Faculty of Science and Technology
Department of Computer Science

Data management platform for citizen science education projects

—

Nina Angelvik

INF-3990 Master's Thesis in Computer Science, May 2018

“It always seems impossible until it’s done.”
–Nelson Mandela

Abstract

The air:bit project is a computer science education project that we developed in 2016 for use in North-Norwegian upper secondary (videregående) schools. Students build and code their own air quality sensor kits (air:bits), before collecting air quality data in their local areas. They create their own air quality related research questions, which they answer by analyzing the collected air:bit data in context of other air quality data sources. The task of managing such datasets is too complex for such an introductory project and requires a specialized service.

This thesis describes the air:bit platform, a scalable and cloud-based data management platform for citizen science education projects. It provides students with a web application for storing, exploring, visualizing and downloading air quality data from air:bits and other data sources. It processes, stores and manages the air:bit data in the Google Cloud Platform, that provides an elastic scaling of storage and computational resources, and in addition simplifies managing the backend.

In 2018 the air:bit platform was used by 174 students from 11 school classes across Northern Norway. The students successfully built and programmed 62 air:bits and they uploaded 222 air:bit data log files to the air:bit platform, comprising 481,186 air quality measurements. We demonstrate the air:bit platform's ability to scale with regards to computational resources and that the latency of data uploads and queries is good enough for this purpose. We also provide the cost of processing and storing air:bit data in the cloud. The air:bit platform uses, on average, 30 seconds to verify and insert data log files for one week of per-minute measurements, and less than 2 seconds to process and retrieve the same data for visualization. The total cost of the air:bit platform is $211 per month or $1.20 per student.

We believe the air:bit platform is useful not only for the air:bit project, but also for other education projects. It can be used by air quality related citizen science projects and the open database provides street-level air quality data that can be used in other analyses. The air:bit platform is online at airbit.uit.no and is open-sourced at github.com/fjukstad/luft and github.com/ninaangelvik/luft.

Acknowledgements

First I would like to thank my advisor, Associate Professor Lars Ailo Bongo, and my co-advisor Bjørn Fjukstad for their continous motivation and guidance during the course of this project and the work related to the SIGCSE 2018 SRC poster submission.

Further I would like to thank the air:bit team: Bjørn Fjukstad, Julianne Iversen, Margaret Dalseng, Fredrik Rasch, Morten Grønnesby, Hedinn Gunhildrud, Maria Wulff Hauglann, Ken-Arne Jensen, Kai-Even Nilssen, Jon Ivar Kristiansen and Pontus Aurdal. And the collaborators: Sonja Grossberndt, Thomas Olsen, Juan Carlos Aviles Solis, Liv Ragnhild Hov and Geir Sagelv. For creating such an exciting project, for sharing your ideas and your knowledge, and for letting me be a part of it all. And to the teachers and students of the participating school classes: Thank you for bringing the project to life.

Then I would like to thank the Department of Computer Science and the School Laboratory for funding my poster presentation at SIGCSE2018. I am truly grateful.

To my fellow classmates: The past five years have been hard, but they would have been impossible without you. Thank you.

To my guys in A123: Einar Holsbø, Morten Grønnesby and Bjørn Fjukstad. Thank you for your kindness and encouragement, and for the warm welcome to your office.

I would like to thank my family, for always encouraging me to do my best.

Finally, to my girlfriend Hanne Bragmo: Thank you for your love and support. You are the raisin to my oatmeal.

Nina, May 2018.

Contents

List of Figures

List of Tables

List of Abbreviations

PM2.5	Particulate matter 2.5 micrometers or less in diameter
PM10	Particulate matter 10 micrometers or less in diameter
NO2	Nitrogen dioxide
VOC	Volatile organic compounds
CSV	Comma-separated values
NILU	Norwegian Institute for Air Research
MET	Meteorological Institute of Norway
API	Application programming interface
HTTP	Hypertext Transfer Protocol
REST	Representational state transfer
GCP	Google Cloud Platform
vCPU	Virtual CPU

/1 Introduction

Over the last decade, European societies and economics have experienced a significant digital and technological innovation. The technological advancement has created completely new job types and according to the European Commission's "White Paper on the Future" [1] it is likely that many children entering primary school today will end up working in jobs that do not yet exist. In the "Recommendation on Key competences for Lifelong Learning", published in 2018 [2], the authors highlight the need for adapting the current education and training systems in order to cope with the complexity and change in our societies. Whereas it used to be enough to equip young people with only a fixed set of skills or knowledge, they now need to develop resilience, a wide range of competencies such as creativity, logical thinking and problem solving, and the ability to adapt to change.

