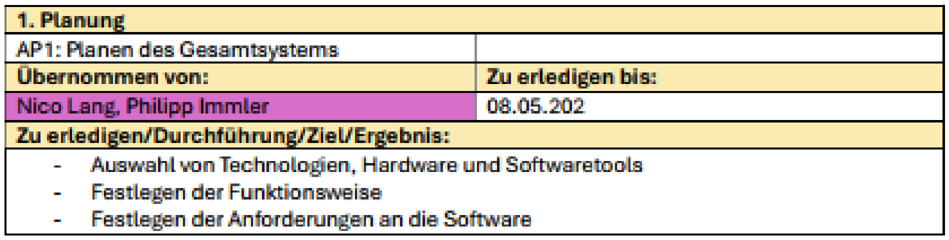
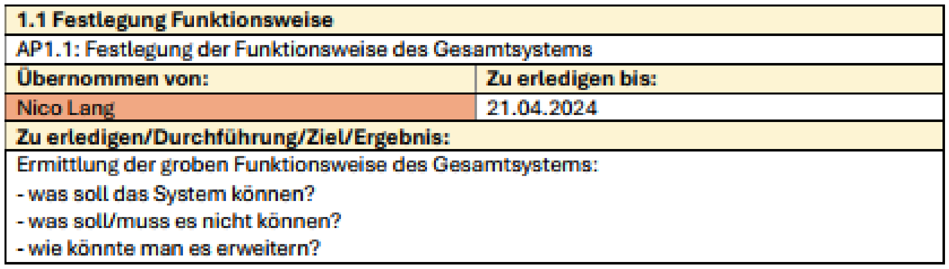
Anhangsverzeichnis

1 Anhang 1: Projektmanagement-Tools

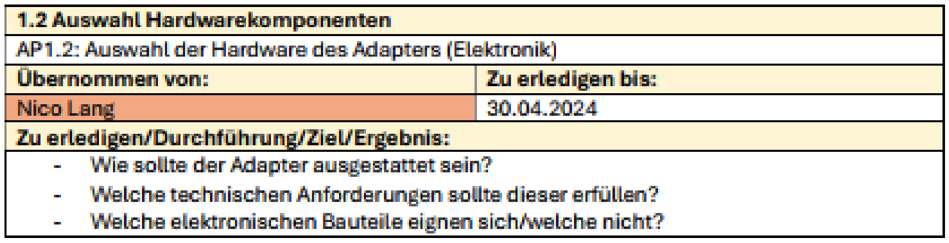
1.1 Anhang 1.1: Definition Arbeitspakete



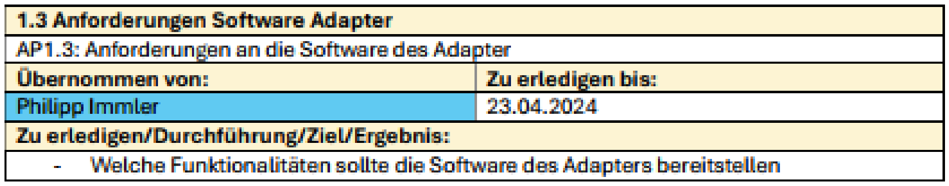
1. Planung	
AP1: Planen des Gesamtsystems	
Übernommen von:	**Zu erledigen bis:**
Nico Lang, Philipp Immler	08.05.202
Zu erledigen/Durchführung/Ziel/Ergebnis:	
- Auswahl von Technologien, Hardware und Softwaretools - Festlegen der Funktionsweise - Festlegen der Anforderungen an die Software	



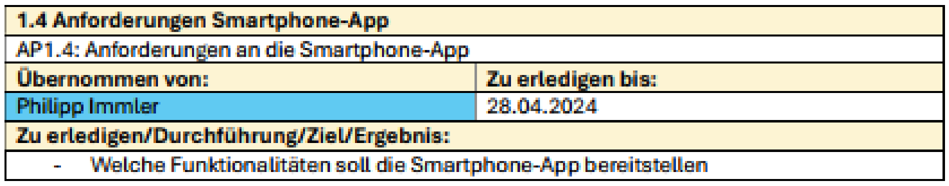
1.1 Festlegung Funktionsweise	
AP1.1: Festlegung der Funktionsweise des Gesamtsystems	
Übernommen von:	**Zu erledigen bis:**
Nico Lang	21.04.2024
Zu erledigen/Durchführung/Ziel/Ergebnis:	
Ermittlung der groben Funktionsweise des Gesamtsystems: - was soll das System können? - was soll/muss es nicht können? - wie könnte man es erweitern?	



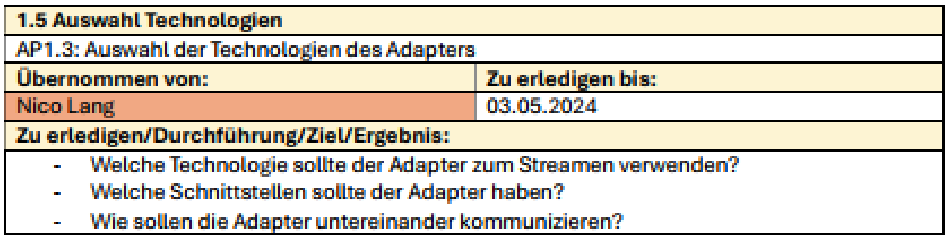
1.2 Auswahl Hardwarekomponenten	
AP1.2: Auswahl der Hardware des Adapters (Elektronik)	
Übernommen von:	**Zu erledigen bis:**
Nico Lang	30.04.2024
Zu erledigen/Durchführung/Ziel/Ergebnis:	
- Wie sollte der Adapter ausgestattet sein? - Welche technischen Anforderungen sollte dieser erfüllen? - Welche elektronischen Bauteile eignen sich/welche nicht?	



1.3 Anforderungen Software Adapter	
AP1.3: Anforderungen an die Software des Adapter	
Übernommen von:	**Zu erledigen bis:**
Philipp Immler	23.04.2024
Zu erledigen/Durchführung/Ziel/Ergebnis:	
- Welche Funktionalitäten sollte die Software des Adapters bereitstellen	



1.4 Anforderungen Smartphone-App	
AP1.4: Anforderungen an die Smartphone-App	
Übernommen von:	**Zu erledigen bis:**
Philipp Immler	28.04.2024
Zu erledigen/Durchführung/Ziel/Ergebnis:	
- Welche Funktionalitäten soll die Smartphone-App bereitstellen	



1.5 Auswahl Technologien	
AP1.3: Auswahl der Technologien des Adapters	
Übernommen von:	**Zu erledigen bis:**
Nico Lang	03.05.2024
Zu erledigen/Durchführung/Ziel/Ergebnis:	
- Welche Technologie sollte der Adapter zum Streamen verwenden? - Welche Schnittstellen sollte der Adapter haben? - Wie sollen die Adapter untereinander kommunizieren?	

1.6 Auswahl Softwaretools	
AP1.4: Auswahl der Tools für die Softwareentwicklung	
Übernommen von:	**Zu erledigen bis:**
Philipp Immler	08.05.2024
Zu erledigen/Durchführung/Ziel/Ergebnis:	
- Welche Bibliotheken/Frameworks/Programmiersprachen werden für die Software des Adapters und für die Smarphoneapp verwendet? - Welche Tools eignen sich/eignen sich nicht? - Mit welchen Tools kann man die Performance steigern?	

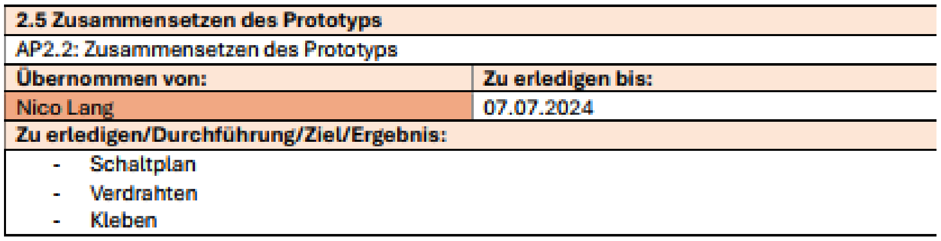
2. Entwicklung	
AP2: Entwicklung/Fertigung der Soft- und Hardware	
Übernommen von:	**Zu erledigen bis:**
Nico Lang, Philipp Immler	07.07.2024
Zu erledigen/Durchführung/Ziel/Ergebnis:	
- Herstellung des Adapters (Gehäuse, Zusammensetzen) - Entwicklung der Software des Adapters - Entwicklung der Smartphoneapp	

2.1 Entwicklung Software Adapter	
AP2.3: Entwicklung der Software des Adapters	
Übernommen von:	**Zu erledigen bis:**
Philipp Immler	06.06.2024
Zu erledigen/Durchführung/Ziel/Ergebnis:	
- Entwicklung der Software des Adapters	

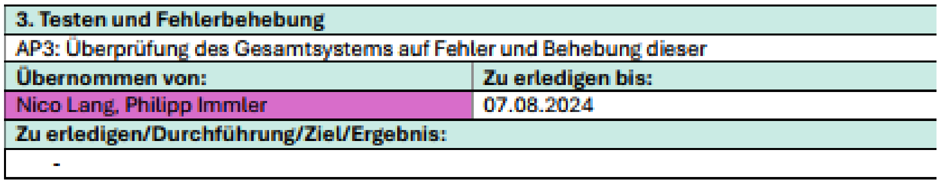
2.2 Entwicklung Smartphone-App	
AP2.4: Entwicklung/Programmierung der Smartphoneapp	
Übernommen von:	**Zu erledigen bis:**
Philipp Immler	02.07.2024
Zu erledigen/Durchführung/Ziel/Ergebnis:	
- Entwicklung der Smartphoneapp	

2.3 Design Adaptergehäuse	
AP2.5: Entwicklung/Design des Adaptergehäuses	
Übernommen von:	**Zu erledigen bis:**
Nico Lang	07.06.2024
Zu erledigen/Durchführung/Ziel/Ergebnis:	
- Design des Modells für das Adaptergehäuse in einem CAD - Wie soll das Gehäuse grob aussehen/worauf sollte Wert gelegt werden? (schlicht, modern, einfach) - Wie kann man das Gehäuse möglichst praktisch und kompakt designen? - Wie kann man das Gehäuse sicher/robust designen? - Wie löst man die Wärmeableitung?	

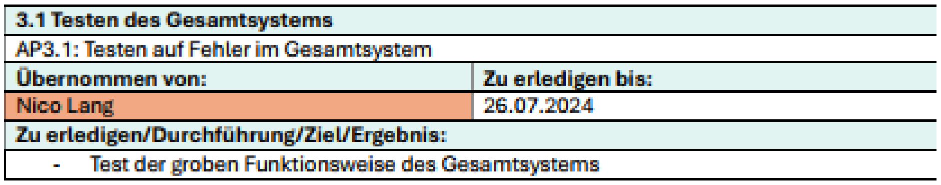
2.4 Fertigung Adaptergehäuse	
AP2.6: Fertigung/Herstellung des Adaptergehäuses	
Übernommen von:	**Zu erledigen bis:**
Nico Lang	09.06.2024
Zu erledigen/Durchführung/Ziel/Ergebnis:	
- Fertigung des zuvor designten Gehäuses für den Adapter - Welche Fertigungsverfahren kommen in Frage? - Welches Fertigungsverfahren wird verwendet? - Wie viel kostet die Herstellung eines Gehäuses?	



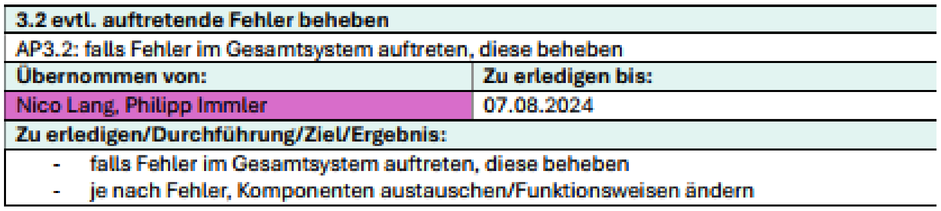
2.5 Zusammensetzen des Prototyps	
AP2.2: Zusammensetzen des Prototyps	
Übernommen von:	**Zu erledigen bis:**
Nico Lang	07.07.2024
Zu erledigen/Durchführung/Ziel/Ergebnis:	
- Schaltplan - Verdrahten - Kleben	



3. Testen und Fehlerbehebung	
AP3: Überprüfung des Gesamtsystems auf Fehler und Behebung dieser	
Übernommen von:	**Zu erledigen bis:**
Nico Lang, Philipp Immler	07.08.2024
Zu erledigen/Durchführung/Ziel/Ergebnis:	
-	

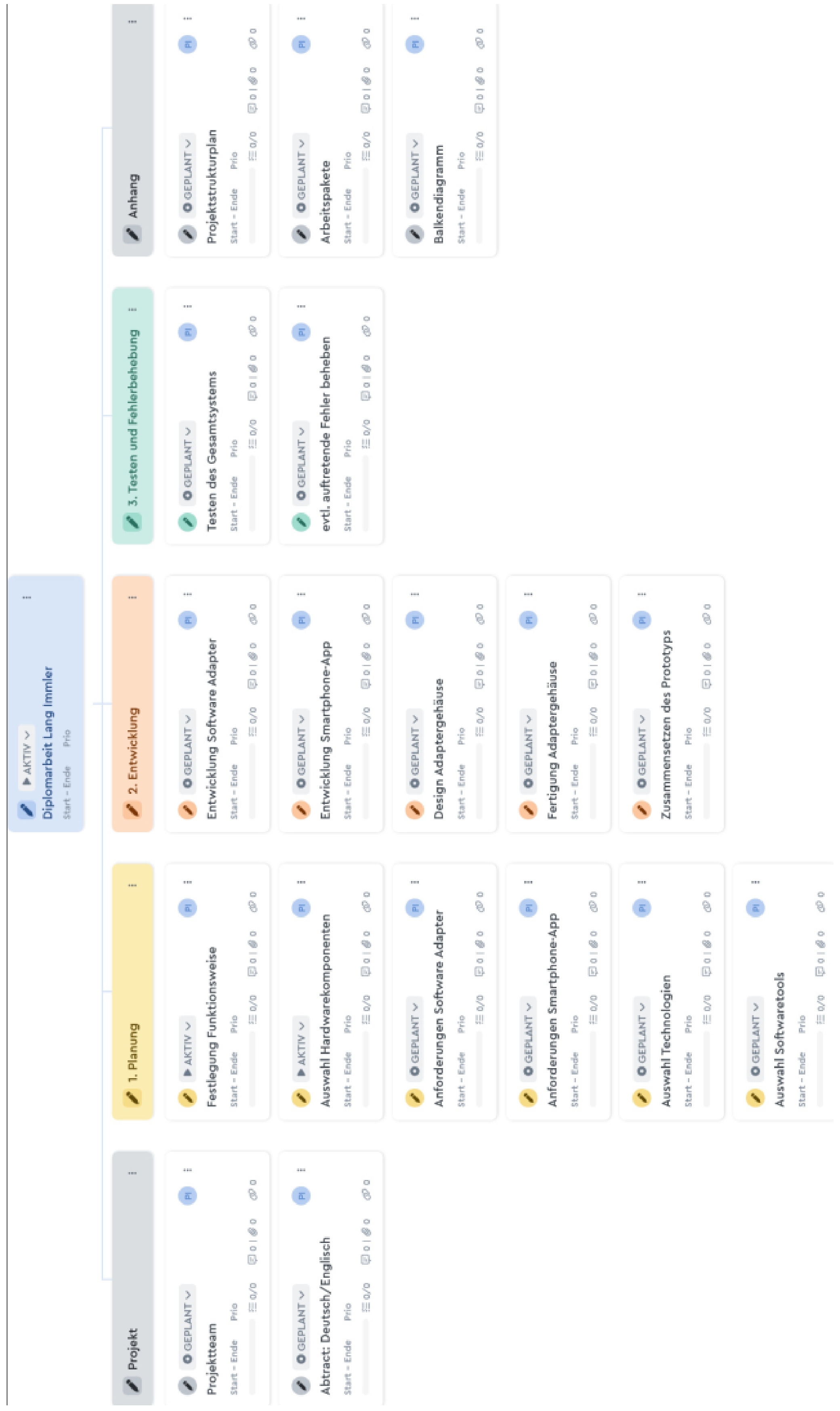


3.1 Testen des Gesamtsystems	
AP3.1: Testen auf Fehler im Gesamtsystem	
Übernommen von:	**Zu erledigen bis:**
Nico Lang	26.07.2024
Zu erledigen/Durchführung/Ziel/Ergebnis:	
- Test der groben Funktionsweise des Gesamtsystems	

