będzie do platformy mikroprocesorowej oraz termin *serwis internetowy* odnosić się będzie do aplikacji internetowej.

*Zadanie* jest zdarzeniem, czynnością lub zbiorem czynności które pozostają do wykonania. *Złożoność* jest parametrem zadania, które określa ile *pomidorów* jest niezbędnych do jego wykonania. *Pomidor* jeden cykl okresów pracy i przerw techniki Pomodoro. *Priorytet* parametr zadania, wynika z techniki matrycy Eisenhowera. Może przyjmować jedną z wartości: ważne i pilne, ważne i nie pilne, nie ważne i pilne, nie ważne i nie pilne. *Działanie* to czynność, która prowadzi do osiągnięcia zamierzonego efektu. Działanie, które nie jest nastawione na określony cel, z punktu widzenia technik zarządzania czasem, jest nieprzewidywalne, niemierzalne i bezwartościowe.

### 1.1.3 Techniki wspomagające zarządzanie sobą w czasie

#### Get Things Done

Cel i działanie są pojęciami, które są szczególnie istotnie z punktu widzenia tej techniki. Według niej złe określenie przedmiotu działania jest równoznaczne z jego brakiem. Cel musi być **konkretny, realny, określony w miejscu i czasie**. Jego sformułowanie powinno być wyrażone w sposób **pozytywny**, motywujący do działania, stanowiący wyzwanie, nie deprecjonujący niczego i nikogo. Według opisu autora - Davida Allena - można ułożyć algorytmy, które pozwalają na sformułowanie dobrego celu oraz ich realizację.

#### Pomodoro

W tej technice ważnymi pojęciami są *okres pracy* i *przerwa*. Udowodniono, że umysł najlepiej pracuje, w momencie gdy jest wypoczęty, najczęściej rano. W miarę upływu

czasu umysł się męczy, tempo się zmniejsza, podatność na błędy wzrasta. Aby utrzymać jego sprawność na wysokim poziomie należy organizować regularne przerwy. Kilkuminutowy, aktywny wypoczynek spowoduje odprężenie po wysiłku i pobudzenie w związku z nową sytuacją. Tak definiowana jest *przerwa*.

*Okres pracy* jest chwilą, w której należy się skupić wyłącznie na wykonywanym zadaniu. Badania dowodzą, że jesteśmy się w stanie skupić na jednej rzeczy około 30 minut. Tyle też trwa sugerowany czas pojedynczego okresu.

*Pomidor* jest zbiorem trzech okresów pracy, pomiędzy którymi znajdują się małe, 5 minutowe przerwy. Po ukończeniu pomidora zalecane jest zrobienie sobie dłuższej przerwy (15 minut).

### Macierz Eisenhowera

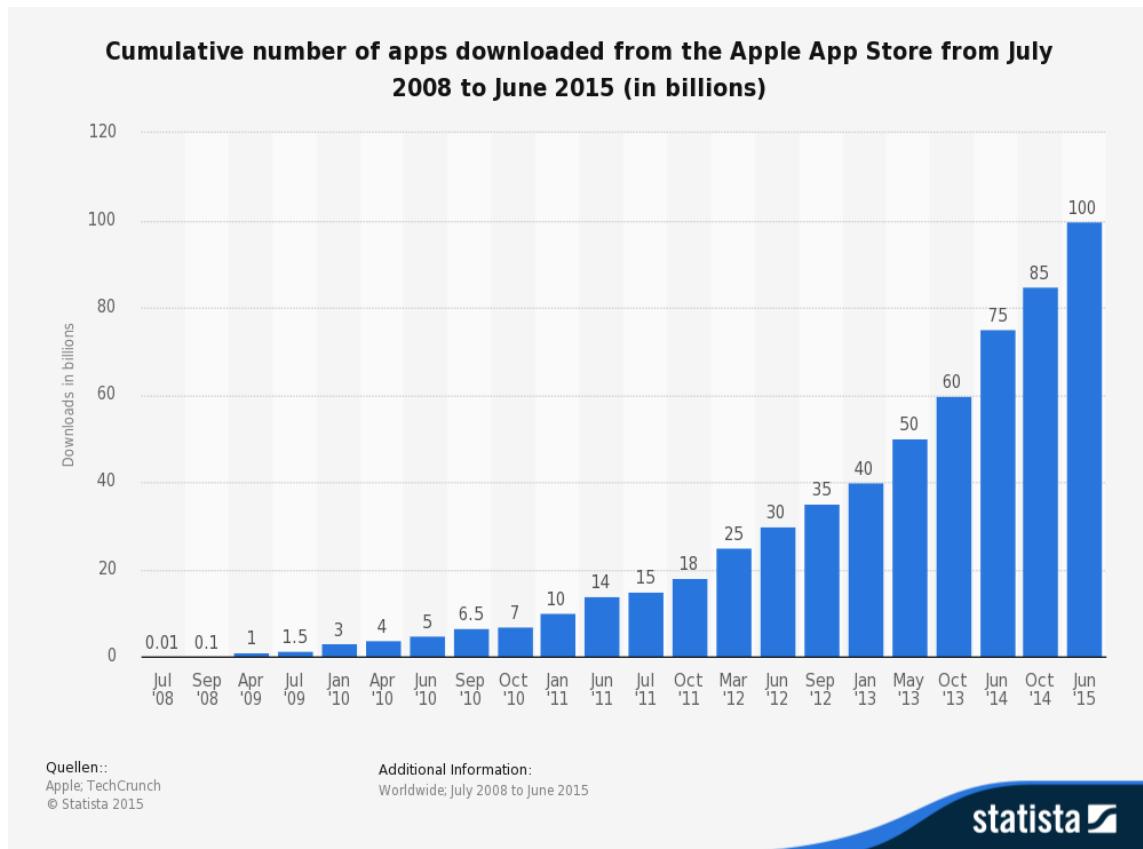
Koncepcja polega na podziale swoich zadań na grupy - ćwiartki macierzy 2x2. Kolumny noszą nazwy: ważne i nie ważne; wiersze: pilne, nie pilne. Na skrzyżowaniu ważne i pilne znajdują się zadania, które należy wykonać natychmiast. Na skrzyżowaniu nie ważne i pilne umieszczane są zadania które są naszymi marzeniami i pragnieniami. Warto je zrobić w wolnej chwili. Na skrzyżowaniu ważne i nie pilne znajdują się zadania, które są dla nas istotne, ale które możemy komuś delektować, albo wykonywać przy czyjeś współpracy. Zadań z komórki na skrzyżowaniu nie ważne i nie pilne należy unikać, i możliwie szybko usuwać.

