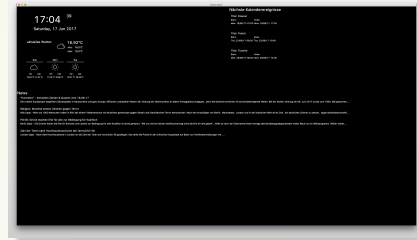


Smart Mirror

Michael R. und Timon S.

TU Dortmund - Fakultät für Informatik

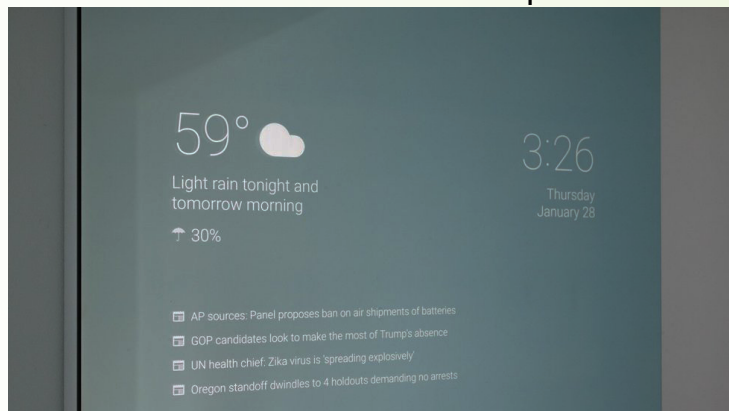
23. Juni 2017



Vergisst du auch häufig den
Geburtstag deiner Großeltern?

Lösung

So kann dir das nicht mehr passieren!



Übersicht

Verteilung der Aufgaben

technische Komponenten

Verteilungsdiagramm

UI

Programmierung

Integration einer zwei-Schichten Architektur

Aufbau eines Widgets

Von den Daten bis zur View

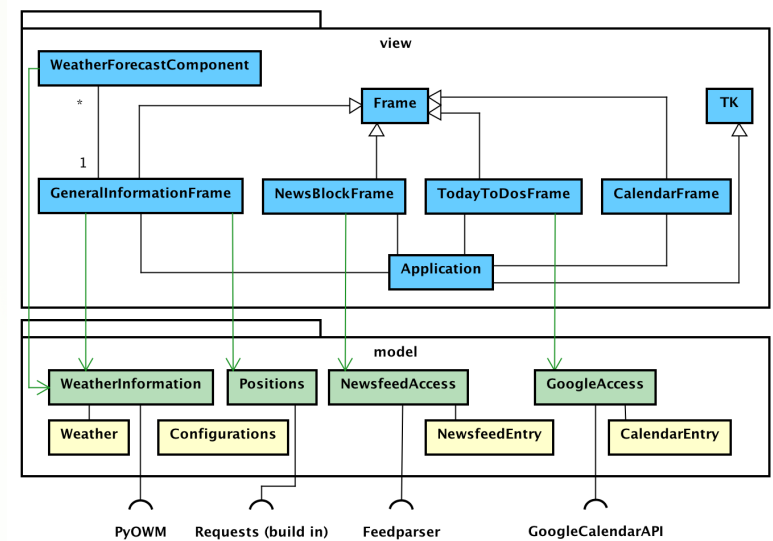
genutzte API's

JSON

Nutzung von API's

Status

Fazit



Aufbau eines Widgets

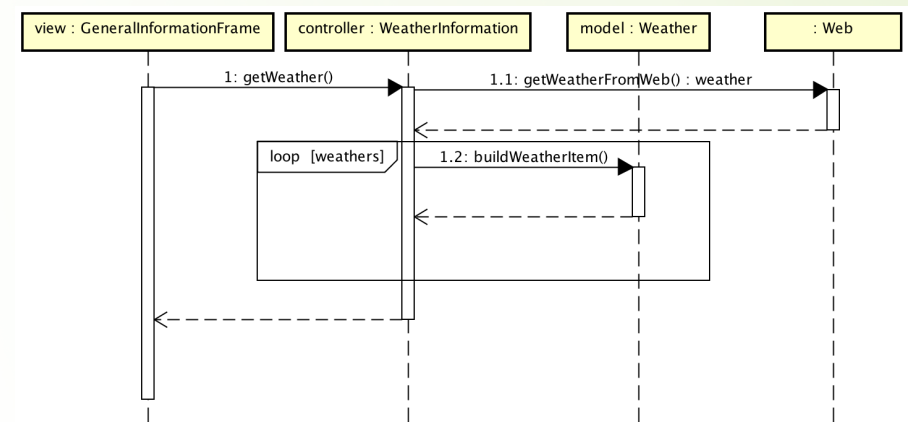
Code

```

1 class GeneralInformation(Frame):
2     temperature_frame = Frame
3     def __init__(self, parent, width, height):
4         Frame.__init__(self, parent)
5         self.time_label = Label(time_frame)
6         self.time_label.configure(bg=background_color, fg=text_color, font
7                                   =(font_type, 60))
8         self.time_label.grid(row=0, column=0, sticky="nw")
9         :
10        :
11        self.update_time()
12
13    def update_time(self):
14        new_time = time.strftime('%H:%M')
15        if new_time != self.current_time_label._getitem_-"text":
16            self.time_label.configure(text=new_time)
17        :
18        :
19        self.after(200, self.update_time)