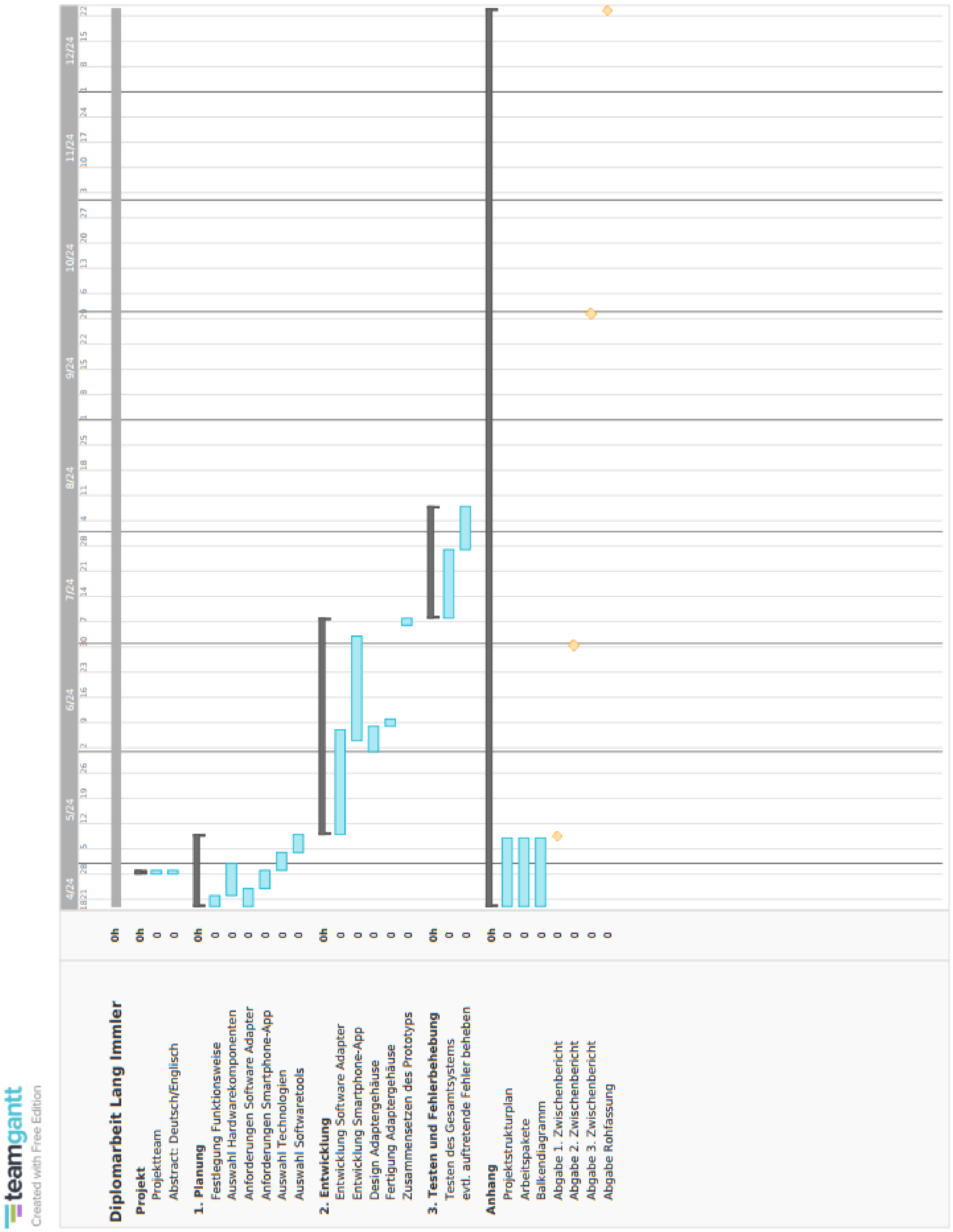


3.2 evtl. auftretende Fehler beheben	
AP3.2: falls Fehler im Gesamtsystem auftreten, diese beheben	
Übernommen von:	**Zu erledigen bis:**
Nico Lang, Philipp Immler	07.08.2024
Zu erledigen/Durchführung/Ziel/Ergebnis:	
- falls Fehler im Gesamtsystem auftreten, diese beheben - je nach Fehler, Komponenten austauschen/Funktionsweisen ändern	

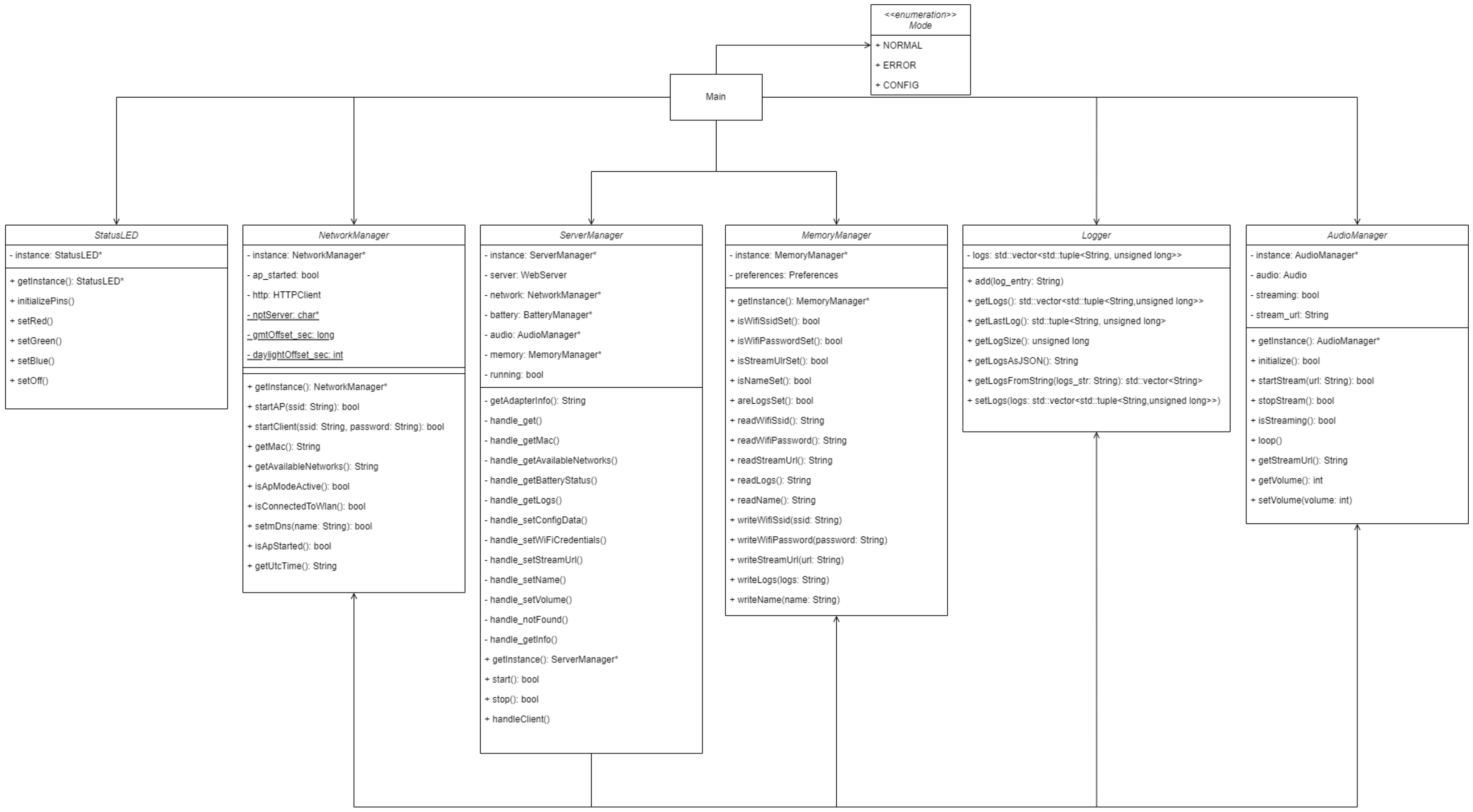
1.2 Anhang 1.2: Projektstrukturplan



1.3 Anhang 1.3: Gantt-Diagramm



2 Anhang 2: UML-Klassendiagramm Adapter



<<enumeration>> Mode
+ NORMAL
+ ERROR
+ CONFIG
Main
StatusLED
- instance: StatusLED*
+ getInstance(): StatusLED*
+ initializePins()
+ setRed()
+ setGreen()
+ setBlue()
+ setOff()
NetworkManager
- instance: NetworkManager*
- ap_started: bool
- http: HTTPClient
- ntpServer: char*
- gmtOffset_sec: long
- daylightOffset_sec: int
+ getInstance(): NetworkManager*
+ startAP(ssid: String): bool
+ startClient(ssid: String, password: String): bool
+ getMac(): String
+ getAvailableNetworks(): String
+ isApModeActive(): bool
+ isConnectedToWlan(): bool
+ setmDns(name: String): bool
+ isApStarted(): bool
+ getUtcTime(): String
ServerManager
- instance: ServerManager*
- server: WebServer
- network: NetworkManager*
- battery: BatteryManager*
- audio: AudioManager*
- memory: MemoryManager*
- running: bool
- getAdapterInfo(): String
- handle_get()
- handle_getMac()
- handle_getAvailableNetworks()
- handle_getBatteryStatus()
- handle_getLogs()
- handle_setConfigData()
- handle_setWiFiCredentials()
- handle_setStreamUrl()
- handle_setName()
- handle_setVolume()
- handle_notFound()
- handle_getInfo()
+ getInstance(): ServerManager*
+ start(): bool
+ stop(): bool
+ handleClient()
MemoryManager
- instance: MemoryManager*
- preferences: Preferences
+ getInstance(): MemoryManager*
+ isWifiSsidSet(): bool
+ isWifiPasswordSet(): bool
+ isStreamUlrSet(): bool
+ isNameSet(): bool
+ areLogsSet(): bool
+ readWifiSsid(): String
+ readWifiPassword(): String
+ readStreamUrl(): String
+ readLogs(): String
+ readName(): String
+ writeWifiSsid(ssid: String)
+ writeWifiPassword(password: String)
+ writeStreamUrl(url: String)
+ writeLogs(logs: String)
+ writeName(name: String)
Logger
- logs: std::vector<std::tuple<String, unsigned long>>
+ add(log_entry: String)
+ getLogs(): std::vector<std::tuple<String,unsigned long>>
+ getLastLog(): std::tuple<String, unsigned long>
+ getLogSize(): unsigned long
+ getLogsAsJSON(): String
+ getLogsFromString(logs_str: String): std::vector<String>
+ setLogs(logs: std::vector<std::tuple<String,unsigned long>>)
AudioManager
- instance: AudioManager*
- audio: Audio
- streaming: bool
- stream_url: String
+ getInstance(): AudioManager*
+ initialize(): bool
+ startStream(url: String): bool
+ stopStream(): bool
+ isStreaming(): bool
+ loop()
+ getStreamUrl(): String
+ getVolume(): int
+ setVolume(volume: int)

3 Anhang 3: Code

3.1 Anhang 3.1: Code Adapter

AudioManager.h

```
#ifndef AUDIOMANAGER_H
#define AUDIOMANAGER_H

#include <Arduino.h>
#include "constants.h"
#include "AudioFileSourceICYStream.h"
#include "AudioFileSourceBuffer.h"
#include "AudioGeneratorMP3.h"
#include "AudioOutputI2S.h"
#include "Logger.h"

class AudioManager{
    private:
        static AudioManager* instance;
        AudioGeneratorMP3 *gen;
        AudioFileSourceICYStream *src;
        AudioFileSourceBuffer *buff;
        AudioOutputI2S *out;
        bool streaming;
        String stream_url;
        int volume;

        AudioManager();
        ~AudioManager();

    public:
        static AudioManager* getInstance();

        /**
         * sets the url, from which the audio stream should be received
         */
        void setStreamUrl(String url);

        /**
         * starts to receive the audio stream from the given url
         * @param url URL of the audio stream, which should be received
         */
        void startStream();

        /**
```

```
         * stops the current audio stream
         */
        void stopStream();

        /**
        * returns the stream url
        */
        String getStreamUrl();

        /**
         * returns if the audio stream paused
         */
        bool isPaused();

        /**
         * handles the audio process
         */
        void loop();

        /**
         * sets the volume of the output
         * @param volume the desired volume, in the range between 0 an 100
         */
        void setVolume(int volume);

        /**
         * returns the volume, which is currently set
         */
        int getVolume();
};
#endif
```

AudioManger.cpp

```
#include "AudioManager.h"

AudioManager* AudioManager::instance = nullptr;

void MDCallback(void *cbData, const char *type, bool isUnicode, const char
   ↪  *string) //for debugging
{
  const char *ptr = reinterpret_cast<const char *>(cbData);
  (void) isUnicode;
  char s1[32], s2[64];
  strncpy_P(s1, type, sizeof(s1));
  s1[sizeof(s1)-1]=0;
  strncpy_P(s2, string, sizeof(s2));
  s2[sizeof(s2)-1]=0;
  Serial.printf("METADATA(%s) '%s' = '%s'\n", ptr, s1, s2);
  Serial.flush();
}

void StatusCallback(void *cbData, int code, const char *string) //for
   ↪ debugging
{
  const char *ptr = reinterpret_cast<const char *>(cbData);
  // Note that the string may be in PROGMEM, so copy it to RAM for printf
  char s1[64];
  strncpy_P(s1, string, sizeof(s1));
  s1[sizeof(s1)-1]=0;
  Serial.printf("STATUS(%s) '%d' = '%s'\n", ptr, code, s1);
  Serial.flush();
}

AudioManager::AudioManager(){
    stream_url = "";
    streaming = false;
    volume = 100;
    audioLogger = &Serial;
    src = new AudioFileSourceICYStream();
    src->RegisterMetadataCB(MDCallback, (void*)"ICY");
    buff = new AudioFileSourceBuffer(src, AUDIO_BUFFERSIZE);
    buff->RegisterStatusCB(StatusCallback, (void*)"buffer");
    out = new AudioOutputI2S();
    out->SetPinout(I2S_BCLK_PIN, I2S_LRC_PIN, I2S_DOUT_PIN);
    out->SetBitsPerSample(AUDIO_BITSPERSAMPLE);
    out->SetChannels(AUDIO_CHANNELS);
    out->SetRate(AUDIO_SAMPLERATE);
    gen = new AudioGeneratorMP3();
    gen->RegisterStatusCB(StatusCallback, (void*)"mp3");
}

AudioManager* AudioManager::getInstance(){
    if(instance == nullptr){
        instance = new AudioManager();
    }
    return instance;
}

void AudioManager::setStreamUrl(String url){
    this->stream_url = url;
```

```
}

void AudioManager::stopStream(){
    Logger::add("stopping audio stream");
    streaming = false;
    if(gen->isRunning()){
        gen->stop();
    }
    if(src->isOpen()){
        src->close();
    }
}

void AudioManager::startStream(){
    Logger::add("start streaming audio from " + stream_url);
    stopStream();
    src->open(stream_url.c_str());
    gen->begin(buff, out);
    streaming = true;
}

String AudioManager::getStreamUrl(){
    return stream_url;
}

bool AudioManager::isPaused(){
    return !streaming;
}

void AudioManager::loop(){
    gen->loop();
}

void AudioManager::setVolume(int volume){
    if(volume >= 0 && volume <= 100){
        this->volume = volume;
        float gain = (float)volume/(float)100;
        Logger::add("setting gain to " + String(gain));
        out->SetGain(gain);
    }
}

int AudioManager::getVolume(){
    return volume;
}
```

BatteryManager.cpp

```cpp
#include "BatteryManager.h"

BatteryManager* BatteryManager::instance = nullptr;

BatteryManager::BatteryManager(){}

BatteryManager::~BatteryManager(){}

BatteryManager* BatteryManager::getInstance(){
    if(instance == nullptr){
        instance = new BatteryManager();
    }
    return instance;
}

/**
 * initializes the needed pins
 */
void BatteryManager::initializePins(){
    // ...
}

/**
 * returns the charging status of the battery
 *
 * @return charging status of the battery, in percent (0 - 100), as a
    ↪ String
 */
int BatteryManager::getBatteryStatus(){
    return 100; //default
}
```

BatteryManager.h

```cpp
#ifndef BATTERYMANAGER_H
#define BATTERYMANAGER_H

#include "Arduino.h"
#include "constants.h"

/**
 * manages the loading and status of the battery
 */
```

```
class BatteryManager{
    private:
        static BatteryManager *instance;

        BatteryManager();
        ~BatteryManager();

    public:
        static BatteryManager* getInstance();
        void initializePins();
        int getBatteryStatus();
};
#endif
```

constants.h

```
/**
 * file with constants, which are needed in the code
 */

#ifndef CONSTANTS_H
#define CONSTANTS_H

#include "Arduino.h"

//pins
const int I2S_BCLK_PIN = 27;
const int I2S_LRC_PIN = 26;
const int I2S_DOUT_PIN = 25;
const int BUTTON_PIN = 12;
const int LED_RED = 15;
const int LED_GREEN = 2;
const int LED_BLUE = 4;

//network
const IPAddress AP_LOCAL_IP(192,168,0,1);
const IPAddress AP_GATEWAY_IP(192,168,0,1);
const IPAddress AP_SUBNET_IP(255,255,255,0);
//const String AP_SSID = "Microcontroller";
const int MAX_RECONNECTION_TRIES = 2;
const unsigned long MAX_CONNECTION_TIME = 5000;

//memory
const String MEMORY_NAMESPACE = "variables";

const String SSID_KEY = "ssid";
const String PASSWORD_KEY = "password";
const String URL_KEY = "wifi";
const String LOGS_KEY = "logs";
const String NAME_KEY = "name";
const String IP_KEY = "ip";

//audio
const int AUDIO_BUFFERSIZE = 32768;
const int AUDIO_BITSPERSAMPLE = 16;
const int AUDIO_SAMPLERATE = 44100;
const int AUDIO_CHANNELS = 2;

//button
const int BUTTON_CONFIG_DURATION = 3000; //time for which the button has
    ↪ to be pressed, that config mode is activated

//other constants
const unsigned long SERIAL_BAUDRATE = 9600;
const int BUTTON_PRESS_SLEEP_TIME = 2000;
const unsigned long WLAN_REQUEST_PERIOD = 10000;
const int AUDIO_VOLUME = 10; //0-21
const int SERVER_PORT = 8080;
//const String DEFAULT_NAME = "MSA";
const String TIME_URL = "http://worldtimeapi.org/api/ip";
const int DEFAULT_VOLUME = 10;
#endif
```