## 1.2 Opis problemu

W dobie dynamicznego wzrostu popularności aplikacji użytkowych na platformy mobilne (rys. 1.1), oraz w momencie dynamicznego rozwoju technologii związanej z pojęciem Internet of Things, platformy mikroprocesorowe stają przed wyzwaniem zapewnienia funkcjonalności znanych z programów pisanych z myślą o jednostkach stacjonarnych oraz sprostaniem specyficzny wymaganiom dotyczącym rozmiaru, źródła zasilania i obsługi urządzeń nietypowych dla środowiska komputerowego .

Zarówno w przypadku oprogramowania obsługującego platformy mobilne [1], jak i w przypadku koncepcji Internet of Things, nie zostały ustalone normy standaryzujące proces tworzenia oprogramowania, czy konstrukcji systemów opartych na połączeniu aplikacji webowych i mobilnych lub stacjonarnych. Za cel pracy przyjęto zaprojektowanie platformy mikroprocesorowej będącej zgodną z aktualnymi trendami rozwoju inżynierii internetowej. Wybory i założenia przyjęte podczas prac są tłumaczone, tak, by w oparciu o nich, móc odpowiedzieć na pytanie o granice adaptowalności,

skalowalności i możliwości rozwoju systemów wbudowanych jako rozproszonego środowiska działania aplikacji użytkowych.



Rysunek 1.1 Wykres ilości pobrań aplikacji mobilnych dostosowanych do systemu operacyjnego popularnego producenta urządzeń mobilnych na przestrzeni ostatnich 7 lat.

### 1.2.1 Wymagania

Określenie wymagań stawianych projektowanemu systemowi zostało oparte o analizę potrzeb użytkownika na przykładzie aplikacji wspomagającej zarządzaniem czasem, oraz ogólnie-przyjętymi wyznaczkami jakimi charakteryzują się aplikacje mobilne[2].

System przeznaczony jest dla pojedynczych użytkowników, do użytku własnego. Z tej perspektywy ważnymi, ale nie kluczowymi wyznacznikami urządzenia są:

1. zapewnienie maksymalnej niezawodności

2. wytrzymałość na zniszczenia i warunki ekstremalne
3. możliwość wykonania kopii zapasowej

Kluczowymi cechami platformy, ze względu na swoje przeznaczenie i środowisko pracy są:

1. łatwe zarządzanie danymi
2. trwałość i pewność zapisu
3. wysoka dostępność do danych
4. intuicyjność interfejsu aplikacji
5. łatwa konfiguracja
6. płynność działania aplikacji
7. szybka synchronizacja

Wymagania podstawowe stawiane aplikacji:

1. możliwość anulowania operacji
2. możliwość manipulacji danymi (dodawanie/edytowanie/usuwanie)
3. interfejs w języku polskim

Dostępność tłumaczy się jako możliwość podglądu zadań z poziomu aplikacji internetowej jak i aplikacji na platformach mikroprocesorowych zsynchronizowanych z aplikacją internetową.

### **1.2.2 Ograniczenia**

Ograniczeniem dla projektu są:

1. wymiary  
platforma mikroprocesorowa powinna być możliwie płaska, oraz posiadać duży ekran dotykowy
2. zasilanie  
systemy wbudowane wymagają specjalnego zasilania

**3. dostęp do internetu**

internet jest niezbędny do dwukierunkowej synchronizacji danych z serwrem

Ograniczeniem dla projektu w przyszłości mogą być:

**1. przepisy regulujące kwestie związane z używaniem urządzeń elektrycznych w określonym środowisku (np. łazienka)**

# Rozdział 2

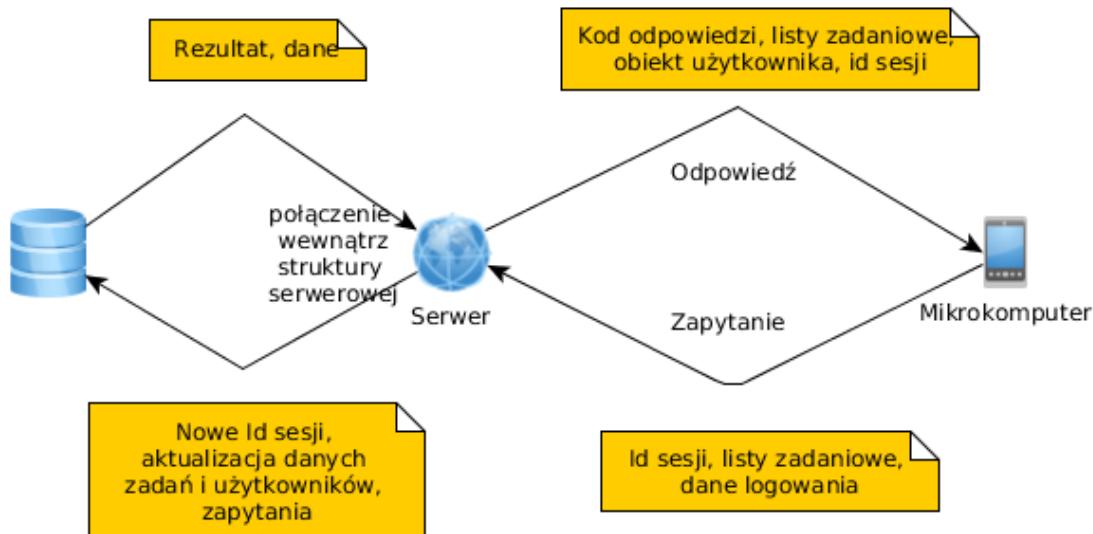
## Rozwiązanie

### 2.1 Koncepcje związane z projektowanym systemem

W ramach pracy wykonano system służący do wspomagania zarządzania czasem. Wybór tematu przykładowej implementacji dyktowany jest możliwością zastosowania technik mikroprocesorowych i koncepcji IoT (Internet of Things) oraz możliwością porównania z podobnymi rozwiązaniami. Aplikacje tego typu są tworzone z myślą o wykonawcy zleceń, jak i o organizatorach. Zasadą ich działania jest zbieranie informacji od użytkownika dotyczące niezbędnych do wykonania zadań oraz układanie ich w formie planu - terminarza. W użyciu przypominają rozbudowany kalendarz umożliwiający łatwe dodawanie nowych zadań oraz podpowiadającego efektywne zagospodarowanie czasu.