```

Daten beziehen



- pyowm
 - Wrapper um die OpenWeatherMap-Plattform

- pyowm
- tkinter
 - um die Oberfläche zu designen

- pyowm
- tkinter
- feedparser
 - vereinfacht den Zugriff auf JSON-Dokumente

- pyowm
- tkinter
- feedparser
- requests
 - einfache Library (von Python integriert)
 - bezieht Daten von entsprechender URL
 - gibt die Daten als Raw-Values zurück: JSON, HTML

- pyowm
- tkinter
- feedparser
- requests
- apiclient, oauth2client
 - von Google bereitgestellte API's
 - Zugang zu dem Google-Kalender
 - initiale Authorisierung der Anwendung mit OAuth2

- JavaScript Object Notation
- Datenaustauschformat, welches einfach zu lesen ist
- unabhängig von Programmiersprachen
- Strukturen:
 - Key-Value
 - Liste

Beispiel

```

1 {"weather": [
2   { "id": 803,
3     "main": "Clouds",
4     "description": "clouds" } ],
5 "base": "cmc stations",
6 "main": {
7   "temp": 293.25,
8   "pressure": 1019
9 }}

```

Nutzung von API's: pyowm

- Zugriff auf Wetterdaten über einen API-Key
- ruft spezifische URL mit dem API-Key auf
- bereitet die Daten als Wetter-Objekte auf

Code

```

1 import pyowm
2 def get_weather():
3     position = get_lat_long()
4     w_api = pyowm.OWM(API_KEY)
5     curr_w = w_api.weather_at_coords(lat=pos[0], lon=pos[1]).
6         get_weather()
7     return curr_w

```

Nutzung von API's: pyowm

- Zugriff auf Wetterdaten über einen API-Key
- ruft spezifische URL mit dem API-Key auf
- bereitet die Daten als Wetter-Objekte auf

Code

```

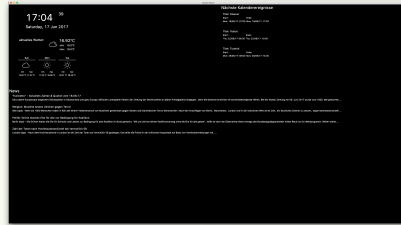
1 import pyowm
2 def get_weather():
3     position = get_lat_long()
4     w_api = pyowm.OWM(API_KEY)
5     curr_w = w_api.weather_at_coords(lat=pos[0], lon=pos[1]).
6         get_weather()
7     return curr_w

```

Zusammenfassung

- API's beziehen Daten vom Web
- API's erzeugen Objekte aus den Daten (meist JSON)
- Datenzugriff über Attribute der Objekte

- ✓ Oberfläche designed
- ✓ benötigte Libraries auf dem Pi installiert
- ✓ Spiegel zusammengebaut
- ✗ Display vermutlich defekt
- ✗ Temperatursensor auslesen
- ✗ Bewegungssensor angeschlossen
- Web-Oberfläche für Datenerfassung



Spannende Aspekte

- Absprache über Teilprojekte
- Zusammenarbeit von Hard- und Software
- man muss im Team arbeiten
- am Ende komplett Lauffähiges System in der Hand

Fazit

Spannende Aspekte

- Absprache über Teilprojekte
- Zusammenarbeit von Hard- und Software
- man muss im Team arbeiten
- am Ende komplett Lauffähiges System in der Hand

Wichtige Erfahrungen

- Absprachen sind wichtig
- es läuft nicht immer alles so, wie gewollt
- Anwendungen für spezielle Hardware schreiben ist sinnvoll