Logger.cpp

```
#include "Logger.h"

std::vector<std::tuple<String,unsigned long>> Logger::logs;

void Logger::add(String log_entry){
    Serial.println(log_entry); //for debug purposes
    int time = 0;
    Logger::logs.push_back(std::make_tuple(log_entry, time));
}

std::vector<std::tuple<String,unsigned long>> Logger::getLogs(){
    return logs;
}

String Logger::getLogsAsJSON(){
    JsonDocument doc;
    for(int i = 0; i < logs.size(); i++){
        doc[i]["log_entry"] = std::get<0>(logs.at(i));
        doc[i]["time"] = std::get<1>(logs.at(i));
    }
    String logs;
    serializeJson(doc, logs);
    return logs;
}

std::tuple<String, unsigned long> Logger::getLastLog(){
    int log_size = getLogSize();
    return logs.at(log_size);
}

unsigned long Logger::getLogSize(){
    return logs.size();
}

std::vector<String> Logger::getLogsFromString(String logs_str){
    //empty
}

void Logger::setLogs(std::vector<std::tuple<String,unsigned long>> logs){
    Logger::logs = logs;
}

void Logger::clearLogs(){
    Logger::logs.clear();
}
```

Logger.h

```
#ifndef LOGGER_H
#define LOGGER_H
#include <Arduino.h>
#include <vector>
#include <ArduinoJson.h>

class Logger{
    private:
        /**
         * vector, in which the logs are written as a String
         */
        static std::vector<std::tuple<String, unsigned long>> logs;

    public:
        /**
```

```
         * adds a log entry to the logs vector
         */
        static void add(String log_entry);

        /**
         * returns the vector of all logs
         */
        static std::vector<std::tuple<String,unsigned long>> getLogs();

        /**
         * returns the last log of the logs vector
         */
        static std::tuple<String, unsigned long> getLastLog();

        /**
         * returns the size of the logs vecotor, as an unsigned long
         */
        static unsigned long getLogSize();

        /**
         * returns the logs as a serialized json
         */
        static String getLogsAsJSON();

        /**
         * reconverts a string with logs, seperated with commas to a log
            ↪ vector
         */
        static std::vector<String> getLogsFromString(String logs_str);

        /**
         * sets log vector to the given log vector
         */
        static void setLogs(std::vector<std::tuple<String,unsigned long>>
           ↪ logs);

        /**
         * clears the vector
         */
        static void clearLogs();
};
#endif
```

main.cpp

```
//including libraries
```

```
#include "Arduino.h"
#include "constants.h"
#include "NetworkManager.h"
#include "StatusLED.h"
#include "MemoryManager.h"
#include "Logger.h"
#include "ServerManager.h"
#include "AudioManager.h"
#include "Mode.h"

Mode mode = NORMAL;
unsigned long actual_time = 0;
unsigned long last_wlan_request_time = 0;
int wlan_reconnect_tries = 0;
unsigned long last_log_size = 0;
String last_log = "";
String name;

unsigned long wlan_connection_start = 0;
int wlan_reconnection_tries = 0;

//for button:
unsigned long press_start = 0;
unsigned long press_end = 0;
bool last_state = 0;

NetworkManager* network;
StatusLED* statusLED;
MemoryManager* memory;
ServerManager* server;
AudioManager* audio;
BatteryManager* battery;

void setMode(Mode m);
void handleButton();
void activateStandby();

void setup(){
    //set serial baudrate
    Serial.begin(SERIAL_BAUDRATE);

    //initialize button pin and attach interrupt to button
    pinMode(BUTTON_PIN, INPUT_PULLDOWN);
    esp_sleep_enable_ext0_wakeup(GPIO_NUM_12, 1); //wakes the esp32 up
       ↪ from deep sleep, when gpio 12 (button pin) is HIGH

    //getting instances of singleton classes
    network = NetworkManager::getInstance();
    statusLED = StatusLED::getInstance();
    memory = MemoryManager::getInstance();
    server = ServerManager::getInstance();
    battery = BatteryManager::getInstance();
    audio = AudioManager::getInstance();

    //setting name
    //name = "MAA_" + network->getMac()

    //turn status led off at the beginning
    statusLED->setOff();

    //if WLAN-credentials are set, read them and try to connect to WLAN
    if(memory->isWlanSsidSet() && memory->isWlanPasswordSet()){
        Logger::add("wlan credentials set in memory");
        String wlan_ssid = memory->readWlanSsid();
        String wlan_password = memory->readWlanPassword();
        Logger::add("SSID: " + wlan_ssid);
        Logger::add("password: " + wlan_password);
        Logger::add("starting wlan client");
        network->startClient(wlan_ssid, wlan_password, name);
        wlan_connection_start = millis();
        while(!network->isConnectedToWlan() && mode != ERROR){
            Serial.print(".");
            delay(100);
            if((millis() - wlan_connection_start) >= MAX_CONNECTION_TIME){ //if the max connection time for the wifi is exceeded, activate error mode
                Logger::add("max wlan connection time exceeded");
                setMode(ERROR);
            }
        }
        if(network->isConnectedToWlan()){ //if connected to wlan, set mode to normal
            Logger::add("connected to wlan");
            setMode(NORMAL);
        }
    } else { //if wlan credentials are not set, set mode to error
        Logger::add("wlan credentials not set in memory");
        setMode(ERROR);
    }
}
```

```

void loop(){
    handleButton(); //check if button is pressed
    actual_time = millis(); //time since start in ms

    if(mode != ERROR){
        if(mode == NORMAL){
        //check if still connected to Wlan
        if((actual_time - last_wlan_request_time) >= WLAN_REQUEST_PERIOD){
            Serial.println("free heap: " + String(esp_get_free_heap_size()));
            if(!network->isConnectedToWlan()){ //if not connected to wlan, try to reconnect
                if(wlan_reconnection_tries <= MAX_RECONNECTION_TRIES){
                    network->reconnect();
                    wlan_reconnect_tries ++;
                    Logger::add("reconnecting to wlan");
                } else {
                    Logger::add("not connected to wlan");
                    setMode(ERROR);
                }
            } else {
                int rssi = network->getRssi();
                wlan_reconnect_tries = 0;
            }
            last_wlan_request_time = actual_time;
        }
        if(!audio->isPaused()){ //if audio routine is running, execute audio loop
            audio->loop();
        }
        } else { //mode is config
            if(!network->isApStarted()){ //if ap is not running, start ap
                Logger::add("starting ap");
                network->startAP(name);
            }
        }

        if(server->isRunning()){ //if server is running, it should handle clients
            server->handleClient();
        } else {
            Logger::add("starting web server");
            server->start();
            Logger::add("setting mDNS");
            if(!network->setmDns(name)){
                Logger::add("mDNS couldn't be set");
            }
        }
    }
}

/**
 * method for setting modes
 * @param m Mode which should be set
 */
void setMode(Mode m){
    if(m == NORMAL){
        Logger::add("setting mode to normal");
        mode = NORMAL;
        statusLED->setGreen();
    } else if(m == ERROR){
        Logger::add("setting mode to error");
        mode = ERROR;
        statusLED->setRed();
    } else if(m == CONFIG){
        Logger::add("setting mode to config");
        mode = CONFIG;
        statusLED->setBlue();
    }
}

/**
 * method for checkin if button is pressed
 */
void handleButton(){
    int state = digitalRead(BUTTON_PIN);
    if(state == 1 && last_state == 0){ //button has been pressed
        press_start = millis();
    } else if(state == 0 && last_state == 1){ //button has been released
        press_end = millis();
    }
    if(press_start > 0 && press_end > 0){
        if((press_end - press_start) >= 3000){
            setMode(CONFIG);
        } else {
            activateStandby();
        }
        press_start = 0;
        press_end = 0;
    }
    last_state = state;
}

/**
 * method for activating standby mode (deep sleep)
 */
void activateStandby(){
    Logger::add("enabling standby");
    statusLED->setOff();
    esp_deep_sleep_start();
}
```

MemoryManager.cpp

```
#include "MemoryManager.h"

MemoryManager* MemoryManager::instance = nullptr;

MemoryManager::MemoryManager(){}
MemoryManager::~MemoryManager(){}

MemoryManager* MemoryManager::getInstance(){
    if (!instance) {
        instance = new MemoryManager();
    }
    return instance;
}

bool MemoryManager::isWlanSsidSet(){
    preferences.begin(MEMORY_NAMESPACE.c_str());
    return preferences.isKey(SSID_KEY.c_str());
    preferences.end();
}

bool MemoryManager::isWlanPasswordSet(){
    preferences.begin(MEMORY_NAMESPACE.c_str());
    return preferences.isKey(PASSWORD_KEY.c_str());
    preferences.end();
}

bool MemoryManager::isStreamUrlSet(){
    preferences.begin(MEMORY_NAMESPACE.c_str());
    return preferences.isKey(URL_KEY.c_str());
    preferences.end();
}
```

```

bool MemoryManager::isNameSet(){
    preferences.begin(MEMORY_NAMESPACE.c_str());
    return preferences.isKey(NAME_KEY.c_str());
    preferences.end();
}

bool MemoryManager::areLogsSet(){
    preferences.begin(MEMORY_NAMESPACE.c_str());
    return preferences.isKey(LOGS_KEY.c_str());
    preferences.end();
}

String MemoryManager::readWlanSsid(){
    Logger::add("reading wlan ssid from memory");
    preferences.begin(MEMORY_NAMESPACE.c_str());
    String ssid = preferences.getString(SSID_KEY.c_str());
    preferences.end();
    return ssid;
}

String MemoryManager::readWlanPassword(){
    Logger::add("reading wlan password from memory");
    preferences.begin(MEMORY_NAMESPACE.c_str());
    String ssid = preferences.getString(PASSWORD_KEY.c_str());
    preferences.end();
    return ssid;
}


String MemoryManager::readStreamUrl(){
    Logger::add("reading stream url from memory");
    preferences.begin(MEMORY_NAMESPACE.c_str());
    String url = preferences.getString(URL_KEY.c_str());
    preferences.end();
    return url;
}

String MemoryManager::readLogs(){
    Logger::add("reading logs from memory");
    preferences.begin(MEMORY_NAMESPACE.c_str());
    String logs = preferences.getString(LOGS_KEY.c_str());
    preferences.end();
    return logs;
}
```

```

String MemoryManager::readName(){
    //Logger::add("reading name from memory");
    preferences.begin(MEMORY_NAMESPACE.c_str());
    String name = preferences.getString(NAME_KEY.c_str());
    preferences.end();
    return name;
}

String MemoryManager::readIp(){
    Logger::add("reading ip from memory");
    preferences.begin(MEMORY_NAMESPACE.c_str());
    String ip = preferences.getString(IP_KEY.c_str());
    preferences.end();
    return ip;
}

void MemoryManager::writeWlanSsid(String ssid){
    Logger::add("writing wlan ssid in memory");
    preferences.begin(MEMORY_NAMESPACE.c_str());
    preferences.putString(SSID_KEY.c_str(), ssid);
    preferences.end();
}

void MemoryManager::writeWlanPassword(String password){
    Logger::add("writing wlan password in memory");
    preferences.begin(MEMORY_NAMESPACE.c_str());
    preferences.putString(PASSWORD_KEY.c_str(), password);
    preferences.end();
}

void MemoryManager::writeStreamUrl(String url){
    Logger::add("writing stream url in memory");
    preferences.begin(MEMORY_NAMESPACE.c_str());
    preferences.putString(URL_KEY.c_str(), url);
    preferences.end();
}

void MemoryManager::writeLogs(String logs){
    Logger::add("writing logs in memory");
    preferences.begin(MEMORY_NAMESPACE.c_str());
    preferences.putString(LOGS_KEY.c_str(), logs);
    preferences.end();
}

```

```
void MemoryManager::writeName(String name){
    Logger::add("writing name in memory");
    preferences.begin(MEMORY_NAMESPACE.c_str());
    preferences.putString(NAME_KEY.c_str(), name);
    preferences.end();
}

void MemoryManager::writeIp(String ip){
    Logger::add("writing ip in memory");
    preferences.begin(MEMORY_NAMESPACE.c_str());
    preferences.putString(IP_KEY.c_str(), ip);
    preferences.end();
}

void MemoryManager::clear(){
    preferences.begin(MEMORY_NAMESPACE.c_str());
    preferences.clear();
    preferences.end();
}
```

MemoryManager.h

```
#ifndef MEMORYMANAGER_H
#define MEMORYMANAGER_H

#include <Arduino.h>
#include <Preferences.h>
#include <constants.h>
#include "Logger.h"

class MemoryManager{
    private:
        static MemoryManager* instance;
        MemoryManager();
        ~MemoryManager();
        MemoryManager(const MemoryManager&) = delete;
        MemoryManager& operator = (const MemoryManager&) = delete;
        Preferences preferences;

    public:
        static MemoryManager* getInstance();

        /**
         * returns, if the wlan ssid is set to the memory
         *
         * @return if WLAN-SSID is set to the memory
         */
        bool isWlanSsidSet();

        /**
         * returns, if the wlan password is set to the memory
         *
         * @return if WLAN-Password is set to the memory
         */
        bool isWlanPasswordSet();

        /**
         * returns, if the stream url is set to the memory
         *
         * @return if Stream-URL is set to the memory
         */
        bool isStreamUrlSet();

        /**
         * returns, if the name is set to the memory
         *
         * @return if name of the microcontroller is set to the memory
         */
        bool isNameSet();

        /**
         * returns, if the last logs are set to the memory
         *
         * @return if Last logs are set to the memory
         */
        bool areLogsSet();

        /**
         * returns, if the ip-address is set to the memory
         *
         * @return if ip address is set to the memory
         */
        bool isIpSet();

        /**
         * reads the wlan ssid from the memory
         * @return WLAN-SSID, as a String
         */
        String readWlanSsid();

        /**
```

```
         * reads the wlan password from the memory
         * @return WLAN-Password, as a String
         */
        String readWlanPassword();

        /**
         * reads the last stream url from the memory
         * @return last Stream-URL, as a String
         */
        String readStreamUrl();

        /**
         * reads the last logs from the memory
         * @return last Logs, as a String
         */
        String readLogs();

        /**
         * reads the name from the memory
         *
         * @return name of the microcontroller, as a String
         */
        String readName();

        /**
         * reads the ip address from the memory
         *
         * @return ip address of the microcontroller, as a String
         */
        String readIp();

        /**
         * writes the given ssid to the memory
         *
         * @param ssid WLAN-SSID which should be written to the memory
         */
        void writeWlanSsid(String ssid);

        /**
         * writes the given password to the memory
         *
         * @param password WLAN-Password which should be written to the
            ↪ memory
         */
        void writeWlanPassword(String password);

        /**
         * writes the given url to the memory
         *
         * @param url Stream-URL which should be written to the memory
         */
        void writeStreamUrl(String url);

        /**
         * writes the given logs to the memory
         *
         * @param logs Logs which should be written to the memory
         */
        void writeLogs(String logs);

        /**
         * writes the given name to the memory
         *
         * @param name Name of the microcontroller, as a String
         */
        void writeName(String name);

        /**
         * writes the given ip address to the memory
         *
         * @param ip IP Address of the microcontroller, as a String
         */
        void writeIp(String ip);

        /**
         * clears the memory
         */
        void clear();
};
#endif
```

Mode.h

```
enum Mode{
    NORMAL,
    ERROR,
    CONFIG
};
```

NetworkManager.cpp

```
//including header file
#include "NetworkManager.h"

NetworkManager* NetworkManager::instance = nullptr;


/**
 * constructor
 * declares the needed variables
 */
NetworkManager::NetworkManager(){
    ap_started = false;
    //Log::add("network manager class created");
}

NetworkManager* NetworkManager::getInstance(){
    if(instance == nullptr){
        instance = new NetworkManager();
    }
    return instance;
}

/**
 * returns the mac address of the esp32
 */
String NetworkManager::getMac(){
    return WiFi.macAddress();
}

/**
 * scans for available networks and returns the ssid and rssi (strength)
    ↪ of the found networks as a json
 */
String NetworkManager::getAvailableNetworks(){
    JsonDocument networks;
    if(WiFi.getMode() == WIFI_AP){
        int available_networks = WiFi.scanNetworks(false);
        for(int i = 0; i < available_networks; i++){
            networks[i]["ssid"] = WiFi.SSID(i);
            networks[i]["rssi"] = WiFi.RSSI(i);
        }
    }
    String result;
    serializeJson(networks, result);
    return result;
```

```
}

/**
 * starts an access point
 */
bool NetworkManager::startAP(String ssid){
    //Log::add("starting ap");
    if(WiFi.getMode() != WIFI_AP){
        WiFi.mode(WIFI_AP);
    }
    ap_started = true;
    return WiFi.softAPConfig(AP_LOCAL_IP, AP_GATEWAY_IP, AP_SUBNET_IP) && WiFi.softAP(ssid);
}

/**
 * starts esp32 wlan client which connects to the access point with the given credentials
 */
bool NetworkManager::startClient(String ssid, String password, String hostname){
    if(WiFi.getMode() == WIFI_AP){ //if wifi is in ap mode, ap mode will be disabled and station mode will be enabled
        WiFi.softAPdisconnect();
        WiFi.mode(WIFI_STA);
    }
    WiFi.disconnect();
    int n = WiFi.scanNetworks();
    for(int i = 0; i < n; i++){
        if(WiFi.SSID(i) == ssid){
            String bssid = WiFi.BSSIDstr(i);
            Logger::add("ap mac: " + bssid);
            WiFi.setHostname(hostname.c_str());
            WiFi.begin(WiFi.SSID(i), password, 0, WiFi.BSSID(i));
            return true;
        }
    }
    return false;
}

void NetworkManager::reconnect(){
    WiFi.reconnect();
}

bool NetworkManager::isApModeActive(){
    return WiFi.getMode() == WIFI_AP;
}

bool NetworkManager::isConnectedToWlan(){
    if(!isApModeActive()){
        return WiFi.status() == WL_CONNECTED;
    }
    return false;
}

bool NetworkManager::setmDns(String name){
    return MDNS.begin(name) && MDNS.addService("http", "tcp", 80);
}

bool NetworkManager::isApStarted(){
    return ap_started;
}

String NetworkManager::getUtcTime(){
    struct tm timeinfo;
    configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
    getLocalTime(&timeinfo);
    return "example";
}

int NetworkManager::getRssi(){
    if(this->isConnectedToWlan()){
        return WiFi.RSSI();
    }
    return 0;
}
```