#### 2.1.1 Model systemu rozproszonego

Koncepcja Internet of Things, w uproszczeniu sprowadza się do rozbudowywania wszystkich urządzeń elektronicznych tak, by miały dostęp do Internetu. Komunikacja z urządzeniami poprzez sieć uznana za najważniejszą cechę urządzeń tworzonych wchodzących w skład takiego systemu. Często komunikacja oraz transfer danych bezpośrednio pomiędzy urządzeniem mobilnym, a komputerem PC, jest funkcją, która nie jest implementowana, ponieważ jest uznawana za nadmiarową [3]. Aby taki system mógł funkcjonować należy zapewnić możliwość nawiązania połączenia, poprawnej transmisji danych, oraz dostępu do serwera, który będzie pełnił rolę węzła głównego. Schemat rozwiązania znajduje się na rysunku 2.1.



Rysunek 2.1 Schemat modelu komunikacji i wymiany danych

### 2.1.2 Platforma mikroprocesorowa

Na rynku istnieje wiele platform umożliwiających wykonanie zakładanego projektu. Lista takich urządzeń znajduje się na rysunku 2.2. Według kategoryzacji sugerowanej w pracy [2], przyjęto, że platformami użytkowymi z punktu widzenia projektu będą te, które wspierają wieloprogramowość i obsługę wielu procesów. Te zadania pełni system operacyjny. Do celów implementacji rozwiązania wykorzystano mikrokomputer Raspberry Pi B, ponieważ jest szeroko dostępny na polskim rynku, posiada wiele gotowych podzespołów do rozbudowy, jest dobrze wspierany przez producenta, ma możliwość zainstalowania systemu operacyjnego, posiada wystarczającą ilość pamięci RAM oraz wystarczająco szybko taktowany procesor, aby uruchomić system operacyjny wraz z graficzną aplikacją użytkownika. Gooseberry oraz Mars Board również spełniały wymagania techniczne, ale nie wymagały dodatkowej konfiguracji i dokupienia modułów zapewniających łączność z internetem i z urządzeniem periferyjnym. Mikrokomputer BeagleBone Black oraz Raspberry Pi oferowały podobne funkcjonalności. O wyborze tego drugiego zadecydowała niższa cena rynkowa.

Raspberry Pi w podstawowej konfiguracji sprzętowej posiada kartę sieciową wraz z interfejsem RJ45 zgodnym z technologią Fast Ethernet. Obsługa warstwy fizycznej odbywa się w module "LAN9514" firmy Microchip, który jest zintegrowany z czterema portami USB 2.0. Komunikacja procesora z modułem opiera się magistrali

USB co narzuca ograniczenie technologiczne dotyczące prędkości przesyłu danych. Port obsługuje komunikaty sterowane za pomocą napięć (np. Wake On Lan, Zmiana statusu połączenia, "Magic Packet"). Wszystkie peryferia mikrokomputera działają w trybie serwera, łącze micro-usb służy wyłącznie do zasilania. Komunikacja z urządzeniem odbywa się na tej samej zasadzie co z innym komputerem stacjonarnym.

Nawiązanie połączenia komputer stacjonarny - mikrokomputer odbywać się może poprzez sieć lokalną, lub po odpowiednim skonfigurowaniu, poprzez internet. Podczas wykonywania zadania projektowego wykorzystywano bezpośrednie połączenie lokalne. System operacyjny dostarczony razem z urządzeniem zawiera niezbędne sterowniki, wspiera protokół dynamicznego przydzielania adresu (DHCP) oraz protokół zdalego logowania z szyfrowaniem (SSH). Dodatkowo, w celu uzyskania dostępu do trybu graficznego zainstalowano program *vncserver* oraz *vncclient* odpowiednio na urządzeniu mobilnym i komputerze z którego nawiązywano połączenie.

Jak wspomniano wyżej, integralną częścią platformy jest system operacyjny. Raspberry Pi umożliwia zastosowanie dowolnego, o ile wpiera on jego architekturę systemu (ARM w wersji 6). W tabeli 2.3 zostały zamieszczone wszystkie, które obsługują procesory ARM. Producent platformy mikroprocesorowej rozwija dedykowaną dystrybucję (Raspbian), która zawiera pełne wsparcie dla zainstalowanych na płytce urządzeń peryferyjnych, oraz część sterowników do modułów rozszerzeń. Ze względu na wsparcie dla wyświetlacza dotykowego, który rozszerza funkcjonalność implementowanego rozwiązania, oraz szerokie wsparcie dla społeczności użytkowników tej dystrybucji w internecie, autorzy projektu zadecydowali użyć jej do realizacji pracy.

Computer Board	Central Processing Unit (CPU)	Random Access Memory (RAM)	Storage	USB-connections
A13-OLinuXino-WIFI [12][13][14][15]	1 GHz	512 MB	Micro SD-card	3
Arduino Mega 2560 [16][17][18]	16 MHz	8196 B	Internal 4092 B	1
BeagleBoard [19][20]	720 MHz	2048 MB	Memory card	1
BeagleBone Black [14][20][21]	1 GHz	512 MB	Micro SD card	1
Cubietruck [14][22][23][24]	Dual-core 1GHz	2048 MB	Internal 2 GB+S-ATA	2
Elektor Linux Board [25][26]	180 MHz	32 MB	4 SD card	1
Gooseberry [27][28][29]	1 GHz	512 MB	4 GB internal + micro SD	1 Mini-USB
Hackberry [30][31]	1 GHz	512 MB	4 GB internal + SD card up to 32 GB	2
Mars Board [32][33]	1.2 GHz	1024 MB	4 GB internal + SD card up to 32 GB	2 + OTG
Netduino plus 2 [34][35][36]	168 MHz	100+ KB	Micro SD	0
Parallelia [37] [38][39]	Dual-core 1 GHz	1024 MB	16 GB micro SD	1+OTG
Raspberry Pi mod. B [40][41]	700 MHz	512 MB	SD card	2
Utilite Value [42][43]	1 GHz	512 MB	Micro SD	4 + OTG

Rysunek 2.2 Zestawienie popularnych platform mikroprocesorowych[4].