Creativity, logical thinking and problem solving can all be fostered through coding. Coding skills will also help to better understand today's digitalised society, therefore the benefits of integrating coding into the school curricula are many. In 2015, European Schoolnet published a report providing an overview of the coding initiatives and plans in both formal and informal learning initiatives across Europe [3].The findings were based on a survey with 21 Ministries of Education, in 20 European countries and Israel. The report revealed that 16 countries (including England, Estonia and France) were already integrating coding in the curriculum at either a national, regional or local level. Finland and Belgium Flanders had concrete plans to integrate it in the curriculum, while Belgium Wallonia, the Netherlands and **Norway** had neither integrated coding into the school curriculum or had any current plans to do so.

In 2016, we developed the air:bit project [4] in collaboration with the School Laboratory at the Faculty of Science and Technology at UiT - the Arctic University of Norway[1], to counteract the lack of computer science education in Norwegian upper secondary (videregående) schools. The project aims to introduce engineering and computer programming by combining computer science and natural sciences, and is conducted in collaboration with the Norwegian Institute for Air Research (NILU)[2] and the Meteorological Institute of Norway (MET)[3]. It is a maker-inspired citizen science approach focused on air pollution data collecting and monitoring, where students work in groups of 2-3 to build and code a portable air quality sensor kit (air:bit), which they use to collect air pollution data in their local environment.

An important part of the education project is for the students to develop their own air quality related research questions, such as “at what time during the day do our local kindergartens experience the highest level of air pollution?”. They will answer the questions by analyzing their own and their fellow students’ air quality data, air quality data from NILU, and in the future, other sources such as MET. However, the task of curating such datasets is too complex for such an introductory project and it therefore requires a specialized service to collect, store and integrate data, as well as query the the collected and integrated data.

This service must satisfy the following requirements:

Usability It must be simple and intuitive to use, such that the students and teachers should not need special training to understand how to use the service.

Scalability It must scale with regards to data storage and data processing, as the air:bit project expands to include more schools, students and air:bits.

Maintainability It should be easy to maintain since the air:bit project does not have a dedicated staff for IT operations.

Performance It should provide fast data uploads and fast data retrievals.

We here present our solution: the air:platform - a scalable, cloud-based data management platform for citizen science education projects.

1. uit.no/skolelab
2. api.nilu.no/docs/
3. api.met.no

1.1 Problems with existing data platforms

Citizen science commonly refers to projects that engage the public to generate and process research data [5]. There are several citizen science platforms available on the Internet. While most are designed for a single project or projects within the same, specific research field [6], some are large-scale and generic, allowing smaller citizen science projects within all fields to create their own data collection tools both with and without additional software development [7]. However, these do not meet all needs for citizen science education projects such as air:bit.

hackAIR[4] and luftdaten.info[5] are two projects that, like air:bit, aim to create awareness on air pollution by enabling the public to collect outdoor air quality data using self-built air quality sensors. Both projects have developed their own data platforms that receive data from their respective sensor kits over WiFi, where the public can explore the uploaded air quality data. Neither platforms are suitable alternatives for the air:bit project, since their sensor kits are too complicated for the students to code themselves and they depend on WiFi coverage to collect data. The two platforms also do not provide interfaces or APIs for querying and downloading data.

ResearchKit[6] and is an open-sourced framework by Apple that enables researchers to create mobile applications that use the sensors and capabilities of iPhone (and Android when using the ResearchDroid[7] library). The framework provides customizable templates for surveys and active tasks (tasks where participants perform activities while the smartphone sensors are actively collecting data). The ResearchKit framework is unsuitable for the air:bit project for two reasons: i) a smartphone does not have all of the necessary sensors for this project, such as a dust particle sensor; and ii) we cannot require the students to possess a smartphone in order to participate in the project.