NetworkManager.h

```
#ifndef NETWORKMANAGER_H
#define NETWORKMANAGER_H

//including needed libraries
#include "Arduino.h"
#include "WiFi.h"
#include "constants.h"
#include "ArduinoJson.h"
#include "MemoryManager.h"
#include "Logger.h"
#include "ESPmDNS.h"
#include "HTTPClient.h"
```

```
#include "time.h"

//using namespace std for String an vectors
using namespace std;

/**
 * responsible for network tasks, like:
 * providing an access point
 * acting as a WiFi client
 */
class NetworkManager{
    private:
        static NetworkManager* instance;
        NetworkManager();
        ~NetworkManager();
        NetworkManager(const NetworkManager*) = delete;
        NetworkManager& operator = (const NetworkManager&) = delete;
        bool ap_started;
        HTTPClient http;
        const char* ntpServer = "pool.ntp.org";
        const long gmtOffset_sec = 0;
        const int daylightOffset_sec = 3600;

    public:
        static NetworkManager* getInstance();

        /**
         * starts the access point
         *
         * @param ssid SSID of the access point, as a String
         * @return starting process successful, as a boolean
         */
        bool startAP(String ssid);

        /**
         * starts a wifi client
         *
         * @param ssid WLAN-SSID, as a String
         * @param password WLAN-Password, as a String
         * @return connection successful, as a bool
         */
        bool startClient(String ssid, String password, String hostname);

        /**
         * reconnects to the ap
         */
        void reconnect();

        /**
         * returns the MAC-Address of the ESP32, as a String
         *
         * @return Mac-Address of the ESP32, as a String
         */
        String getMac();

        /**
         * scans for available networks and returns the ssid and rssi (
             ↪ strength)
         * of the found networks as a JSON converted to a String
         *
         * @return all available networks, as a serialized json
         */
        String getAvailableNetworks();

        /**
         * returns if wifi module is in access point mode
         *
         * @return Acces Point Mode active, as a bool
         */
        bool isApModeActive();

        /**
         * returns if wifi client is connected to WLAN
         *
         * @return connected to WLAN, as a bool
         */
        bool isConnectedToWlan();

        /**
         * sets mDNS
         *
         * @param name Name of the domain
         */
        bool setmDns(String name);

        /**
         * returns if ap is started
         *
         * @return ap started, as a bool
         */
        bool isApStarted();

        /**
         * returns the current utc time, requested from a time server, as
            ↪ a String
         *
         * @return utc time, as a string
         */
        String getUtcTime();

        /**
         * returns the RSSI of the network currently connected
         */
        int getRssi();
};
#endif
```

ServerManager.cpp

```
#include "ServerManager.h"

ServerManager* ServerManager::instance = nullptr;

ServerManager::ServerManager(){
    network = NetworkManager::getInstance();
    battery = BatteryManager::getInstance();
    audio = AudioManager::getInstance();
    memory = MemoryManager::getInstance();
    running = false;
}

ServerManager::~ServerManager(){}

ServerManager* ServerManager::getInstance(){
    if (!instance) {
        instance = new ServerManager();
    }
    return instance;
}

String ServerManager::getInfo(){
    String name = memory->readName();
    String mac = network->getMac();
    int volume = audio->getVolume();
    int battery_status = battery->getBatteryStatus();
    String station_url = audio->getStreamUrl();
    JsonDocument doc;
    doc["name"] = name;
    doc["mac"] = mac;
    doc["volume"] = volume;
    doc["battery"] = battery_status;
    doc["stationUrl"] = station_url;
    String info;
    serializeJson(doc, info);
    return info;
}

void ServerManager::handle_get(){
    Logger::add("get request on route / received");
    server.send(200, "text/plain", "get request received");
}

void ServerManager::handle_getInfo(){
    //Logger::add("get request on route /getInfo received");
    String adapterInfo = getInfo();
    server.send(200, "application/json", adapterInfo);
}

void ServerManager::handle_getAvailableNetworks(){
    Logger::add("get request on route /getAvailableNetworks received");
    String availableNetworks = network->getAvailableNetworks();
    server.send(200, "application/json", availableNetworks);
}

void ServerManager::handle_getLogs(){
    Logger::add("get request on route /getLogs received");
    String logs = Logger::getLogsAsJSON();
    server.send(200, "application/json", logs);
}

void ServerManager::handle_setWifiCredentials(){
    Logger::add("post request on route /setWifiCredentials received");
    if(server.hasArg("ssid") && server.hasArg("password")){
        String ssid = server.arg("ssid");
        String password = server.arg("password");
        Logger::add("writing ssid: " + ssid + " to memory");
        memory->writeWlanSsid(ssid);
        Logger::add("writing password: " + password + " to memory");
        memory->writeWlanPassword(password);
        server.send(201);
        Logger::add("restarting esp");
        ESP.restart();
    } else {
        server.send(400);
    }
}

void ServerManager::handle_setStreamUrl(){
    Logger::add("put request on route /setStreamUrl received");
    if(server.hasArg("url")){
        String url = server.arg("url");
        audio->setStreamUrl(url);
        audio->startStream();
        server.send(200);
    } else {
        server.send(400);
    }
}

void ServerManager::handle_setName(){
    Logger::add("put request on route /setName received");
    if(server.hasArg("name")){
        String name = server.arg("name");
        Logger::add("setting new name: " + name + " to memory");
        memory->writeName(name);
        server.send(200);
        Logger::add("restarting esp");
        ESP.restart();
    } else {
        server.send(400);
    }
}

void ServerManager::handle_setVolume(){
    Logger::add("put request on route /setVolume received");
    if(server.hasArg("volume")){
        int volume = server.arg("volume").toInt();
        audio->setVolume(volume);
        server.send(200);
    } else {
        server.send(400);
    }
}

void ServerManager::handle_pauseStream(){
    Logger::add("put request on route /pauseStream received");
    audio->stopStream();
    server.send(200);
}

void ServerManager::handle_continueStream(){
    Logger::add("put request on route /continueStream received");
    audio->startStream();
    server.send(200);
}

void ServerManager::handle_notFound(){
    server.send(404, "not found!");
}

bool ServerManager::start(){
    server.begin(SERVER_PORT);
    server.on("/", HTTP_GET, bind(&ServerManager::handle_get, this));
    server.on("/getAvailableNetworks", HTTP_GET, bind(&ServerManager::
       ↪ handle_getAvailableNetworks, this));
    server.on("/getLogs", HTTP_GET, bind(&ServerManager::handle_getLogs,
       ↪ this));
    server.on("/getInfo", HTTP_GET, bind(&ServerManager::handle_getInfo,
       ↪ this));
    server.on("/setName", HTTP_PUT, bind(&ServerManager::handle_setName,
       ↪ this));
    server.on("/setStreamUrl", HTTP_PUT, bind(&ServerManager::
       ↪ handle_setStreamUrl, this));
    server.on("/setVolume", HTTP_PUT, bind(&ServerManager::
       ↪ handle_setVolume, this));
    server.on("/setWifiCredentials", HTTP_POST, bind(&ServerManager::
       ↪ handle_setWifiCredentials, this));
    server.on("/pauseStream", HTTP_POST, bind(&ServerManager::
       ↪ handle_pauseStream, this));
    server.on("/continueStream", HTTP_POST, bind(&ServerManager::
       ↪ handle_continueStream, this));
    server.onNotFound(bind(&ServerManager::handle_notFound, this));
    running = true;
    return true;
}

bool ServerManager::stop(){
    server.stop();
    running = false;
    return true;
}
```

```

void ServerManager::handleClient(){
    server.handleClient();
}

bool ServerManager::isRunning(){
    return running;
}
```

ServerManager.h

```
#ifndef ServerManager_H
#define ServerManager_H

#include "Arduino.h"
#include "WebServer.h"
#include "constants.h"
#include "NetworkManager.h"
#include "BatteryManager.h"
#include "ArduinoJson.h"
#include "AudioManager.h"
#include "MemoryManager.h"

class ServerManager{
    private:
        static ServerManager* instance;
        ServerManager();
        ~ServerManager();
        ServerManager(const ServerManager*) = delete;
        ServerManager& operator = (const ServerManager&) = delete;
        WebServer server;
        NetworkManager* network;
        BatteryManager* battery;
        AudioManager* audio;
        MemoryManager* memory;
        bool running;

        /**
         * creates a json, filed with info about the adapter
         *
         * @return info info about the adapter as a serialized json
         */
        String getInfo();

        /**
         * handles a get request to the standard / route
         */
        void handle_get();

         /**
         * handles a put request to the /getInfo route
         */
        void handle_getInfo();

        /**
         * handles a get request to the /getAvailableNetworks route
         */
        void handle_getAvailableNetworks();

        /**
         * handles a get request to the /getLogs route
         */
        void handle_getLogs();

        /**
         * handles a post request to the /setWifiCredentials route
         */
        void handle_setWifiCredentials();

        /**
         * handles a post request to the /setStreamUrl route
         */
        void handle_setStreamUrl();

        /**
         * handles a post request to the /setName route
         */
        void handle_setName();

        /**
         * handles a post request to the /setVolume route
         */
        void handle_setVolume();

        /**
         * handles a post request to the /pauseStream route
         */
        void handle_pauseStream();

        /**
         * handles a post request to the /continueStream route
         */
        void handle_continueStream();

        /**
         * handles a request to a undefined route
         */
        void handle_notFound();
    public:
        static ServerManager* getInstance();

        /**
         * starts the webserver
         * @return if the start process was successful
         */
        bool start();

        /**
         * stops the webserver
         * @return if the stop process was successful
         */
        bool stop();

        /**
         * handles the clients
         */
        void handleClient();

        /**
         * returns if the wifi credentials are received from the client
         * @return if webserver received WiFi-credentials from client
         */
        bool wlanCredentialsReceived();

        /**
         * handles if the stream url is received from the client
         * @return if webserver received Stream-URL from client
         */
        bool urlReceived();

        /**
         * handles if the name is received from the client
         * @return if webserver received name of microcontroller from client
         */
        bool nameReceived();

        /**
         * handles if the volume is received from the client
         * @return if webserver received volume for audio output from
            ↪ client
         */
        bool volumeReceived();

        /**
         * returns the ssid, which was received from the client
         * @return WLAN-SSID, which the webserver received from the client
            ↪ , as a String
         */
        String getReceivedSsid();

        /**
         * returns the password, which was received from the client
         * @return WLAN-Password, which the webserver received from the
            ↪ client, as a String
         */
        String getReceivedPassword();

        /**
         * returns the url, which was received from the client
         * @return Stream-URL, which the webserver received from the
            ↪ client, as a String
         */
        String getReceivedUrl();

         /**
         * returns the name, which was received from the client
         * @return name of the microcontroller, which the webserver
            ↪ received from the client, as a String
         */
        String getReceivedName();

        /**
         * returns the volume, which was received from the client
         * @return value of the volume which the webserver received from
            ↪ the client, as a int
         */
        int getReceivedVolume();


        /**
```

```
         * returns if the webserver is running
         * @return if webserver is running
         */
        bool isRunning();
};
#endif
```

StatusLED.cpp

```
#include "StatusLED.h"

StatusLED* StatusLED::instance = nullptr;

StatusLED::StatusLED(){
    //initializing led pins
    pinMode(LED_RED, OUTPUT);
    pinMode(LED_GREEN, OUTPUT);
    pinMode(LED_BLUE, OUTPUT);
}

StatusLED::~StatusLED(){
    //empty
}

StatusLED* StatusLED::getInstance(){
    if(instance == nullptr){
        instance = new StatusLED();
    }
    return instance;
}

/**
 * sets the color of the led to red
 */
void StatusLED::setRed(){
    digitalWrite(LED_RED, HIGH);
    digitalWrite(LED_GREEN, LOW);
    digitalWrite(LED_BLUE, LOW);
}

/**
 * sets the color of the led to green
 */
void StatusLED::setGreen(){
    digitalWrite(LED_RED, LOW);
    digitalWrite(LED_GREEN, HIGH);
    digitalWrite(LED_BLUE, LOW);
}

/**
 * sets the color of the led to blue
 */
void StatusLED::setBlue(){
    digitalWrite(LED_RED, LOW);
    digitalWrite(LED_GREEN, LOW);
    digitalWrite(LED_BLUE, HIGH);
}

/**
 * sets the led off (no light)
 */
void StatusLED::setOff(){
    digitalWrite(LED_RED, LOW);
    digitalWrite(LED_GREEN, LOW);
    digitalWrite(LED_BLUE, LOW);
}
```

StatusLED.h

```
#ifndef STATUSLED_H
#define STATUSLED_H

#include <Arduino.h>
#include <constants.h>

/**
 * manages the state of the connected RGB led
 */
class StatusLED{
    private:
        static StatusLED *instance;

        StatusLED();
        ~StatusLED();

    public:
        static StatusLED* getInstance();

        /**
         * sets the color of the led to red
         */
        void setRed();

        /**
         * sets the color of the led to green
         */
        void setGreen();

        /**
         * sets the color of the led to blue
         */
        void setBlue();

        /**
         * sets the led off (no light)
         */
        void setOff();
};
#endif
```

3.2 Anhang 3.2: Code Smartphone-App

api/AdapterAPI.tsx

```
import axios from "axios";
import Network from "../types/Network";
import AdapterData from "@/types/AdapterData";

export const AdapterAPI = {

    getUrlFromMac(mac: string): string{
        let withoutSeperator = mac.replace(":", "");
        let uniquePart = withoutSeperator.substring(6, withoutSeperator.
           ↪ length-1);
        let url = "http://msa_" + uniquePart + ".local:8080";
        return url;
    },
    /**
     * get information of adapter via http get-request
     * @param {string} mac - mac of adapter
     * @returns {Promise<AdapterData>} - Promise, with Data as AdapterData
        ↪  type (name: string, mac: string, volume: number, battery:
        ↪ number, stationUrl: string)
     */
    async getInfo(mac: string): Promise<AdapterData>{
        const url = this.getUrlFromMac(mac) + "/getInfo";
        try{
            const res = await axios.get(url, {timeout: 2500});
            return {name: res.data.name, mac: mac, volume: res.data.volume, battery: res.data.battery, streamUrl: res.data.stationUrl, connected: true};
        } catch(err) {
            throw err;
        }
    },
    async getInfoFromHost(hostName: string): Promise<AdapterData>{
        const url = hostName + "/getInfo";
        try{
            const res = await axios.get(url, {timeout: 2500});
            return JSON.parse(res.data);
        } catch(err) {
            throw err;
        }
    },
    async getAvailableNetworks(mac: string): Promise<Network[]>{
        const url = this.getUrlFromMac(mac) + "/getAvailableNetworks";
        try{
            const res = await axios.get(url);
            return JSON.parse(res.data);
        } catch(err) {
            throw err;
        }
    },
    async getPaused(mac: string): Promise<boolean>{
        const url = this.getUrlFromMac(mac) + "/getPaused";
        try{
            const res = await axios.get(url);
            return JSON.parse(res.data).paused;
        } catch(err) {
            throw err;
        }
    },
    async sendConfigData(mac: string, wifiSsid: string, wifiPassword: string, newAdapterName: string){
        const url = this.getUrlFromMac(mac) + "/setConfigData";
        const data = "ssid=" + wifiSsid + "&password=" + wifiPassword + "&name=" + newAdapterName;
        return axios.post(url, data);
    },
    async sendVolume(mac: string, volume: number){
        const url = this.getUrlFromMac(mac) + "/setVolume";
        const data = "volume=" + volume;
        try{
            return axios.put(url, data);
        } catch(err){
            throw err;
        }
    },
    async sendStreamUrl(mac: string, streamUrl: string){
        const url = this.getUrlFromMac(mac) + "/setStreamUrl";
        const data = "url=" + streamUrl;
        try{
            return axios.put(url, data);
        } catch(err){
            throw err;
        }
    },
    async sendPauseStream(mac: string){
        const url = this.getUrlFromMac(mac) + "/pauseStream";
        try{
            return axios.post(url);
        } catch(err){
            throw err;
        }
    },
    async sendContinueStream(mac: string){
        const url = this.getUrlFromMac(mac) + "/continueStream";
        try{
            return axios.post(url);
        } catch(err){
            throw err;
        }
    }
}
```