<b>General Linux Distributions</b>	<b>Lightweight OS</b>	<b>Unix-like OS</b>	<b>Entertainment</b>	<b>Special Features</b>
Alt Linux	Arch Linux	FreeBSD	GeeXbox	IPFire
Debian	Bodhi Linux	NetBSD	OpenELEC	Kali Linux
Fedora	Moebius	openSUSE	PiBox	Plan9
Gentoo Linux	PiBang Linux	Slackware Linux	PiMAME	
Lubuntu	Pidora		Raspbmc	
RISC OS	Raspbian		RaspyFi	
Ubuntu	SliTaz		Xbian	
	Tiny Core Linux			

Rysunek 2.3 Zestawienie systemów operacyjnych wspierających architekturę ARM[4].

## 2.2 Tworzenie interaktywnej aplikacji

Spośród dostępnych metod tworzenia aplikacji, których przeznaczeniem są platformy mobilne [2], w kontekście jej użycia jako elementu systemu wbudowanego możliwe do wdrożenia są: aplikacje pisane w kodzie natywnym danego urządzenia (w przypadku systemu Linux, byłby to język C++), przy wykorzystaniu niezależnego od platformy środowiska uruchomieniowego (Java/Python) lub przy wykorzystaniu technologii webowych.

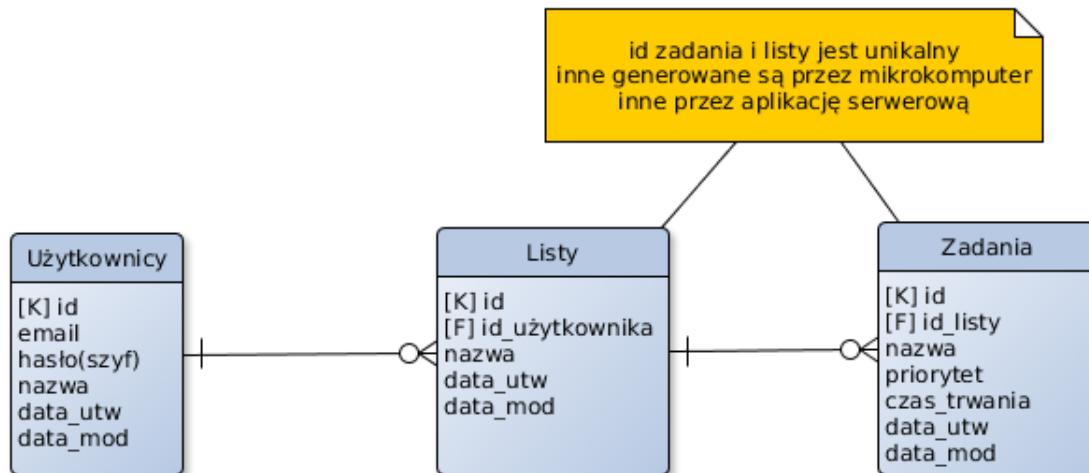
Chcąc zachować elastyczność kodu i oraz zapewnić niezależność działania aplikacji od platformy i konfiguracji sprzętowej urządzenia na którym jest uruchamiana, aplikacja powinna zostać napisana w środowisku innym niż natywne. Z punktu widzenia przeznaczenia aplikacji - środowiska, w którym ważną rolę odgrywa ekran dotykowy - wybrane narzędzie programistyczne powinno wspierać i obsługiwać gesty wykonywane przez użytkownika w sposób jednolity i prosty, tak by umożliwić programistom skupienie się na logice wytwarzanej aplikacji. Zestawem bibliotek wspomagających wytwarzanie aplikacji w trybie graficznym obsługującym wiele zdarzeń - gestów jednocześnie, jest Kivy, PyQt dla języka Python oraz JavaFX i MT4j dla języka Java,

Do wytwarzania oprogramowania na platformę mobilną wykorzystano język Python. Jest to język, który jest aktualnie najbardziej popularnym wśród użytkowników mikrokomputerów, ze względu na prostotę składni i zachowanie wysokiej elastyczności kodu i możliwości rozbudowy aplikacji. Jako zestaw narzędzi wspomagających tworzenie aplikacji graficznej wybrano Kivy, ponieważ w odróżnieniu od PyQt zawiera wsparcie dla komunikacji sieciowej, oraz własny język (*kv*) opisu elementów widoku (*widgetów*).

Synchronizacja z aplikacją webową odbywa się przy użyciu zapytań RESTowych. Zapytania RESTowe, to odpowiednie zapytania protokołu HTTP, takie jak GET, POST, PUT, DELETE, które umożliwiają manipulację danymi. Za ich pomocą wykonywane są akcje tworzenia, czytania, modyfikacji i usuwania (ang. *CRUD - Create, Read, Update and Delete*). Kivy dostarcza moduł zwany UrlRequest, działający na podobnej zasadzie co funkcje zarządzające obiektami typu XHR[5] w języku JavaScript. UrlRequest obsługuje zapytania asynchroniczne. Oznacza to, że komunikacja sieciowa obsługiwana jest w osobnym wątku; główny powinien pozostać w stanie nie zablokowanym, ponieważ wykonuje on operacje aplikacji graficznej aplikacji graficznej, niezbędne do generowania widoku i utrzymania interakcji z użytkownikiem. Po otrzymaniu wyniku, moduł UrlRequest wykonuje zdefiniowaną w jego parametrach metodę sygnalizującą program główny o rezultacie zapytania.

Aplikacja, której celem jest gromadzenie informacji o zadaniach użytkownika musi posiadać strukturę danych. Można taką strukturę tworzyć samodzielnie w postaci

struktur danych i odpowiednio powiązanych klas, oraz opisać mechanizmy manipulacji danymi umożliwiające zapis i odczyt tych struktur w plikach. Współcześnie istnieją rozwiązania zaczerpnięte z technologii webowej, które ułatwiają tworzenie, systematyzację i przechowywanie danych. Do najpopularniejszych należą ORM (*Object-relational mapping*) oraz SQL-Speaking Objects. W swoim działaniu opierają się na relacyjnych bazach danych, posiadają mechanizmy zapewniające spójność danych oraz wbudowane mechanizmy zabezpieczające przed przypadkowym i szkodliwym działaniem czynników niezależnych od algorytmu podczas operacji zapisu-/odczytu danych do własnej bazy danych. Dzięki tego typu rozwiązaniu, programista ma zapewniony dostęp do pliku przechowującego dane na zewnątrz aplikacji, które nie znikną po jego zamknięciu, oraz ma pewność co do poprawności i kompletności operacji na nim wykonywanych. Podczas implementacji aplikacji do zarządzania zadaniami wykorzystano ORM dedykowany językowi Python: *SQLAlchemy*.