The Plume API provides access to Plume Labs' air quality platform, allowing third-party companies access their data for a fee (some organizations, such as universities, can apply for free access). The API does, however, not allow users to upload data, and therefore we cannot use their data platform to store data from the air:bits.

4. hackair.eu/about-hackair/
5. luftdaten.info/en/home-en/
6. researchkit.org/
7. blog.appliedinformaticsinc.com/researchdroid-an-android-forms-and-consent-library/

1.2 Our solution: the air:bit platform

The air:bit platform is a service that collects, stores and queries air quality data, and visualizes the results. Unlike other air quality data platforms, it accepts air quality data from any air quality sensor kit as long as the data is formatted correctly (described in more detail in chapter 2.5) and uploaded as a csv file. The air:bit platform fulfils the platform requirements as follows:

Usability The air:bit web application provides the students and teachers with educational resources and simple interfaces to upload air:bit data log files, and explore and download their collected data through interactive visualizations. The visualizations also integrate the student uploaded data with external sources.

Scalability The computational parts of the air:bit platform are deployed to the Google Cloud Platform, which provides automatic scaling of storage and computational resources.

Maintainability The computational part of the air:bit platform are deployed to the App Engine Platform as a Service (PaaS) in the Google Cloud Platform, which manages infrastructure and maintenance.

Performance The air:bit platform performs timely uploads, but we identified needs for optimization with regards to the data query latency.

1.3 Use of air:bit platform in high schools

We launched the air:bit project in the fall 2017 and invited all high schools in Northern Norway where the students specialize in STEM subjects to participate. In spring 2018, 174 students from 11 North-Norwegian upper secondary (videregående) schools successfully built and programmed 62 air:bits. From February through April the students uploaded 222 data log files, comprising 481,186 air:bit measurements. Figures 1.1 and 1.2 show screenshots of the web application visualizing air quality measurements collected in Bodø in March 2018.

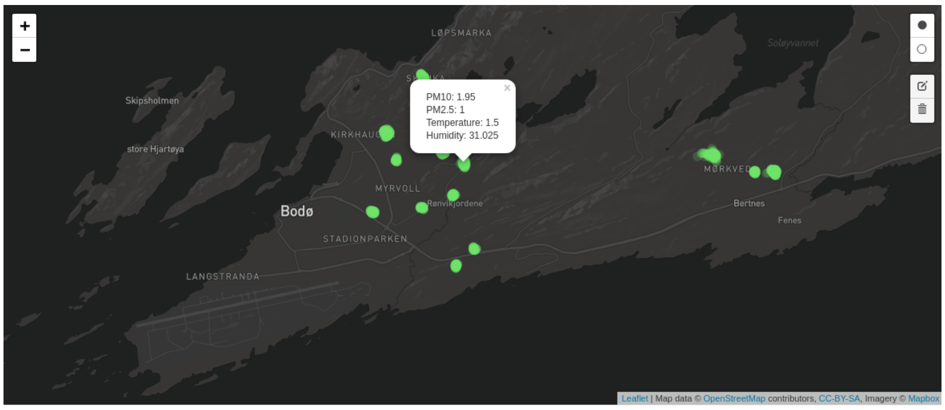


Figure 1.1: Map with air:bit measurements collected in Bodø in March 2018.

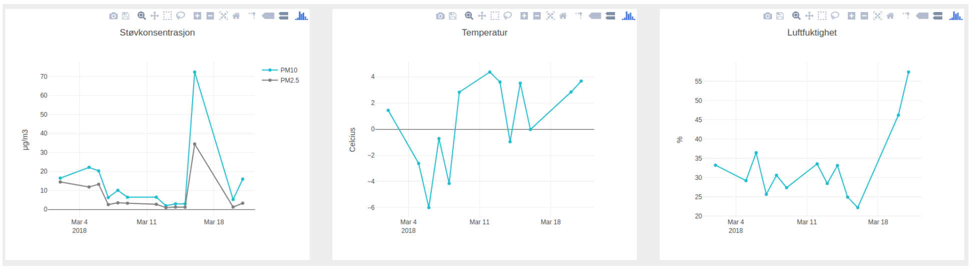


Figure 1.2: Charts visualizing the levels of PM10 and PM2.5 (left), temperature (middle) and humidity (right) in March 2018 based on collected air:bit data.