api/FirebaseAPI.tsx

```
import { initializeApp } from "firebase/app";
import { createUserWithEmailAndPassword, signInWithEmailAndPassword,
   ↪ initializeAuth, getReactNativePersistence, signOut,
   ↪ sendPasswordResetEmail, confirmPasswordReset } from "firebase/auth";
import { getFirestore, setDoc, doc, getDoc, onSnapshot } from "firebase/
   ↪ firestore";
import User from "../types/User";
import Station from "@/types/Station";
import AsyncStorage from "@react-native-async-storage/async-storage";

const firebaseConfig = {
  apiKey: "AIzaSyBYW16NMGumkvA27llE6VyTszrAR80UDbo",
  authDomain: "msa-app-dad57.firebaseapp.com",
  projectId: "msa-app-dad57",
  storageBucket: "msa-app-dad57.firebasestorage.app",
  messagingSenderId: "278556649604",
  appId: "1:278556649604:web:6eb08d9dc209d160ccbad1",
  measurementId: "G-WMPLDFTYY2"
};

type Adapter = {
    name: string,
    mac: string
}

const app = initializeApp(firebaseConfig);
const auth = initializeAuth(app, {persistence: getReactNativePersistence(AsyncStorage)});
const storage = getFirestore();

export const Authentication = {
    async logIn(email: string, password: string): Promise<User>{
        try{
            const res = await signInWithEmailAndPassword(auth, email, password);
            if(res.user.email === null){
                throw "email is null";
            }
            return {uid: res.user.uid, email: res.user.email};
        } catch(err){
            throw err;
        }
    },
    async register(email: string, password: string): Promise<void>{
        try{
            await createUserWithEmailAndPassword(auth, email, password);
            return
        } catch(err) {
            throw err;
        }
    },
    async logOut(){
        return signOut(auth);
    },
    onAuthChange(callback: (user: User | null) => void){
        auth.onAuthStateChanged((user) => {
            let newUser;
            if(user !== null && user.uid !== null && user.email !== null){
                newUser = {uid: user.uid, email: user.email};
            } else {
                newUser = null;
            }
            callback(newUser);
        });
    },
    onAuthReady(callback: () => void){
        auth.authStateReady().then(() => callback())
        .catch(err => {
            console.error(err);
        });
    },
    getUser(): User | null{
        const user = auth.currentUser;
        if(user !== null && user.email !== null){
            return {uid: user.uid, email: user.email};
        }
        return null;
    },
    async sendPwResetEmail(email: string){
        return sendPasswordResetEmail(auth, email);
    },
    async confirmPwReset(code: string, newPw: string){
        return confirmPasswordReset(auth, code, newPw);
    }
}

export const CloudStorage = {
    async getAdapterList(): Promise<Adapter[]>{
        if(auth.currentUser !== null){
            let uid = auth.currentUser.uid;
            try{
                const docName = "user_" + uid;
                const res = await getDoc(doc(storage, "adapter", docName))
                   ↪ ;
                const data = res.data();
                if(data === undefined || data.adapterList === undefined){
                    throw "data is undefined";
                }
                console.log("adapter data:", data.adapterList);
                return data.adapterList;
            } catch(err) {
                throw err;
            }
        } else {
            throw "user is null";
        }
    },
    async getStationList(): Promise<Station[]>{
        if(auth.currentUser !== null){
            let uid = auth.currentUser.uid;
            try{
                const docName = "user_" + uid;
                const res = await getDoc(doc(storage, "station", docName))
                    ↪ ;
                const data = res.data();
                if(data === undefined || data.stationList === undefined){
                    throw "data is undefined";
                }
                return data.stationList;
            } catch(err) {
                throw err;
            }
        } else {
            throw "user is null";
        }
    },
    async setAdapterList(newAdapterList: Adapter[]): Promise<void>{
        if(auth.currentUser !== null){
            let uid = auth.currentUser.uid;
            try{
                const docName = "user_" + uid;
                const data = {adapterList: newAdapterList};
                await setDoc(doc(storage, "adapter", docName), data);
                return
            } catch(err){
                throw err;
            }
        } else {
            throw "user is null";
        }
    },
    async setStationList(newStationList: Station[]): Promise<void>{
        if(auth.currentUser !== null){
            let uid = auth.currentUser.uid;
            try{
                const docName = "user_" + uid;
                const data = {stationList: newStationList};
                await setDoc(doc(storage, "station", docName), data);
                return
            } catch(err){
                throw err;
            }
        } else {
            throw "user is null";
        }
    },
    onAdapterChange(callback: (newAdapterList: Adapter[]) => void){
        if(auth.currentUser !== null){
            let uid = auth.currentUser.uid;
            const docName = "user_" + uid;
            const document = doc(storage, "adapter", docName);
            onSnapshot(document, (newDoc) => {
                const data = newDoc.data();
                let adapterList = [];
                if(data !== undefined){
                    adapterList = data.adapterList;
                }
                callback(adapterList);
            })
        } else {
            throw "user is null";
        }
    },
    onStationChange(callback: (newStationList: Station[]) => void){
        if(auth.currentUser !== null){
            let uid = auth.currentUser.uid;
            const docName = "user_" + uid;
            const document = doc(storage, "station", docName);
            onSnapshot(document, (newDoc) => {
                const data = newDoc.data();
                let stationList = [];
                if(data !== undefined){
                    stationList = data.stationList;
                }
                callback(stationList);
            })
        } else {
            throw "user is null";
        }
    }
}
```

api/RadioBrowserAPI.tsx

```
import axios from "axios";
import Station from "@/types/Station";
import Country from "../types/Country";
import Language from "../types/Language";

export const RadioBrowserAPI = {
    async getCountryNames(): Promise<Country []>{
        try{
            const res = await axios.get("https://de1.api.radio-browser.info/json/countries?order=stationcount&reverse=true&limit=50");
            const countries = res.data;
            const countryList: Country[] = [];
            for(let country of countries){
                countryList.push({name: country.name, code: country.iso_3166_1});
            }
            const sortedCountries = countryList.sort((a, b) => {
                if(a.name == b.name){
                    return 0;
                } else if(a.name > b.name) {
                    return 1;
                } else {
                    return -1;
                }
            });
            return sortedCountries;
        } catch(err) {
            throw err;
        }
    },
    async getLanguageNames(): Promise<Language []>{
        try{
            const res = await axios.get("https://de1.api.radio-browser.info/json/languages?order=stationcount&reverse=true&limit=20");
            const languages = res.data;
            const languageList: Language[] = [];
            for(let language of languages){
                let oldName = language.name;
                let newName = oldName.charAt(0).toUpperCase() + oldName.slice(1);
                if(oldName.includes(' ')){
                    let spaceIdx = newName.indexOf(' ');
                    newName = newName.slice(0, spaceIdx) + ' ' + newName.charAt(spaceIdx+1).toUpperCase() + newName.slice(spaceIdx+2);
                }
                languageList.push({name: newName, code: language.iso_639});
            }
            const sortedLanguages = languageList.sort((a, b) => {
                if(a.name == b.name){
                    return 0;
                } else if(a.name > b.name) {
                    return 1;
                } else {
                    return -1;
                }
            });
            return sortedLanguages;
        } catch(err) {
            throw err;
        }
    },
    async getStations(countryName: string, languageName: string, maxStations: number, dontShow: Station[] | null): Promise<Station[]>{
        let url = "http://de1.api.radio-browser.info/json/stations/search?order=clickcount&reverse=true&hidebroken=true&codec=mp3&limit=" + maxStations;
        if(languageName !== null && languageName !== "-"){
            url += "&language=" + languageName.toLowerCase();
        }
        if(countryName !== null && languageName !== "-"){
            url += "&country=" + countryName;
        }
        console.log(url);
        try{
            const stations = await axios.get(url);
            const result: Station[] = [];
            stations.data.forEach((val: any) => {
                if(dontShow !== null){
                    let containsUuid = false;
                    for(let favStation of dontShow){ //check if station is already in favourite stations
                        if(favStation.uuid == val.stationuuid){
                            containsUuid = true;
                        }
                    }
                    if(!containsUuid){
                        const station = {uuid: val.stationuuid, name: val.
                           ↪ name, iconUrl: val.favicon, url: val.url};
                        result.push(station);
                    }
                } else {
                    const station = {uuid: val.stationuuid, name: val.name
                       ↪ , iconUrl: val.favicon, url: val.url};
                    result.push(station);
                }
            })
            return result;
        } catch(err) {
            throw err;
        }
    },
    async getStationInfo(streamUrl: string){
        const url = "http://de1.api.radio-browser.info/json/stations/byurl
           ↪ ?url=" + streamUrl;
        try{
            const apiRes = await axios.get(url);
            return apiRes.data[0];
        } catch(err) {
            throw err;
        }
    }
}
```

app/index.tsx

```
import { useContext } from "react";
import { UserContext } from "../context/UserContext";
import { Redirect } from "expo-router";
import { StyleSheet, Text } from "react-native";
import LoadingScreen from "@/components/LoadingScreen";
import { SafeAreaView } from "react-native-safe-area-context";
import { GlobalStyle } from "@/constants/Style";

export default function Index(){
    const { user, available } = useContext(UserContext);

    const style = StyleSheet.create({
        container: {
            alignItems: 'center'
        }
    })

    if(available){
        return( user !== null
            ? <Redirect href={"/(tabs)/connection"}/>
            : <Redirect href={"/(auth)/login"}/>
        )
    } else {
        return(
            <SafeAreaView style={[GlobalStyle.page, style.container]}>
                <Text style={GlobalStyle.textBig}>Willkommen in der MSA
                    ↪ App!</Text>
                <LoadingScreen text="Lade Daten ..."/>
            </SafeAreaView>
        )
    }
}
```

app/_layout.tsx

```
import { Stack } from 'expo-router';
import { UserProvider } from '../context/UserContext';

export default function Layout() {
  return(
    <UserProvider>
      <Stack screenOptions={{
        headerShown: false
      }}>
        <Stack.Screen name='index'/>
      </Stack>
    </UserProvider>
  )
}
```

app/(auth)/login.tsx

```
import { useState } from "react";
import { Text, TextInput, Button, SafeAreaView, View, StyleSheet } from "
   ↪ react-native";
import { GlobalStyle, Colors } from "@/constants/Style";
import { router } from "expo-router";
import { Authentication } from "../../api/FirebaseAPI";
import { MemoryService } from "../../services/MemoryService";

```

```
export default function LoginScreen(){
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [errorText, setErrorText] = useState("");

  const style = StyleSheet.create({
    inputContainer: {
      alignItems: 'center'
    }, error: {
      color: Colors.red
    },
    container: {
      backgroundColor: Colors.grey,
      width: '80%',
      alignSelf: 'center',
      marginTop: 70,
      padding: 10,
      borderRadius: 20,
      alignItems: 'center'
    },
    input: {
      fontSize: 18,
      borderColor: Colors.lightGrey,
      borderRadius: 5,
      borderWidth: 2,
      width: 200,
      marginBottom: 20,
      marginTop: 5,
      color: Colors.white,
      textAlign: 'center',
    }
  })

  return(
    <SafeAreaView style={GlobalStyle.page}>
      <View style={style.container}>
        <View style={style.inputContainer}>
          <Text style={GlobalStyle.textBig}>E-Mail:</Text>
          <TextInput style={style.input} onChangeText={(text) => {setEmail
             ↪ (text)}}/>
          <Text style={GlobalStyle.textBig}>Passwort:</Text>
          <TextInput style={style.input} onChangeText={(text) => {
             ↪ setPassword(text)}} secureTextEntry/>
        </View>
        <Button color={Colors.lightTurquoise} title="Anmelden" onPress={()
```

```
           ↪  => {
          Authentication.logIn(email, password).then(res => {
            MemoryService.setUser({uid: res.uid, email: res.email});
            router.replace("/(tabs)/connection");
          }).catch(err => {
            setErrorText(err.message);
          })
        }}/>
        <Text style={[GlobalStyle.textMedium, style.error]}>{errorText}</
           ↪ Text>
        <Text style={GlobalStyle.textMedium}>Haben Sie noch kein Konto?</
           ↪ Text>
        <Button color={Colors.lightTurquoise} title="Registrieren" onPress
           ↪ ={() => {
          router.replace("/register");
        }}/>
      </View>
    </SafeAreaView>
  )
}
```

app/(auth)/register.tsx

```
import {  useState } from "react";
import { Text, TextInput, Button, SafeAreaView, View, StyleSheet } from "
   ↪ react-native";
import { GlobalStyle, Colors } from "@/constants/Style";
import { router } from "expo-router";
import { Authentication } from "../../api/FirebaseAPI";

export default function RegisterScreen(){
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [errorText, setErrorText] = useState("");

  const style = StyleSheet.create({
    inputContainer: {
      alignItems: 'center'
    }, error: {
      color: Colors.red
    },
    container: {
      backgroundColor: Colors.grey,
      width: '80%',
      alignSelf: 'center',
      marginTop: 70,
      padding: 10,
      borderRadius: 20,
      alignItems: 'center'
    },
    input: {
      fontSize: 18,
      borderColor: Colors.lightGrey,
      borderRadius: 5,
      borderWidth: 2,
      width: 200,
      marginBottom: 20,
      marginTop: 5,
      color: Colors.white,
      textAlign: 'center',
    }
  })

  return(
    <SafeAreaView style={GlobalStyle.page}>
      <View style={style.container}>
        <View style={style.inputContainer}>
          <Text style={GlobalStyle.textBig}>E-Mail:</Text>
          <TextInput style={style.input} onChangeText={(text) => {setEmail
             ↪ (text)}}/>
          <Text style={GlobalStyle.textBig}>Passwort:</Text>
          <TextInput style={style.input} onChangeText={(text) => {
             ↪ setPassword(text)}} secureTextEntry/>
        </View>
        <Button color={Colors.lightTurquoise} title="Registrieren" onPress
           ↪ ={() => {
          Authentication.register(email, password).then(() => {
            console.log("redirecting to login");
            router.replace("/login");
          }).catch(err => {
            setErrorText(err.message);
          })
        }}/>
        <Text style={[GlobalStyle.textMedium, style.error]}>{errorText}</
           ↪ Text>
        <Text style={GlobalStyle.textMedium}>Haben Sie bereits ein Konto
           ↪ ?</Text>
        <Button color={Colors.lightTurquoise} title="Anmelden" onPress={()
           ↪  => {
          router.replace("/login");
        }}/>
      </View>
    </SafeAreaView>
  )
}
```

app/(auth)/_layout.tsx

```
import { Stack } from 'expo-router';

export default function Layout() {
  return(
    <Stack>
        <Stack.Screen name='index'/>
        <Stack.Screen name='login'/>
        <Stack.Screen name='register'/>
    </Stack>
  )
}
```

app/(tabs)/_layout.tsx

```
import FontAwesome from "@expo/vector-icons/FontAwesome";
import MaterialIcons from "@expo/vector-icons/MaterialIcons";
import { Tabs } from "expo-router";
import { Colors } from "@/constants/Style";
import MaterialCommunityIcons from "@expo/vector-icons/
   ↪ MaterialCommunityIcons";
import { StationProvider } from "@/context/StationContext";
import { AdapterProvider } from "@/context/AdapterContext";
import { UserProvider } from "@/context/UserContext";

export default function TabLayout() {
  return (
    <UserProvider>
      <AdapterProvider>
        <StationProvider>
          <Tabs
            screenOptions={{
              tabBarActiveTintColor: Colors.darkTurquoise,
              tabBarStyle: { backgroundColor: Colors.grey },
              tabBarInactiveTintColor: Colors.white,
              headerShown: false,
            }}
          >
            <Tabs.Screen
              name="connection"
              options={{
                title: "Verbindungen",
                tabBarIcon: ({ color }) => (
                  <FontAwesome name="chain" size={28} color={color} />
                ),
              }}
            />
            <Tabs.Screen
              name="adapter"
              options={{
                title: "Adapter",
                tabBarIcon: ({ color }) => (
                  <MaterialIcons size={28} name="speaker-group" color={
                     ↪ color} />
                ),
              }}
            />
            <Tabs.Screen
              name="music"
              options={{
                title: "Musik",
                tabBarIcon: ({ color }) => (
                  <MaterialIcons size={28} name="library-music" color={
                     ↪ color} />
                ),
              }}
            />
            <Tabs.Screen
              name="profile"
              options={{
                title: "Profil",
                tabBarIcon: ({ color }) => (
                  <MaterialCommunityIcons
                    name="account-box"
                    size={28}
                    color={color}
                  />
                ),
              }}
            />
          </Tabs>
        </StationProvider>
      </AdapterProvider>
    </UserProvider>
  );
}
```