Rysunek 2.4 Uproszczony model bazy danych. Symbol [K] oznacza klucz podstawowy, [F] oznacza klucz obcy.

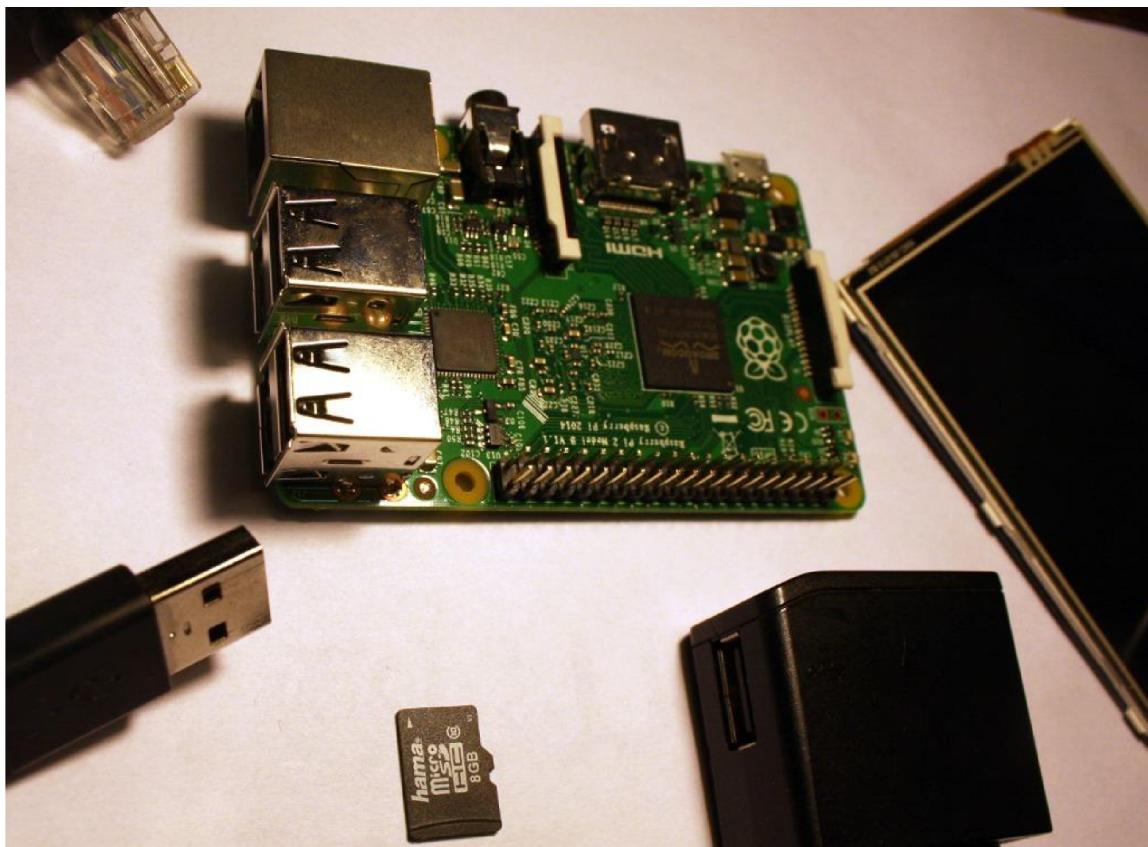
Baza danych aplikacji w uproszczeniu wygląda tak jak przedstawiono na rysunku 2.4.

Zestawienie narzędzi oraz technologii używanej przy tworzeniu aplikacji mobilnej: Instalacja i konfiguracja systemu operacyjnego:

- Raspbian - dystrybucja Linuxa
- LXDE - środowisko graficzne

Środowisko tworzenia aplikacji:

- Język - Python, wersja 2.7
- Interfejs użytkownika - Kivy
- Obsługa zapytań RESTowych - Kivy
- Implementacja modelu bazy danych (ORM) - SQLAlchemy



Rysunek 2.5 Konfiguracja systemu: mikrokomputer, pamięć, zasilanie, wyświetlacz i medium transmisyjne

### 2.3 Aplikacja serwerowa

Do zaimplementowania aplikacji serwerowej wykorzystaliśmy:

- język programowania Ruby, wersja 2.2.2 [6]
- Framework Ruby on Rails, wersja 4.2.3 [7]
- CoffeScript [8]
- Sass [9]
- Bibliotekę jQuery do języka javaScript [10]
- Bazę danych SQLlite

Wybór padł na technologię Ruby on Rails głównie ze względu na łatwość integracji z REST-owym API, ale ważnymi czynnikami były także: szybkość powstawania aplikacji internetowych w tym frameworku, oraz na obecną popularność na rynku.

### 2.3.1 Instalacja i uruchomienie aplikacji serwerowej

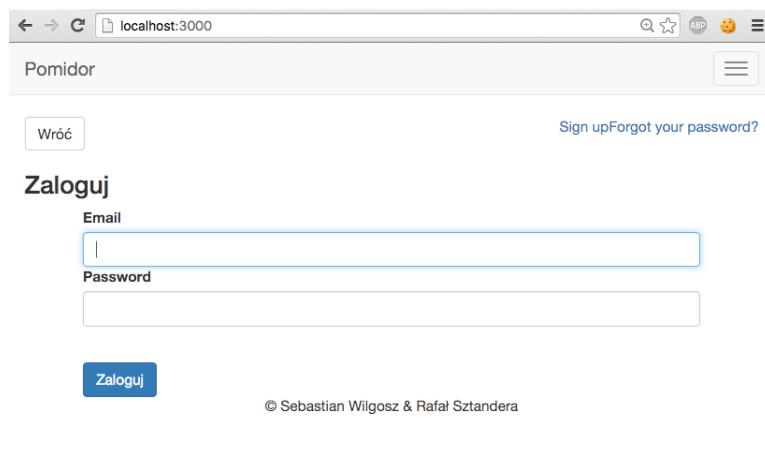
Aby uruchomić środowisko aplikacji, należy mieć zainstalowany język ruby oraz framework Ruby on Rails, wraz ze wszystkimi potrzebnymi wtyczkami. W systemie Ubuntu, aby to osiągnąć, należy najpierw zainstalować manager wersji Ruby (RVM [11]). Kiedy mamy to zrobione, należy uruchomić terminal, jąprzejść do folderu aplikacji, a następnie zainstalować wersję 2.2.2 języka Ruby oraz odpowiedni komplet gemów.

```
rvm install 2.2.2
rvm use 2.2.2@test --create
gem install bundler
bundle install
```

Po zakończeniu wykonywania powyższych komend, środowisko aplikacji powinno być zainstalowane. Kolejnym krokiem jest uruchomienie lokalnego serwera aplikacji.

```
rails s
```

Od tej pory aplikacja powinna być dostępna pod adresem <http://localhost:3000>, co widać na rysunku 2.6,



Rysunek 2.6 Widok strony głównej aplikacji

# Rozdział 3

## Dostarczone API

Poniżej zamieszczona została szczegółowa dokumentacja zaimplementowanego API. Każda sekcja składa się z przykładowego zapytania do serwera, listy wymaganych argumentów, które należy przekazać przez URL, możliwe statusy odpowiedzi, które może wysłać serwer, oraz szczegółową odpowiedzią przykładową serwera.