app/(tabs)/adapter/addAdapter.tsx

```
import { SafeAreaView, Button, StyleSheet } from "react-native";
import { GlobalStyle, Colors } from "@/constants/Style";
import { router } from "expo-router";

export default function AddAdapter(){
    const style = StyleSheet.create({
        container: {
            justifyContent: 'center'
        }
    })
    return(
        <SafeAreaView style={[GlobalStyle.page, style.container]}>
            <Button color={Colors.lightTurquoise} title="Neuen Adapter
              ↪ hinzufuegen" onPress={() => router.push("/(tabs)/adapter
              ↪ /addNewAdapter")}/>
            <Button color={Colors.lightTurquoise} title="Bestehenden
              ↪ Adapter hinzufuegen" onPress={() => router.push("/(tabs)
              ↪ /adapter/addExistingAdapter")}/>
        </SafeAreaView>
    )
}
```

app/(tabs)/adapter/addExistingAdapter.tsx

```
import { SafeAreaView, Text, TextInput, Button } from "react-native";
import { GlobalStyle } from "@/constants/Style";
import { useContext, useState } from "react";
import { AdapterAPI } from "@/api/AdapterAPI";
import { AdapterContext } from "@/context/AdapterContext";
import AdapterData from "@/types/AdapterData";
import { CloudStorage } from "@/api/FirebaseAPI";

export default function AddExistingAdapter(){
    const [mac, setMac] = useState("");
    const { adapterList } = useContext(AdapterContext);
    return(
        <SafeAreaView style={GlobalStyle.page}>
            <Text>Mac:</Text>
            <TextInput value={mac} onChangeText={(text) => setMac(text)}/>
            <Button title="Suche!" onPress={() => {
                AdapterAPI.getInfo(mac).then(res => {
                    let newAdapterList = [... adapterList];
                    let adapter: AdapterData = {name: res.name, mac,
                       ↪ volume: res.volume, battery: res.battery,
                       ↪ streamUrl: res.streamUrl, connected: true}
                    newAdapterList.push(adapter);
                    CloudStorage.setAdapterList(newAdapterList);
                }).catch(err => {
                    alert("Adapter kann nicht gefunden werden! Versuche es
                       ↪  erneut!")
                })
            }}/>
        </SafeAreaView>
    )
}
```

app/(tabs)/adapter/addNewAdapter.tsx

```
import { useEffect, useState } from "react";
import { Text, View, Button, SafeAreaView, TextInput } from "react-native
   ↪ ";
import { StyleSheet } from "react-native";
import ErrorScreen from "@/components/ErrorScreen";
import LoadingScreen from "@/components/LoadingScreen";
import TextInputWindow from "@/components/TextInputWindow";
import { AdapterAPI } from "@/api/AdapterAPI";
import { GlobalStyle, Colors } from "@/constants/Style";
import Network from "@/types/Network";
import NetworkList from "@/components/NetworkList";
import AdapterData from "@/types/AdapterData";

export default function AddNewAdapter(){
    const [isReachable, setReachable] = useState(false);
    const [loading, setLoading] = useState(true);
    const [adapter, setAdapter] = useState<AdapterData|null>(null);
    const [networkList, setNetworkList] = useState<Network[]|null>(null);
    const [selectedSsid, setSelectedSsid] = useState("");
    const [name, setName] = useState("");

    const host = "http://192.168.0.1:8080";

    useEffect(() => {
        setLoading(true);
        AdapterAPI.getInfoFromHost(host).then((res) => {
            setAdapter({name: res.name, mac: res.mac, battery: res.battery
               ↪ , volume: res.volume, connected: false, streamUrl: res.
               ↪ streamUrl});
            setLoading(false);
            setReachable(true);
            AdapterAPI.getAvailableNetworks(host).then(res => {
                setNetworkList(res);
            })
        }).catch(err => {
            setLoading(false);
            console.error(err);
            setReachable(false);
        })
    }, []);

    const style = StyleSheet.create({
        container: {
            alignSelf: 'center'
        },
        container2: {
            flexDirection: 'row'
        },
        icon: {
            marginLeft: 10
        },
        listContainer: {
            height: '30%',
        }
    })

    if(loading){
        return(
            <SafeAreaView style={GlobalStyle.page}>
                <LoadingScreen text="Versuche Adapter zu erreichen..."/>
            </SafeAreaView>
        )
    } else {
        if(isReachable && (adapter !== null) && (networkList !== null)){
            return(
                <SafeAreaView style={GlobalStyle.page}>
                    <View style={style.container2}>
                        <Text style={GlobalStyle.textBig}>{"Name: " +
                           ↪ adapter.name}</Text>
                        <TextInput value={adapter.name} onChangeText={(
                           ↪ text) => {setName(text)}}/>
                    </View>
                    <Text style={GlobalStyle.textBig}>{"Mac: " + adapter.
                       ↪ mac}</Text>
                    <Text style={GlobalStyle.textBig}>Mit WLAN verbinden
```

```
           ↪ :</Text>
          <View style={style.listContainer}>
            <NetworkList networks={networkList} onItemSelect
              ↪ ={(item: Network) => setSelectedSsid(item.
              ↪ ssid)}/>
          </View>
          <Button title="Adapter hinzufuegen" color={Colors.
           ↪ lightTurquoise}/>
          {selectedSsid.length > 0 &&
            <TextInputWindow text={"Passwort fuer " +
              ↪ selectedSsid + " eingeben:"} isPassword={
              ↪ true} onEnter={(password: string) => {alert(
              ↪ password)}} onCancel={() => {setSelectedSsid
              ↪ ("")}}/>
          }
        </SafeAreaView>
      )
    } else {
      return(
        <SafeAreaView style={GlobalStyle.page}>
          <ErrorScreen errorText="Adapter nicht erreichbar.
           ↪ Versichere dich, dass du mit dem WLAN des
           ↪ Adapters verbunden bist!" buttonText="Nochmal
           ↪ Versuchen" onButtonPress={() => {
            console.error("function not available!");
          }}/>
        </SafeAreaView>
      )
    }
  }
}
```

app/(tabs)/adapter/index.tsx

```
import { SafeAreaView } from "react-native";
import AdapterList from "@/components/AdapterList";
import { GlobalStyle } from "@/constants/Style";
import { useContext } from "react";
import { AdapterContext } from "@/context/AdapterContext";
import AdapterData from "@/types/AdapterData";
import { CloudStorage } from "@/api/FirebaseAPI";

export default function AdapterScreen(){
  const { adapterList } = useContext(AdapterContext);

  function deleteAdapter(selectedAdapter: AdapterData){
        let newAdapterList = [];
        for(let adapter of adapterList){
            if(!(selectedAdapter.mac == adapter.mac)){
                newAdapterList.push(adapter);
            }
        }
        CloudStorage.setAdapterList(newAdapterList);
    }

    return(
        <SafeAreaView style={GlobalStyle.page}>
            <AdapterList adapterList={adapterList} editable
               ↪ showOnlyAvailable={false} onItemSelect={() => {}}
               ↪ onDeleteAdapter={(adapter: AdapterData) => {
               ↪ deleteAdapter(adapter)}}/>
        </SafeAreaView>
    )
}
```

app/(tabs)/adapter/_layout.tsx

```
import { Stack } from 'expo-router';
import { Colors } from '@/constants/Style';

export default function Layout() {
  return (
    <Stack screenOptions={{
          headerStyle: {backgroundColor: Colors.grey},
          headerTitleStyle: {color: Colors.white}
          }}>
        <Stack.Screen name='index' options={{headerTitle: 'Adapter'}}/>
        <Stack.Screen name='addAdapter' options={{headerTitle: 'Adapter
           ↪ hinzufuegen'}}/>
        <Stack.Screen name='addNewAdapter' options={{headerTitle: 'Neuen
           ↪ Adapter hinzufuegen'}}/>
        <Stack.Screen name='addExistingAdapter' options={{headerTitle: '
           ↪ Bestehenden Adapter hinzufuegen'}}/>
    </Stack>
  );
}
```

app/(tabs)/connection/addConnection.tsx

```
import { Button, SafeAreaView, Text } from "react-native";
import { GlobalStyle, Colors } from "@/constants/Style";
import { useState, useEffect, useContext } from "react";
import AdapterList from "@/components/AdapterList";
import Station from "@/types/Station";
import { router } from "expo-router";
import StationList from "@/components/StationList";
import { AdapterAPI } from "@/api/AdapterAPI";
import AdapterData from "@/types/AdapterData";
import { AdapterContext } from "@/context/AdapterContext";

export default function AddConnection(){
    const [selectedAdapter, setSelectedAdapter] = useState<AdapterData|
       ↪ null>(null);
    const [selectedStation, setSelectedStation] = useState<Station|null>(
       ↪ null);
    const [buttonDisabled, setButtonDisabled] = useState(true);
    const { adapterList } = useContext(AdapterContext);

    useEffect(() => {
        if(selectedAdapter === null || selectedStation === null){
            setButtonDisabled(true);
        } else {
            setButtonDisabled(false);
        }
    }, [selectedAdapter, selectedStation]);

    return(
        <SafeAreaView style={GlobalStyle.page}>
            <Text style={GlobalStyle.textBig}>Adapter auswaehlen:</Text>
            <AdapterList adapterList={adapterList} editable={false}
               ↪ showOnlyAvailable onItemSelect={(item: AdapterData) => {
               ↪ setSelectedAdapter(item)}} onDeleteAdapter={()=>{}}/>
            <Text style={GlobalStyle.textBig}>Station auswaehlen:</Text>
            <StationList editable={false} onItemSelect={(item: Station) =>
               ↪  {setSelectedStation(item)}}/>
            <Button title="Bestaetigen" disabled={buttonDisabled} color={
               ↪ Colors.lightTurquoise} onPress={() => {
                if((selectedAdapter !== null) && (selectedStation !== null
                   ↪ )){
                    AdapterAPI.sendStreamUrl(selectedAdapter.name,
                       ↪ selectedStation.url).then(() => {
                        router.back();
                    })
                }
            }}/>
        </SafeAreaView>
    )
}
```

app/(tabs)/connection/index.tsx

```
import { SafeAreaView } from "react-native"
import ConnectionList from "@/components/ConnectionList";
import { GlobalStyle } from "@/constants/Style";
import { useContext } from "react";
import { AdapterContext } from "@/context/AdapterContext";
import { useState, useEffect } from "react";
import { RadioBrowserAPI } from "@/api/RadioBrowserAPI";
import Connection from "@/types/Connection";

export default function ConnectionScreen(){
    const [connectionList, setConnectionList] = useState<Connection[]>([])
       ↪ ;
    const { adapterList } = useContext(AdapterContext);

    useEffect(() => {
        for(let adapter of adapterList){
            if(adapter.connected && adapter.streamUrl.length > 0){
                let newConnectionList: Connection[] = [];
                let stationInfo = RadioBrowserAPI.getStationInfo(adapter.
                   ↪ streamUrl);
                let connection: Connection = {adapter: adapter, station: {
                   ↪ name: stationInfo.name, uuid: stationInfo.uuid, url:
                   ↪  stationInfo.url, iconUrl: stationInfo.favicon},
                   ↪ paused: true};
                newConnectionList.push(connection);
                setConnectionList(newConnectionList);
            }
        }
    }, [adapterList]);

    return (
        <SafeAreaView style={GlobalStyle.page}>
            <ConnectionList connectionList={connectionList} onItemPress
               ↪ ={()=>{}}/>
        </SafeAreaView>
    );
}
```

app/(tabs)/connection/_layout.tsx

```
import { Stack } from 'expo-router';
import { Colors } from '@/constants/Style';

```

```
export default function Layout() {
  return (
    <Stack screenOptions={{
              headerStyle: {backgroundColor: Colors.grey},
              headerTitleStyle: {color: Colors.white},
              headerBackTitle: "Zurueck"
              }}>
        <Stack.Screen name='index' options={{headerTitle: 'Verbindungen
           ↪ '}}/>
        <Stack.Screen name='addConnection' options={{headerTitle: '
           ↪ Verbindung hinzufuegen'}}/>
    </Stack>
  );
}
```

app/(tabs)/music/favouriteStationSelect.tsx

```
import { useEffect, useState, useContext } from "react";
import { FlatList, StyleSheet, Pressable, SafeAreaView } from "react-
   ↪ native";
import { Colors, GlobalStyle } from "@/constants/Style";
import Station from "@/types/Station";
import { router, useLocalSearchParams } from "expo-router";
import AntDesign from '@expo/vector-icons/AntDesign';
import StationItem from "@/components/StationItem";
import { RadioBrowserAPI } from "@/api/RadioBrowserAPI";
import { StationContext } from "@/context/StationContext";
import { CloudStorage } from "@/api/FirebaseAPI";

export default function Radios(){
    const [stations, setStations] = useState(Array());
    const [selectedStations, setSelectedStations] = useState(Array());
    const maxStations = 50;
    const {countryName, languageName} = useLocalSearchParams();
    const { stationList } = useContext(StationContext);

    function handleStationPress(station: Station){
        let newSelectedStations = [... selectedStations];
        if(newSelectedStations.includes(station)){
            const idx = newSelectedStations.indexOf(station);
            newSelectedStations.splice(idx, 1);
        } else {
            newSelectedStations.push(station);
        }
        let selectedNames: string[] = [];
        newSelectedStations.map((val) => {
            selectedNames.push(val.name);
        })
        console.log(selectedNames);
        setSelectedStations([... newSelectedStations]);
    }

    function isSelected(station: Station){
        let selectedUuids: string[] = [];
        selectedStations.forEach((val) => {
            selectedUuids.push(val.uuid);
        })
        let selected = selectedUuids.includes(station.uuid);
        return selected;
    }

    const style = StyleSheet.create({
        list: {
            height: '90%'
        },
        icon: {
            marginTop: 10,
            marginRight: 20,
            alignSelf: 'flex-end'
        }
    })

    useEffect(()=>{
        if(typeof countryName === "string" && typeof languageName === "string"){
            RadioBrowserAPI.getStations(countryName, languageName, maxStations, stationList).then(res =>{
                console.log(res);
                if(res != null){
                    setStations(res);
                }
            }).catch(err => {
                console.error(err);
            })
        }
    }, []);

    return(
        <SafeAreaView style={GlobalStyle.page}>
            <FlatList style={style.list} data={stations} renderItem={({item}) =>
```

```
                <Pressable onPress={() => handleStationPress(item)}>
                    <StationItem station={item} selected={isSelected(item)}/>
                </Pressable>
            }/>
            <Pressable onPress={() => {
                let newStationList;
                if(stationList != null){
                    newStationList = stationList.concat(selectedStations);
                } else {
                    newStationList = selectedStations;
                }
                console.log("new stationlist:", newStationList);
                try{
                    CloudStorage.setStationList(newStationList).then(() => {
                        router.replace("/music");
                    })
                }catch(err){
                    console.error(err);
                }
            }}>
                <AntDesign style={style.icon} name="check" size={50} color={Colors.lightTurquoise}/>
            </Pressable>
        </SafeAreaView>
    )
}
```

app/(tabs)/music/index.tsx

```
import { SafeAreaView } from "react-native";
import { GlobalStyle } from "@/constants/Style";
import StationList from "@/components/StationList";

export default function MusicScreen(){
    return(
        <SafeAreaView style={GlobalStyle.page}>
            <StationList editable onItemSelect={() => {}}/>
        </SafeAreaView>
    )
}
```

app/(tabs)/music/radiosearch.tsx

```
import { useEffect, useState } from "react";
import { ScrollView, Button } from "react-native";
import {Picker} from '@react-native-picker/picker';
import { Colors, GlobalStyle } from "@/constants/Style";
import { router } from "expo-router";
import { SafeAreaView } from "react-native-safe-area-context";
import { RadioBrowserAPI } from "@/api/RadioBrowserAPI";
import Language from "@/types/Language";
import Country from "@/types/Country";
import { SystemService } from "@/services/SystemService";

const Item = Picker.Item;

export default function RadioSearch(){
    const [selectedCountryName, setSelectedCountryName] = useState("");
    const [selectedLanguageName, setSelectedLanguageName] = useState("");
    const [countryDataset, setCountryDataset] = useState<Country[] | null
      ↪ >(null);
    const [languageDataset, setLanguageDataset] = useState<Language[] |
      ↪ null>(null);
    const [isDataFetched, setDataFetched] = useState(false);

    useEffect(()=>{
        RadioBrowserAPI.getCountryNames().then(res => {
            if(res != null){
                setCountryDataset(res);
            }
        }).catch(err => {
            console.error(err);
        });

        RadioBrowserAPI.getLanguageNames().then(res => {
            if(res != null){
                setLanguageDataset(res);
            }
        }).catch(err => {
            console.error(err);
        })
    },[]);

    useEffect(() => {
        const systemCountryCode = SystemService.getRegionCode();
        const systemLanguageCode = SystemService.getLanguageCode();
        if(countryDataset !== null){
            let systemCountry = countryDataset.find(country => country.
              ↪ code == systemCountryCode);
            if(systemCountry !== undefined){
                setSelectedCountryName(systemCountry.name);
            }
        }
        if(languageDataset !== null){
            let systemLanguage = languageDataset.find(language => language
               ↪ .code == systemLanguageCode);
            if(systemLanguage !== undefined){
                setSelectedLanguageName(systemLanguage.name);
            }
        }
    }, [countryDataset, languageDataset]);

    if(countryDataset !== null && languageDataset !== null){
        return(
            <SafeAreaView style={GlobalStyle.page}>
                <ScrollView>
                    <Picker onValueChange={(countryName: string) => {
                       ↪ setSelectedCountryName(countryName)}}
                       ↪ selectedValue={selectedCountryName}>
                        <Item key={"-"} value={"-"} label="-" color={
                           ↪ Colors.white}/>
                        {countryDataset.map((country, idx) =>(
                            <Item key={idx} value={country.name} label={
                               ↪ country.name} color={Colors.white}/>
                        ))}
                    </Picker>
                    <Picker onValueChange={(languageName: string) => {
                       ↪ setSelectedLanguageName(languageName)}}
                       ↪ selectedValue={selectedLanguageName}>
                        <Item key={"-"} value={"-"} label="-" color={
                           ↪ Colors.white}/>
                        {languageDataset.map((language, idx) =>(
                            <Item key={idx} value={language.name} label={
                               ↪ language.name} color={Colors.white}/>
                        ))}
                    </Picker>
                    <Button color={Colors.lightTurquoise} title="Search!"
                       ↪ onPress={() => {
                        router.push({pathname: "/(tabs)/music/
                           ↪ favouriteStationSelect", params: {
                           ↪ countryName: selectedCountryName,
                           ↪ languageName: selectedLanguageName}})
                    }}/>
                </ScrollView>
            </SafeAreaView>
        )
    }
}
```

app/(tabs)/music/_layout.tsx

```
import { Stack } from 'expo-router';
import { Colors } from '@/constants/Style';

export default function Layout() {
  return (
    <Stack screenOptions={{
              headerStyle: {backgroundColor: Colors.grey},
              headerTitleStyle: {color: Colors.white}
              }}>
        <Stack.Screen name='index' options={{headerTitle: 'Stationen'}}/>
        <Stack.Screen name='favouriteStationSelect' options={{headerTitle:
           ↪  'Stationen auswaehlen'}}/>
        <Stack.Screen name='radiosearch' options={{headerTitle: 'Stationen
           ↪  filtern'}}/>
    </Stack>
  );
}
```

app/(tabs)/profile/index.tsx

```
import { useContext } from "react";
import { Text, Button, SafeAreaView, StyleSheet } from "react-native";
import { GlobalStyle, Colors } from "@/constants/Style";
import { UserContext } from "@/context/UserContext";
import { Authentication } from "@/api/FirebaseAPI";
import { router } from "expo-router";

export default function ProfileScreen(){
  const { user } = useContext(UserContext);

  const style = StyleSheet.create({
    inputContainer: {
      alignItems: 'center'
    }, error: {
      color: Colors.red
    }
  })

  if(user !== null){
    return(
      <SafeAreaView style={GlobalStyle.page}>
```

```
        <Text>{"Email: " + user.email}</Text>
        <Button title="Abmelden" color={Colors.lightTurquoise} onPress={() => {
          Authentication.logOut().then(()=> router.replace("/"));
        }}/>
      </SafeAreaView>
    )
  }
}
```

app/(tabs)/profile/_layout.tsx

```
import { Stack } from 'expo-router';
import { Colors } from '@/constants/Style';

export default function Layout() {
  return (
    <Stack screenOptions={{
              headerStyle: {backgroundColor: Colors.grey},
              headerTitleStyle: {color: Colors.white}
              }}>
        <Stack.Screen name='index' options={{headerTitle: 'Profil'}}/>
    </Stack>
  );
}
```

components/AdapterItem.tsx

```
import { Text, View } from "react-native";
import Ionicons from '@expo/vector-icons/Ionicons';
import { StyleSheet } from "react-native";
import {Colors, GlobalStyle} from "@/constants/Style";
import Adapter from "../types/AdapterData";
import BatteryIndicator from "./BatteryIndicator";

type Props = {
  adapter: Adapter,
  selected: boolean,
  reachable: boolean
};

const style = StyleSheet.create({
    icon: {
        width: 50,
        height: 50,
    },
    container1: {
        flexDirection: 'row',
        justifyContent: 'space-between',
        alignItems: 'center',
        backgroundColor: Colors.white,
        borderColor: Colors.black,
        borderWidth: 1,
        borderRadius: 10,
        padding: 10,
        marginBottom: 7
    },
    container2: {
        flexDirection: 'row',
        justifyContent: 'space-between',
        alignContent: 'space-between',
        width: '20%'
    }
})
export default function AdapterItem({adapter, selected, reachable}: Props)
   ↪ {
  let backgroundColor = "lightgrey";
  if(selected){
    backgroundColor = Colors.lightTurquoise;
  } else if(reachable){
    backgroundColor = Colors.grey;
  } else {
    backgroundColor = "lightgrey";
  }

  return (
    <View style={[style.container1, {backgroundColor: backgroundColor}]}>
      <View>
        <Text style={GlobalStyle.textBig}>{adapter.name}</Text>
        <Text style={GlobalStyle.textMedium}>{adapter.mac}</Text>
      </View>
      {reachable
        ?
          <BatteryIndicator batteryPercentage={adapter.battery}/>
        : <Ionicons name="cloud-offline" size={24} color={Colors.white}/>
      }
    </View>
  );
}
```

components/AdapterList.tsx

```
import { useState } from "react";
import { View, FlatList, StyleSheet, Pressable } from "react-native";
import { router } from "expo-router";
import ErrorScreen from "@/components/ErrorScreen";
import DeleteButton from "./DeleteButton";
import AddToListButton from "./AddToListButton";
import AdapterItem from "./AdapterItem";
import { Alert } from "react-native";
import AdapterData from "@/types/AdapterData";

type Props = {
  adapterList: AdapterData[];
  onItemSelect: Function;
  onDeleteAdapter: Function;
  editable: boolean;
  showOnlyAvailable: boolean;
};

export default function AdapterList({
  adapterList,
  onItemSelect,
  onDeleteAdapter,
  editable,
  showOnlyAvailable,
}: Props) {
  const [selectedAdapter, setSelectedAdapter] = useState<AdapterData |
     ↪ null>(null);

  function handleItemPress(item: AdapterData) {
    if (selectedAdapter !== null && selectedAdapter.mac == item.mac) {
      setSelectedAdapter(null);
      onItemSelect(null);
    } else {
      setSelectedAdapter(item);
      onItemSelect(item);
    }
  }

  function handleDeletePress() {
    if (selectedAdapter !== null) {
      Alert.alert(
        "Adapter loeschen",
        "Wollen Sie den Adapter '" +
          selectedAdapter.name +
          "' wirklich loeschen?",
        [
        {
          text: "Nein",
          onPress: () => {
            setSelectedAdapter(null);
          },
        },
        {
          text: "Ja",
          onPress: () => {
            onDeleteAdapter(selectedAdapter);
          },
        },
      ]
    );
  }
}

function isSelected(item: AdapterData) {
  if (selectedAdapter !== null && selectedAdapter.mac == item.mac) {
    if ((showOnlyAvailable && item.connected) || !showOnlyAvailable) {
      return true;
    }
  }
  return false;
}

const style = StyleSheet.create({
  container: {
    width: "95%",
    alignSelf: "center",
  },
  icon: {
    alignSelf: "flex-start",
  },
  iconContainer: {
    flexDirection: "row",
    width: "95%",
    justifyContent: "space-between",
    alignSelf: "center",
  },
});

if (adapterList.length > 0) {
  return (
    <View style={style.container}>
```

```
        <FlatList
          data={adapterList}
          renderItem={({ item }) => (
            <Pressable
              onPress={() => {
                handleItemPress(item);
              }}
            >
              <AdapterItem adapter={item} selected={isSelected(item)}
                 ↪ reachable={item.connected}/>
            </Pressable>
          )}
        />
        {editable && (
          <View style={style.iconContainer}>
            <AddToListButton
              onPress={() => router.push("/(tabs)/adapter/addAdapter")}
            />
            {selectedAdapter !== null && (
              <DeleteButton
                onPress={() => {
                  handleDeletePress();
                }}
              />
            )}
          </View>
        )}
      </View>
    );
  } else {
    if(showOnlyAvailable) {
      return (
        <ErrorScreen
          errorText="Kein Adapter verfuegbar!"
          buttonText="Neuen Adapter hinzufuegen"
          onButtonPress={() => router.push("/(tabs)/adapter/addAdapter")}
        />
      );
    } else {
      return (
        <ErrorScreen
          errorText="Du hast noch keine Adapter hinzugefuegt!"
          buttonText="Adapter hinzufuegen"
          onButtonPress={() => router.push("/(tabs)/adapter/addAdapter")}
        />
      );
    }
  }
}
```

components/AddToListButton.tsx

```
import { Pressable } from "react-native";
import Entypo from "@expo/vector-icons/Entypo";
import { Colors } from "@/constants/Style";

type Props = {
    onPress: Function
}

export default function AddToListButton({onPress}: Props){
    return (
        <Pressable style={{alignSelf: 'flex-start'}} onPress={() =>
            ↪ onPress()}>
            <Entypo name="add-to-list" size={30} color={Colors.
                ↪ lightTurquoise} />
        </Pressable>
    )
}
```

components/BatteryIndicator.tsx

```
import { Text, View } from "react-native";
import FontAwesome from '@expo/vector-icons/FontAwesome';
import Ionicons from '@expo/vector-icons/Ionicons';
import { GlobalStyle, Colors } from "@/constants/Style";

type Props = {
    batteryPercentage: number
};

type IconNameType = "battery-empty" | "battery-full" | "battery-three-
    ↪ quarters" | "battery-half" | "battery-quarter";

export default function BatteryIndicator({batteryPercentage}: Props){
    let iconName: IconNameType;
    if(batteryPercentage > 75){
        iconName = "battery-full";
    } else if(batteryPercentage > 50){
        iconName = "battery-three-quarters";
    } else if(batteryPercentage > 25){
        iconName = "battery-half";
    } else if(batteryPercentage > 0){
        iconName = "battery-quarter";
    } else {
        iconName = "battery-empty";
    }

    if(batteryPercentage > 0){
        return(
            <View>
                <FontAwesome name={iconName} size={24} color={Colors.white
                    ↪ }/>
                <Text style={GlobalStyle.textMedium}>{batteryPercentage +
                    ↪ "%"}</Text>
            </View>
        )
    } else {
        return(
            <Ionicons name="battery-charging" size={24} color={Colors.
                ↪ white} />
        )
    }
}
```

components/ConnectionItem.tsx

```
import { View, Pressable } from "react-native";
import AntDesign from '@expo/vector-icons/AntDesign';
import { StyleSheet } from "react-native";
import {Colors} from "@/constants/Style";
import AdapterItem from "./AdapterItem";
import StationItem from "./StationItem";
import Connection from "../types/Connection";
import PlayPauseButton from "./PlayPauseButton";
import VolumeSelector from "./VolumeSelector";
import { AdapterAPI } from "@/api/AdapterAPI";

type Props = {
  connection: Connection
};

const style = StyleSheet.create({
    container: {
        flexDirection: 'column',
        justifyContent: 'space-between',
        width: '100%',
        backgroundColor: Colors.lightGrey,
        padding: 10,
        borderRadius: 20,
        marginBottom: 7
    },
    controlElementContainer: {
      alignItems: 'center'
    },
    xButton: {
      alignSelf: 'flex-end',
      paddingBottom: 15
    },
})

export default function ConnectionItem({connection}: Props) {
  function endConnection(){
    AdapterAPI.sendPauseStream(connection.adapter.mac).then(() => {
      AdapterAPI.sendStreamUrl(connection.adapter.mac, "");
    })
  }

  return (
    <View style={style.container}>
      <Pressable style={style.xButton} onPress={() => endConnection()}>
        <AntDesign name="disconnect" size={24} color={Colors.
           ↪ lightTurquoise} />
      </Pressable>
      <AdapterItem adapter={connection.adapter} selected={false} reachable
         ↪ ={true}/>
      <StationItem station={connection.station} selected={false}/>
      <View style={style.controlElementContainer}>
        <PlayPauseButton paused={connection.paused} onPress={() => {}}/>
        <VolumeSelector initVolumePercentage={connection.adapter.volume}
           ↪ onValueChange={(val: number) => {AdapterAPI.sendVolume(
           ↪ connection.adapter.name, val)}}/>
      </View>
    </View>
  );
}
```

components/ConnectionList.tsx

```
import { View, FlatList, StyleSheet, Pressable } from "react-native";
import { GlobalStyle } from "@/constants/Style";
import { router } from "expo-router";
import ErrorScreen from "@/components/ErrorScreen";
import { SafeAreaView } from "react-native-safe-area-context";
import AddToListButton from "./AddToListButton";
import ConnectionItem from "./ConnectionItem";
import Connection from "@/types/Connection";

type Props = {
  connectionList: Connection[];
  onItemPress: Function;
};

export default function ConnectionList({ connectionList, onItemPress }:
    ↪ Props) {
  const style = StyleSheet.create({
    container: {
      width: "95%",
      alignSelf: "center",
    },
    icon: {
      alignSelf: "flex-start",
    },
  });

  if (connectionList.length > 0) {
    return (
      <View style={style.container}>
        <FlatList
          data={connectionList}
          renderItem={({ item }) => (
            <Pressable onPress={() => onItemPress(item)}>
              <ConnectionItem
                connection={item}
              />
            </Pressable>
          )}
        />
        <AddToListButton
          onPress={() => router.push("/(tabs)/connection/addConnection")}
        />
      </View>
    );
  } else {
    return (
      <SafeAreaView style={GlobalStyle.page}>
        <ErrorScreen
          errorText="Es sind zurzeit keine Verbindungen vorhanden!"
          buttonText="Verbindung erstellen"
          onButtonPress={() => router.push("/(tabs)/connection/
             ↪ addConnection")}
        />
      </SafeAreaView>
    );
  }
}
```

components/DeleteButton.tsx

```
import { Pressable, StyleSheet } from "react-native"
import FontAwesome from "@expo/vector-icons/FontAwesome"
import { Colors } from "@/constants/Style"

type Props = {
    onPress: Function
}

export default function DeleteButton({onPress}: Props){
    return (
        <Pressable style={{alignSelf: 'flex-end'}} onPress={() => {onPress
           ↪ ()}}>
            <FontAwesome name="trash-o" size={30} color={Colors.red}/>
        </Pressable>
    )
}
```

components/ErrorScreen.tsx

```
import { Colors, GlobalStyle } from "@/constants/Style";
import { View, Text, Button, StyleSheet } from "react-native";

type Props = {
    errorText: string,
    buttonText: string,
    onButtonPress: Function
}

const style = StyleSheet.create({
    container:{
        height: '70%',
        justifyContent: 'center',
        alignItems: 'center',
    },
    text: {
        textAlign: 'center',
    }
})

export default function ErrorScreen({errorText, buttonText, onButtonPress
    ↪ }: Props){
    return(
        <View style={style.container}>
            <Text style={[GlobalStyle.textBig, style.text]}>{errorText}</
                ↪ Text>
            <Button color={Colors.lightTurquoise} title={buttonText}
                ↪ onPress={() => onButtonPress()}/>
        </View>
    )
}
```

components/LoadingScreen.tsx

```
import { GlobalStyle } from "@/constants/Style";
import { ActivityIndicator, Text, View, StyleSheet } from "react-native";
import { Colors } from "react-native/Libraries/NewAppScreen";

type Props = {
    text: string
}

const style = StyleSheet.create({
    container: {
        flex: 1,
        justifyContent: 'center',
        alignItems: 'center'
    }
})

export default function LoadingScreen({text}: Props){
    return (
        <View style={style.container}>
            <ActivityIndicator size="large" color={Colors.white}/>
            <Text style={GlobalStyle.textMedium}>{text}</Text>
        </View>
    )
}
```

components/NetworkItem.tsx

```
import { Text, StyleSheet, View } from "react-native";
import { Colors, GlobalStyle } from '@/constants/Style';
import MaterialIcons from '@expo/vector-icons/MaterialIcons';

```

```
type Props = {
  ssid: string,
  rssi: number,
  selected: boolean
};

const style = StyleSheet.create({
    container: {
        flexDirection: 'row',
        justifyContent: 'space-between',
        alignItems: 'center',
        borderWidth: 1,
        borderRadius: 10,
        padding: 10,
        marginBottom: 7,
    }
})

function getWifiItem(rssi :number){
  if(rssi > -50){
    return "network-wifi";
  } else if(rssi > -60){
    return "network-wifi-3-bar";
  } else if(rssi > -70){
    return "network-wifi-2-bar";
  } else {
    return "network-wifi-1-bar";
  }
}

export default function NetworkItem({ssid, rssi, selected}: Props) {
  return (
    <View style={[style.container, {backgroundColor: selected ? Colors.
      ↪ lightTurquoise : Colors.grey}]}>
        <Text style={GlobalStyle.textMedium}>{ssid}</Text>
        <MaterialIcons name={getWifiItem(rssi)} size={24} color={Colors.
          ↪ white}/>
    </View>
  );
}
```

components/NetworkList.tsx

```
import { useState } from "react";
import { View, FlatList, StyleSheet, Pressable } from "react-native";
import Network from "@/types/Network";
import NetworkItem from "./NetworkItem";

type Props = {
    networks: Network[],
    onItemSelect: Function
}

export default function NetworkList({networks, onItemSelect}: Props){
    const [selectedNetwork, setSelectedNetwork] = useState<Network|null>(
      ↪ null);

    const style = StyleSheet.create({
        container: {
            width: '95%',
            alignSelf: 'center'
        }
    })

    return(
        <View style={style.container}>
            <FlatList data={networks} renderItem={({item}) =>
                <Pressable onPress={() => {
                    setSelectedNetwork(item);
                    onItemSelect(item);
                }}>
                    <NetworkItem ssid={item.ssid} rssi={item.rssi}
                      ↪ selected={(selectedNetwork !== null) && (item.
                      ↪ ssid == selectedNetwork.ssid)}/>
                </Pressable>
            }/>
        </View>
    )
}
```

components/PlayPauseButton.tsx

```
import { Pressable } from "react-native";
import AntDesign from '@expo/vector-icons/AntDesign';
import { Colors } from "@/constants/Style";
import Entypo from '@expo/vector-icons/Entypo';

type Props = {
    paused: boolean,
    onPress: Function
}