Każdy obiekt JSON zawiera klucz główny, definiujący typ zwracanego modelu, wewnątrz którego zdefiniowane są jego atrybuty.

### 3.1 Obsługa błędów

Poniżej znajduje się lista akceptowanych przez serwer zapytań od aplikacji, oraz lista zwracanych obiektów w formacie JSON.

Serwer nie powinien zwracać nieprzechwytywanych wyjątków, lecz obiekt JSON z odpowiednim polem statusu oraz odpowiednim opisem błędu:

#### 3.1.1 Forbidden

Ten błąd zwracany jest w przypadku próby wykonania nieautoryzowanej akcji, np. próby utworzenia listy/zadania bez wcześniejszego zalogowania.

**Format:**

```
{  
    "message": "You are not authorized to access this resource." ,  
    "status": 403  
}
```

### 3.1.2 Not found

Próba odwołania do zasobu nieistniejącego na serwerze.

**Format:**

```
{  
    "message": "Resource is not found.",  
    "status": 404  
}
```

### 3.1.3 Unprocessable entity (422)

Nieprawidłowy format zapytania, na przykład gdy w parametrach do zadania podamy argumenty bez nazwy obiektu:

```
{  
    "name": "Test 1"  
}
```

zamiast:

```
{  
    "task": {  
        "name": "Test 1"  
    }  
}
```

**Format:**

```
{  
    "message": "Unprocessable entity.",  
    "status": 422  
}
```

### 3.1.4 Invalid (unauthorized) (401)

Autoryzacja jest możliwa, ale się nie udała (przy logowaniu, jeśli podamy nieprawidłowe dane)

**Format:**

```
{  
    "message": "Incorrect email or password",  
    "access_token": null  
    "status": 401  
}
```

### 3.1.5 Invalid (406)

Występuje, gdy nie są spełnione wymogi walidacji, np

```
{  
    "message": "List cannot be saved - please , review the errors below." ,  
    "list": {  
        "errors": {  
            "title": [  
                "can't be blank" ,  
                "has already been taken , but 'List 2 is available'  
            ]  
        }  
    }  
}
```

## 3.2 Sesje logowania

### 3.2.1 Logowanie

Aby zalogować użytkownika i otrzymać odpowiadający mu klucz dostępu, należy wysłać zapytanie do serwera, przekazując w parametrach email oraz hasło. W przypadku pomyślnego logowania, serwer zwróci kod dostępu, *access\_token*, który od tej pory będzie autoryzował użytkownika.

Każde zapytanie do serwera bez podanego w parametrze klucza dostępu zwróci obiekt JSON odpowiadający błędowi (403) 3.1

```
POST http://tomato-cal.herokuapp.com/login.json
```

Wymagane parametry:

- email
- password

Statusy odpowiedzi:

- 200 - ok
- 403 - unauthorised

Odpowiedzi:

Poprawna (200)

```
{
  "status": 200,
  "access_token": "abcdefgh"
}
```

Nieprawidłowe dane logowania (403)

```
{
  "message": "Invalid email or password",
  "status": 403
}
```

### 3.3 Zarządzanie listami

#### 3.3.1 Pobranie list zadań:

Założenie jest takie, że każdy użytkownik ma dostęp wyłącznie do swoich list.

```
GET http://tomato-cal.herokuapp.com/lists?access_token="abc23@klj1309"
```

Wymagane argumenty:

- access\_token - token identyfikujący użytkownika

Statusy:

- 200 - ok
- 403 - not authorized

#### Odpowiedzi:

Poprawna (200) - zwracana jest tablica list wraz z tablicami zadań danej listy

```
{
  "lists_count": 9,
  "lists": [
    {
      "id": 1,
      "user_id": 1,
      "name": "This is sample list",
      "identifier": "serv_cklsjef323sd3",
      "created_at": "2013-05-30T13:47:41Z",
      "updated_at": "2013-05-30T13:47:41Z"
    }
  ]
}
```

### 3.3.2 Pobranie pojedynczej listy zadań:

```
GET http://tomato-cal.herokuapp.com/lists/1?access_token=abc23@klj1309"
```

Wymagane argumenty:

- access\_token - token identyfikujący użytkownika
- id - id identyfikujące daną listę

Statusy:

- itemize 200 - ok
- itemize 403 - not authorized

#### Odpowiedzi:

Poprawna (200) - zwracana jest pojedyncza lista wraz z tablicą zadań.

```
{
  "list": {
    "id": 1,
    "user_id": 1,
    "user": {
      "id": >1,
      "created_at": "2015-11-22T15:47:22.701Z",
      "updated_at": "2015-11-22T15:47:22.768Z",
      "email": "email1@example.com"
    },
    "name": "This is sample list",
    "identifier": "serv_cklsjef323sd3",
    "created_at": "2013-05-30T13:47:41Z",
    "updated_at": "2013-05-30T13:47:41Z",
    "tasks": [
      {
        "id": 1,
        "name": "Task 1",
        "identifier": "serv_cklsjef323sd3",
        "description": "Sample Text",
        "priority": 3,
        "duration": 2,
        "position_x": 12,
        "position_y": 84
      }
    ]
  }
}
```

### 3.3.3 Tworzenie listy

```
POST http://tomato-cal.herokuapp.com/lists.json
```

Wymagane argumenty

- access\_token - token identyfikujący użytkownika§
- name - unikalny tytuł

Statusy:

- 200 - ok
- 403 - forbidden
- 406 - not acceptable - validation error

Odpowiedzi:

Correct (201)

```
{
  "list": [
    {
      "id": 1,
      "user_id": 1,
      "user": {
        "id": >1,
        "created_at": "2015-11-22T15:47:22.701Z",
        "updated_at": "2015-11-22T15:47:22.768Z",
        "email": "email1@example.com"
      },
      "name": "This is sample list",
      "identifier": "serv_cklsjef323sd3",
      "created_at": "2013-05-30T13:47:41Z",
      "updated_at": "2013-05-30T13:47:41Z",
      "tasks": []
    }
  ]
}
```

## 3.4 Zarządzanie zadaniami

### 3.4.1 Pobranie zadań danej listy:

Założenie jest takie, że każdy użytkownik ma dostęp wyłącznie do swoich zadań.

```
GET http://tomato-cal.herokuapp.com/lists/1/tasks?access_token="abc23@klj1309"
```

Wymagane argumenty

- access\_token - token identyfikujący użytkownika
- list\_id - identyfikacja listy

Statusy:

- 200 - ok
- 403 - not authorized

#### Odpowiedzi:

Poprawna (200) - zwracana jest tablica zadań

```
{
  "tasks_count": 9,
  "tasks": [
    {
      "id": 1,
      "name": "Test task",
      "priority": 1,
      "created_at": "2015-11-22T15:47:22.701Z",
      "updated_at": "2015-11-22T15:47:22.768Z",
      "identifier": "serv_llopY8Kz",
      "x": 104,
      "y": 45,
      "duration": 3,
      "list_id": 1
    }
  ]
}
```

#### 3.4.2 Pobranie pojedynczego zadania:

```
GET http://tomato-cal.herokuapp.com/lists/1/tasks/1?access_token="abc23@klj1309"
```

Wymagane argumenty

- access\_token - token identyfikujący użytkownika
- list\_id - id identyfikujące daną listę
- id - id identyfikujące dane zadanie

Statusy:

- 200 - ok
- 403 - not authorized

### Odpowiedzi:

Poprawna (200) - zwracane jest pojedyncze zadanie.

```
{
  "task" => {
    "id" => 1,
    "name" => "Test task",
    "priority"=> 1,
    "created_at"=>"2015-11-22T15:47:22.701Z",
    "updated_at"=>"2015-11-22T15:47:22.768Z",
    "identifier" => "serv_13llzyI2k",
    "x"=> 104,
    "y"=> 45,
    "duration" => 3,
    "list_id" => 1,
    "list" => {
      "id"=>1,
      "name"=>"Test list",
      "identifier"=>"serv_13llzyI2k",
      "created_at"=>"2015-11-22T15:47:22.701Z",
      "updated_at"=>"2015-11-22T15:47:22.768Z",
      "user_id"=>1
    }
  }
}
```

### 3.4.3 Tworzenie zadania

```
POST http://tomato-cal.herokuapp.com/lists/1/tasks.json
```

Wymagane argumenty

- access\_token - token identyfikujący użytkownika
- title - unikalny tytuł
- list\_id - identifikator listy

Statusy:

- 201 - created
- 403 - forbidden
- 406 - not acceptable - validation error

Odpowiedzi:

Poprawna (201)

```
{  
    "task" => {  
        "id" => 1,  
        "name" => "Test task",  
        "priority"=> 1,  
        "created_at"=>"2015-11-22T15:47:22.701Z",  
        "updated_at"=>"2015-11-22T15:47:22.768Z",  
        "identifier" => "serv_13llzyI2k",  
        "x"=> 104,  
        "y"=> 45,  
        "duration" => 3,  
        "list_id" => 1,  
        "list" => {  
            "id"=>1,  
            "name"=>"Test list",  
            "identifier"=>"serv_13llzyI2k",  
            "created_at"=>"2015-11-22T15:47:22.701Z",  
            "updated_at"=>"2015-11-22T15:47:22.768Z",  
            "user_id"=>1  
        }  
    }  
}
```

## 3.5 Synchronizacja

Aby zsynchronizować bazy, należy wykonać zapytanie:

```
GET http://tomato-cal.herokuapp.com/sync
```

Wymagane argumenty:

- access\_token - token identyfikujący użytkownika
- snapshot - obraz bazy danych na urządzeniu: kolekcja wszystkich list i zadań.

Przykład:

```
curl -H 'Content-Type: application/json' -H 'Accept: application/json' -X POST http://tomato-cal.herokuapp.com/sync?access_token=42ffF2zhLai3 -d '  
{  
    lists: [  
        {  
            identifier: "rpi_4LK3kll2nza",  
            name: "Test list",  
            updated_at: "2013-05-30T13:47:41Z",  
            tasks: [  
                {  
                    identifier: "rpi_4LK3kll2nza",  
                    name: "Test task",  
                    priority: 1,  
                    created_at: "2013-05-30T13:47:41Z",  
                    updated_at: "2013-05-30T13:47:41Z"  
                }  
            ]  
        }  
    ]  
}
```

```

        "name" => "Test task",
        "priority"=> 1,
        "updated_at"=>"2015-11-22T15:47:22.768Z",
        "identifier" => 'rpi_0v9Kds2ngi',
        "x"=> 104,
        "y"=> 45,
        "duration" => 3
    }
]
}
],
}

```

Statusy:

- 200 - ok
- 403 - not authorized

Odpowiedzi:

Poprawna (200) - zwracana jest zsynchronizowana tablica list wraz z tablicami zadań danej listy

```

{
  lists: [
    {
      id: 1,
      identifier: "rpi_4LK3kll2nza",
      name: "Test list",
      created_at: "2013-05-30T13:47:41Z",
      updated_at: "2013-05-30T13:47:41Z",
      tasks: [
        {
          "id" => 1,
          "name" => "Test task",
          "priority"=> 1,
          "created_at"=>"2015-11-22T15:47:22.701Z",
          "updated_at"=>"2015-11-22T15:47:22.768Z",
          "identifier" => task.identifier,
          "x"=> 104,
          "y"=> 45,
          "duration" => 3,
          "list_id" => 1
        }
      ]
    }
  ]
}