```

```
export default function PlayPauseButton({paused, onPress}: Props){
    return(
        <Pressable onPress={() => onPress()}>
            { paused
                ? <Entypo name="controller-play" size={30} color={Colors.
                   ↪ lightTurquoise}/>
                : <AntDesign name="pause" size={30} color={Colors.
                   ↪ lightTurquoise}/>
            }
        </Pressable>
    )
}
```

components/StationItem.tsx

```
import { Text, View, Image, StyleSheet, Pressable } from "react-native";
import { Colors, GlobalStyle } from '@/constants/Style';
import Station from "../types/Station";

type Props = {
  station: Station,
  selected: boolean
};

const style = StyleSheet.create({
    icon: {
        width: 50,
        height: 50,
    },
    container: {
        flexDirection: 'row',
        justifyContent: 'space-between',
        alignItems: 'center',
        borderColor: Colors.black,
        borderWidth: 1,
        borderRadius: 10,
        padding: 5,
        marginBottom: 7
    }
})

export default function StationItem({station, selected}: Props) {
  return (
    <View style={[style.container, {backgroundColor: selected ? Colors.
       ↪ lightTurquoise : Colors.grey}]}>
        <Text style={GlobalStyle.textBig}>{station.name}</Text>
        <Image source={{uri: station.iconUrl}} style={style.icon}/>
    </View>
  );
}
```

components/StationList.tsx

```
import { useContext, useState } from "react";
import { View, FlatList, StyleSheet, Pressable } from "react-native";
import { GlobalStyle } from "@/constants/Style";
import StationItem from "@/components/StationItem";
import Station from "@/types/Station";
import { router } from "expo-router";
import ErrorScreen from "@/components/ErrorScreen";
import { SafeAreaView } from "react-native-safe-area-context";
import DeleteButton from "./DeleteButton";
import AddToListButton from "./AddToListButton";
import { Alert } from "react-native";
import { StationContext } from "@/context/StationContext";
import { CloudStorage } from "@/api/FirebaseAPI";

type Props = {
  onItemSelect: Function;
  editable: boolean;
};

export default function StationList({ onItemSelect, editable }: Props) {
  const { stationList } = useContext(StationContext);
  const [selectedStation, setSelectedStation] = useState<Station | null>(
     ↪ null);

  function handleItemPress(item: Station) {
    if (selectedStation !== null && selectedStation.uuid == item.uuid) {
      setSelectedStation(null);
      onItemSelect(null);
    } else {
      setSelectedStation(item);
      onItemSelect(item);
    }
  }

  function deleteItem() {
    if(selectedStation !== null && stationList.length > 0) {
        let newStationList = [... stationList];
        for(let i = 0; i < newStationList.length; i++){
            if(newStationList[i].uuid == selectedStation.uuid){
                newStationList.splice(i, 1);
                break;
            }
        }
        if(newStationList.length !== 0){
            CloudStorage.setStationList(newStationList);
        } else{
            CloudStorage.setStationList([]);
        }
    }
}

function handleDeletePress() {
    if (selectedStation !== null) {
        Alert.alert(
            "Station loeschen",
            "Wollen Sie die Station '" +
                selectedStation.name +
                "' wirklich loeschen?",
            [
                {
                    text: "Nein",
                    onPress: () => {
                        setSelectedStation(null);
                    },
                },
                {
                    text: "Ja",
                    onPress: () => {
                        deleteItem();
                    },
                },
            ]
        );
    }
}

const style = StyleSheet.create({
    container: {
        width: "95%",
        alignSelf: "center",
        marginTop: 20
    },
    icon: {
        alignSelf: "flex-start",
    },
    iconContainer: {
      flexDirection: "row",
      width: "95%",
      justifyContent: "space-between",
      alignSelf: "center",
    },
  });

  if(stationList.length > 0) {
    return (
      <View style={style.container}>
        <FlatList
          data={stationList}
          renderItem={({ item }) => (
            <Pressable
              onPress={() => {
                handleItemPress(item);
              }}
            >
              <StationItem
                station={item}
                selected={
                  selectedStation !== null && selectedStation.uuid == item
                     ↪ .uuid
                }
              />
            </Pressable>
          )}
        />
        {editable && (
          <View style={style.iconContainer}>
            <AddToListButton
              onPress={() =>
                router.push("/(tabs)/music/radiosearch", {
                  relativeToDirectory: true,
                })
              }
            />
            {selectedStation !== null && (
              <DeleteButton
                onPress={() => {
                  handleDeletePress();
                }}
              />
            )}
          </View>
        )}
      </View>
    );
  } else {
    return (
      <SafeAreaView style={GlobalStyle.page}>
        <ErrorScreen
          errorText="Du hast noch keine Stationen hinzugefuegt!"
          buttonText="Station hinzufuegen"
          onButtonPress={() => router.push("/(tabs)/music/radiosearch")}
        />
      </SafeAreaView>
    );
  }
  }
```

components/TextInputWindow.tsx

```
import { useState } from "react";
import { Text, Button, View, TextInput, StyleSheet } from "react-native";
import { Colors, GlobalStyle } from "@/constants/Style";

type Props = {
    text: string,
    isPassword: boolean,
    onEnter: Function,
    onCancel: Function
}

const style = StyleSheet.create({
    container:{
        backgroundColor: Colors.grey,
        position: 'absolute',
        zIndex: 2,
        padding: 20,
        alignSelf: 'center',
        borderRadius: 10,
        marginTop: 50
    },
    input: {
        borderColor: Colors.white,
        borderWidth: 0.2,
        marginTop: 20,
        color: Colors.white
    },
    container2: {
        flexDirection: 'row',
        marginTop: 20
    }
})

export default function TextInputWindow({text, isPassword, onEnter,
   ↪ onCancel}: Props){
    const [password, setPassword] = useState("");
    return(
        <View style={style.container}>
            <Text style={GlobalStyle.textMedium}>{text}</Text>
            <TextInput style={style.input} value={password} onChangeText
               ↪ ={(text) => setPassword(text)} secureTextEntry={
               ↪ isPassword}/>
            <View style={style.container2}>
                <Button color={Colors.lightTurquoise} title="Abbrechen"
                   ↪ onPress={() => {onCancel()}}/>
                <Button color={Colors.lightTurquoise} title="Bestaetigen"
                   ↪ onPress={() => {onEnter(password)}}/>
            </View>
        </View>
    )
}
```

components/VolumeSelector.tsx

```
import { Text, View, Button} from "react-native";
import { GlobalStyle, Colors } from "@/constants/Style";
import Slider from "@react-native-community/slider";
import { useState } from "react";
import { StyleSheet } from "react-native";

type Props = {
    initVolumePercentage: number,
    onValueChange: Function
};

const style = StyleSheet.create({
    container: {
        alignItems: 'center'
    },
    innerContainer: {
        flexDirection: 'row'
    }
})

export default function VolumeSelector({initVolumePercentage,
   ↪ onValueChange}: Props){
    const [volume, setVolume] = useState(initVolumePercentage);
    return(
        <View style={style.container}>
            <View style={style.innerContainer}>
                <Button title="-" color={Colors.lightTurquoise}
                    onPress={() => {if(volume > 0) {
                                        setVolume(volume-1)
                                        onValueChange(volume)
                                    }}}
                />
                <Slider
                    minimumValue={0}
                    maximumValue={100}
                    step={1}
                    value={volume}
                    onSlidingComplete={(val) => onValueChange(volume)}
                    onValueChange={(val) => {setVolume(val)}}
                    vertical={true}
                    thumbTintColor={Colors.white}
                    style={{width: '50%'}}
                    minimumTrackTintColor={Colors.lightTurquoise}
                    maximumTrackTintColor={Colors.lightTurquoise}
                />
                <Button title="+" color={Colors.lightTurquoise}
                    onPress={() => {if(volume < 100) {
                                        setVolume(volume+1)
                                        onValueChange(volume)
                                    }}}
                />
            </View>
            <Text style={GlobalStyle.textBig}>{volume + "%"}</Text>
        </View>
    )
}
```

components/WifiItem.tsx

```
import { Text, View, StyleSheet } from "react-native";
import {Colors} from "@/constants/Style";

type Props = {
  ssid: string,
  rssi: number,
  selected: boolean
};

const style = StyleSheet.create(
  {
    icon: {
      width: 50,
      height: 50,
  },
  container: {
      flexDirection: 'row',
      justifyContent: 'space-between',
      alignItems: 'center',
      backgroundColor: Colors.white,
      borderColor: Colors.black,
      borderWidth: 1,
      borderRadius: 10,
      padding: 10,
      marginBottom: 7
  }
}
);

export default function WifiItem({ssid, rssi, selected}: Props) {
  return (
    <View style={[style.container, {backgroundColor: selected ? Colors.
      ↪ lightTurquoise : Colors.white}]}>
      <Text>{ssid}</Text>
      <Text>{rssi}</Text>
    </View>
  );
}
```

components/navigation/TabBarIcon.tsx

```
// You can explore the built-in icon families and icons on the web at
   ↪ https://icons.expo.fyi/

import Ionicons from '@expo/vector-icons/Ionicons';
import { type IconProps } from '@expo/vector-icons/build/createIconSet';
import { type ComponentProps } from 'react';

export function TabBarIcon({ style, ...rest }: IconProps<ComponentProps<
   ↪ typeof Ionicons>['name']>) {
  return <Ionicons size={28} style={[{ marginBottom: -3 }, style]} {...
```

```
    ↪ rest} />;
}
```

components/__tests__/ThemedText-test.tsx

```
import * as React from 'react';
import renderer from 'react-test-renderer';

import { ThemedText } from '../ThemedText';

it(`renders correctly`, () => {
  const tree = renderer.create(<ThemedText>Snapshot test!</ThemedText>).
    ↪ toJSON();

  expect(tree).toMatchSnapshot();
});
```

components/__tests__/__snapshots__/ThemedText-test.tsx.snap

```
// Jest Snapshot v1, https://goo.gl/fbAQLP

exports[`renders correctly 1`] = `
<Text
  style={
    [
      {
        "color": "#11181C",
      },
      {
        "fontSize": 16,
        "lineHeight": 24,
      },
      undefined,
      undefined,
      undefined,
      undefined,
      undefined,
    ]
  }
>
  Snapshot test!
</Text>
`;
```

context/AdapterContext.tsx

```
import {
  createContext,
  useContext,
  useState,
  useEffect,
  ReactNode,
} from "react";
import { CloudStorage } from "../api/FirebaseAPI";
import { AdapterAPI } from "@/api/AdapterAPI";
import { UserContext } from "./UserContext";
import AdapterData from "@/types/AdapterData";

type Props = {
  children: ReactNode;
};

type AdapterContextType = {
  adapterList: AdapterData[];
};

const defaultContext: AdapterContextType = {
  adapterList: [],
};

export const AdapterContext = createContext<AdapterContextType>(
    ↪ defaultContext);

export const AdapterProvider = ({ children }: Props) => {
  const { user } = useContext(UserContext);
  const [adapterList, setAdapterList] = useState<AdapterData[]>(
    defaultContext.adapterList
  );

  function requestAdapters() {
    if (adapterList !== null) {
      let newAdapterList: AdapterData[] = [];
      let promiseList = [];
      for(let adapter of adapterList) {
        let promise = AdapterAPI.getInfo(adapter.mac);
        promiseList.push(promise);
      }
      Promise.allSettled(promiseList).then((results) => {
        for (let result of results) {
          if (result.status == "fulfilled") {
            let val = result.value;
            let newAdapter = {
              name: val.name,
              mac: val.mac,
              battery: val.battery,
              volume: val.volume,
              streamUrl: val.streamUrl,
              connected: true,
            };
            newAdapterList.push(newAdapter);
          }
        }
      });
      for(let adapter of adapterList) {
        let containsMac = false;
        for(let newAdapter of newAdapterList){
          if(newAdapter.mac == adapter.mac){
            containsMac = true;
            break;
          }
        }
        if(!containsMac){
          let newAdapter = {
            name: adapter.name,
            mac: adapter.mac,
            battery: 0,
            volume: 0,
            streamUrl: "",
            connected: false,
          };
          newAdapterList.push(newAdapter);
        }
      }
      setAdapterList(newAdapterList);
    }
  }

  useEffect(() => {
    let intervalId = 0;
    if (user !== null) {
      CloudStorage.onAdapterChange((newAdapterList) => {
        for(let newAdapter of newAdapterList){
          let containsMac = false;
          for(let adapter of adapterList){
            if(adapter.mac == newAdapter.mac){
              containsMac = true;
              break;
            }
          }
          if(!containsMac){
            let newAdapters = [... adapterList];
            let adapter: AdapterData = {name: newAdapter.name, mac:
              ↪ newAdapter.mac, volume: 0, battery: 0, streamUrl: "",
              ↪ connected: false};
            newAdapters.push(adapter);
            setAdapterList(newAdapters);
          }
          requestAdapters();
        }
        intervalId = setInterval(() => requestAdapters(), 5000);
      });
    } else {
      console.log("user is null");
    }
    return () => clearInterval(intervalId);
  }, [user]);

  return (
    <AdapterContext.Provider value={{ adapterList }}>
      {children}
    </AdapterContext.Provider>
  );
  };
```

context/StationContext.tsx

```
import { createContext, useContext, useState, useEffect, ReactNode } from
   ↪ "react";
import { CloudStorage } from "../api/FirebaseAPI";
import Station from "@/types/Station";
import { UserContext } from "./UserContext";

type Props = {
    children: ReactNode;
};

type StationContextType = {
    stationList: Station[]
};

const defaultContext = {
    stationList: []
};
```

```

export const StationContext = createContext<StationContextType>(
    ↪ defaultContext);

export const StationProvider = ({children}: Props) => {
    const { user } = useContext(UserContext);
    const [stationList, setStationList] = useState<Station[]>(
        ↪ defaultContext.stationList);

    useEffect(() => {
        if(user !== null){
            CloudStorage.onStationChange((newStationList: Station[]) => {
                setStationList(newStationList);
            })
        }
    }, [user]);

    return(
        <StationContext.Provider value={{stationList}}>
            {children}
        </StationContext.Provider>
    )
}
```

context/SystemDataContext.tsx

```
import { createContext, useState, useEffect, ReactNode } from "react";
import { SystemService } from "../services/SystemService";

type Props = {
    children: ReactNode;
};

type SystemDataContextType = {
    connectedToInternet: boolean
};

const defaultContext = {
   connectedToInternet: false
};

export const SystemDataContext = createContext<SystemDataContextType>(
    ↪ defaultContext);

export const SystemDataProvider = ({children}: Props) => {
    const [connectedToInternet, setConnectedToInternet] = useState(false);

    useEffect(() => {
        SystemService.isConnectedToInternet().then(res => {
            setConnectedToInternet(res);
        }).catch(err => {
            console.error(err);
        })
    }, []);

    return(
        <SystemDataContext.Provider value={{connectedToInternet}}>
            {children}
        </SystemDataContext.Provider>
    )
}
```

context/UserContext.tsx

```
import { createContext, useState, useEffect, ReactNode } from "react";
import { Authentication } from "../api/FirebaseAPI";
import User from "../types/User";

type Props = {
    children: ReactNode;
};

type UserContextType = {
    user: User | null,
    available: boolean
};

const defaultContext = {
    user: null,
    available: false
};

export const UserContext = createContext<UserContextType>(defaultContext);

export const UserProvider = ({children}: Props) => {
    const [user, setUser] = useState<User | null>(defaultContext.user);
    const [available, setAvailable] = useState(defaultContext.available);

    useEffect(() => {
        Authentication.onAuthChange((newUser) => {
            console.log("auth changed");
            console.log("user:", newUser);
```

```
                setUser(newUser);
                setAvailable(true);
            })
        }, []);

        return(
            <UserContext.Provider value={{user, available}}>
                {children}
            </UserContext.Provider>
        )
}
```

types/AdapterData.ts

```
type AdapterData = {
    name: string,
    mac: string
    volume: number;
    battery: number;
    streamUrl: string;
    connected: boolean;
}

export default AdapterData;
```

types/Connection.ts

```
import Station from "./Station";
import AdapterData from "./AdapterData";

type Connection = {
    adapter: AdapterData;
    station: Station;
    paused: boolean;
}

export default Connection;
```

types/Country.ts

```
type Country = {
    name: string,
    code: string
}

export default Country;
```

types/Language.ts

```typescript
type Language = {
    name: string,
    code: string
}

export default Language;
```

types/Network.ts

```typescript
type Network = {
    ssid: string,
    rssi: number
}

export default Network;
```

types/Station.ts

```typescript
type Station = {
    uuid: string;
    name: string;
    iconUrl: string;
    url: string;
}

export default Station;
```

types/User.ts

```typescript
type User = {
    uid: string,
    email: string
}

export default User;
```

types/UserData.ts

```typescript
import Station from "./Station"
import AdapterData from "./AdapterData";

type UserData = {
    adapterList: AdapterData[] | null,
    stationList: Station[] | null
}

export default UserData;
```