```

## Rozdział 4

### Wnioski

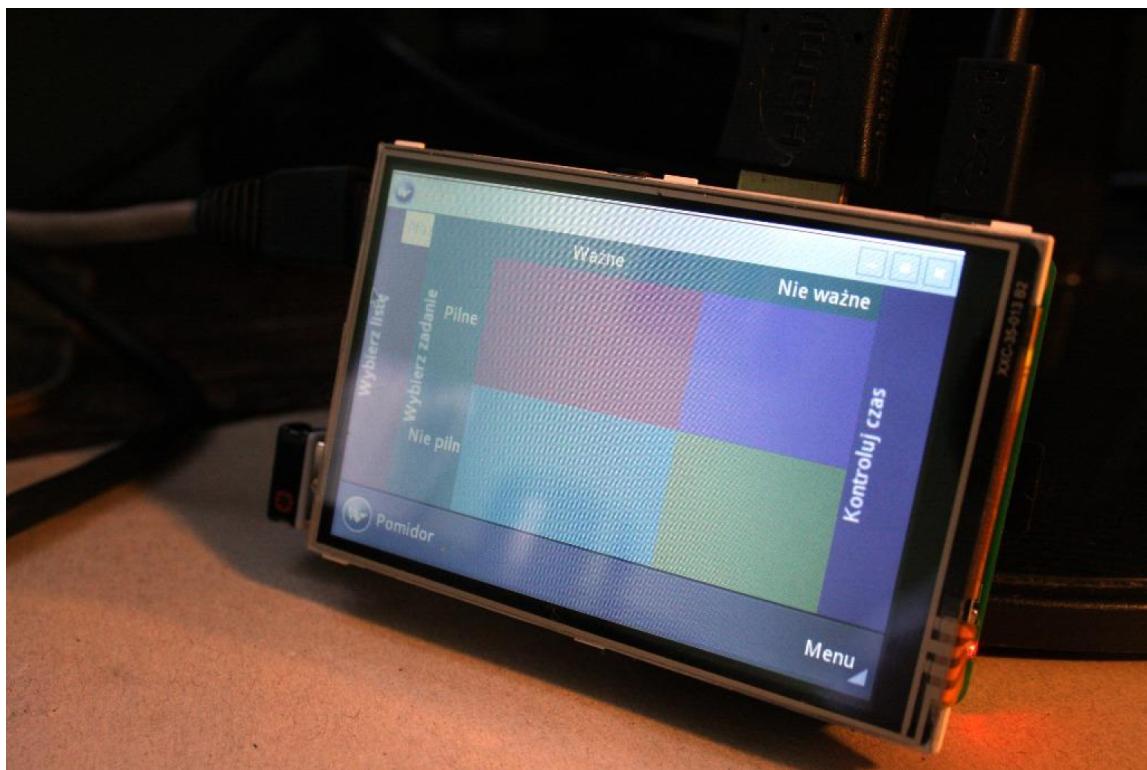
W niniejszej pracy został opisany proces przygotowania systemu wybudowanego opartego na koncepcji Internet of Things. Przedstawiono narzędzia umożliwiające zaprojektowanie, i wytworzenie takiego systemu, wraz z wyjaśnieniem decyzji podejmowanych przy wyborze poszczególnych. Opisane zostały również wybrane koncepcje wytwarzania oprogramowania i oraz autorska metoda synchronizacji danych przy użyciu asynchronicznych zapytań RESTowych oraz obiektów JSON.

Część praktyczna pracy składa się z urządzenia odpowiednio skonfigurowanego do pracy z wyświetlaczem dotykowym, aplikacji mobilnej, która łączy w sobie wszystkie założone funkcjonalności takie jak możliwość dodawania i zarządzania listami zadań, możliwością porządkowania zadań według priorytetów oraz sterowanie wykonywaniem zadań (czasomierz). Zaimplementowany został mechanizm synchronizacji oraz aplikacja webowa. Całość tworzy system, który dowodzi spełnienia postawionych założeń oraz prezentuje przykładowe zastosowanie systemu z użyciem platformy mikroprocesorowej - w tym przypadku - systemu wspomagającego zarządzaniem czasem i zadaniami.

Podczas projektowania systemu autorzy nie przewidzieli, że zastosowane rozwiązanie - zestaw bibliotek Kivy obsługujące część graficzną aplikacji - nie będzie odbslugiwane przez wbudowaną w urządzenie (Raspberry Pi) kartę graficzną. Bez wspomagania sprzętowego aplikacja działa bardzo wolno. W najnowszym wydaniu, producent Kivy zapowiedział wsparcie dla tego urządzenia.

Podsumowując, współczesna technologia umożliwia tworzenie oprogramowania użytkowego, które może w pełni integrować się z istniejącymi systemami wbudowanymi takimi jak składowe inteligentnego domu, czy urządzeń zaprojektowanych według koncepcji Internet of Things. Stworzenie systemu rozproszonego według opisanego w pracy rozwiązania wymaga niewielkiego nakładu finansowego i czasowego.

Spektrum rozwiązań jakie podobne systemy mogą oferować, jest bardzo duże i zróżnicowane, podobnie jak to ma miejsce w przypadku aplikacji mobilnych dedykowanych dla wybranych producentów systemów operacyjnych.



Rysunek 4.1 Aplikacja zainstalowana na urządzeniu

# Bibliografia

- [1] Mizouni, R.; Serhani, A. ; Dssouli, R. ; Benharref, A.: Back to Results Challenges in “mobilizing” desktop applications: a new methodology for requirements engineering, Coll. of Inf. Technol., UAE Univ., Al-Ain, United Arab Emirates, 2009
- [2] Ngu Phuc Huy; Do Van Thanh: Selecting the right mobile app paradigms Dept. of Telematics, Norwegian Univ. of Sci. & Technol., Trondheim, Norway, 2012
- [3] Razzaque, M.A.; Milojevic-Jevric, M. ; Palade, A. ; Clarke, S.: Middleware for Internet of Things: a Survey, Trinity College Dublin, Dublin, Ireland.
- [4] Chi Winth Ea; Morten Sørbo: Multi-purpose Embedded Communication Gateway: System Design and Testbed Implementation, University of Agder, 2014
- [5] Adeyeye, M.; Makitla, I. ; Fogwill, T.: Determining the signalling overhead of two common WebRTC methods: JSON via XMLHttpRequest and SIP over WebSocket, Dept. of Inf. Technol., Cape Peninsula Univ. of Technol., Cape Town, South Africa, 2013
- [6] Dokumentacja języka Ruby, <https://www.ruby-lang.org/en/>
- [7] Dokumentacja Ruby on rails, <http://guides.rubyonrails.org/>
- [8] Dokumentacja CoffeeScript, <http://coffeescript.org/>
- [9] Dokumentacja Sass, <http://sass-lang.com/>
- [10] Dokumentacja biblioteki jQuery, <https://jquery.com/>
- [11] Opis instalacji RVM (Ruby Version Manager), <https://rvm.io/rvm/install>