1.4 Contributions

The contributions of this work are

- Design, implementation and deployment of a cloud-based citizen science data management platform.

- Evaluation of a cloud-based citizen science data management platform with more than 174 monthly active users.

- Demonstration of a cloud-based citizen science data management platform that enables inexperienced programmers to store, visualize and download air quality sets.

1.5 Outline

This thesis is structured as follows. Chapter 2 provides the information necessary to understand the scope of this thesis, including an overview of the pilot air:bit projected conducted in spring 2017 and an introduction to air pollution. In chapter 3 we describe the complete air:bit platform architecture. The design and implementation is described in chapter 4. In chapter 5 we describe the project methodology. The evaluation of the air:bit platform is in chapter 6 with regards to the backend's resource usage, latency, scalability and cost. Related work is in chapter 7. Concluding remarks are given in chapter 8 and in chapter 9 we present our future work, pinpointing areas of improvement for the air:bit platform.

In February 2018 I participated in the Student Research Competition at SIGCSE 2018, the 49th ACM Technical Symposium on Computer Science and presented a poster on the air:bit data management platform. I also co-authored a paper on the air:bit project. The poster and paper are respectively included in Appendix A and B. We have open-sourced the air:platform codebase on github and included the repository urls in Appendix C.

The air:bit platform is online at airbit.uit.no. The source code for the air:bit frontend service is available at github.com/fjukstad/luft and the code for the backend data management system at github.com/ninaangelvik/luft.

2 Background

In spring 2017 we piloted the air:bit project with a class of 28 students from Kongsbakken VGS in Tromsø, for which we developed and evaluated a prototype air:bit, a prototype backend storage system and a pilot frontend service for exploring and visualizing collected air quality measurements. Although the pilot was successful, it also identified areas for improvement. The experiences and results from the pilot were used to develop the current air:bit and the air:bit platform. In this chapter we therefore describe i) the two prototypes, pinpointing what worked and motivating the needed improvements before making the project operational in the spring 2018; ii) the pilot frontend service which formed the basis for the air:bit frontend service and the air:bit web application at airbit.uit.no; iii) the current version of the air:bit and the air:bit educational resources published at airbit.uit.no. But first we give a brief introduction to air pollution, including why we want to monitor air quality and what we are monitoring.

2.1 Air pollution

Air pollution affects our health, our environment and our climate [8]. The WHO has termed it the largest single environmental health risk in the world [9], and both short and long term exposure to poor air quality as a result of air pollution contribute to respiratory disease, cardiovascular disease and certain cancers [8, 10, 11, 12, 13].

Air pollution originates from a wide range of sources, including exhaust from combustion engines burning fossil fuel, chemical emissions from factories and micro-particles from cars driving with studded tires on snow-free roads [8, 14]. The different sources generate different pollutants, e.g. Particulate Matter (PM), nitrogen oxides (NO_x) and Ozone (O_3). We focus on measuring PM since it is the major cause of poor air quality in Norway and there are simple and affordable sensors available to measure dust densities in the air. PM is often divided into two categories: PM10 and PM2.5, the numbers indicating the size of the dust particles. PM2.5 describes dust particles smaller than 2.5 µm and PM10 describes dust particles smaller than 10 µm. PM10 thereby also include all particles in PM2.5. We care about measuring dust particles smaller than 10 µm, as these can get deep into our lungs, possibly even our bloodstream, affecting both our lungs and heart [15].

In Northern Norway the air quality is heavily affected by rapid changes in weather in the winter months, especially while the seasons are changing. This is mainly due to the use of studded tires on dry roads, generating dust particles (PM) in the air, but also emissions from diesel powered cars are contributing to worsening the air quality [8]. Reducing the use of cars will improve air quality in these months and we believe that it can be done by creating awareness on the local conditions and their impact on health, rather than by enforcing it. European and Norwegian legislation ensure that the air quality is monitored and that the air quality forecasts are made public to the citizens (typically provided by the Norwegian Institute of Air Research (NILU) and the Meteorological Institute of Norway (MET)). However, since the monitoring is typically done using advanced stationary equipment that is too expensive to locate throughout all populated areas, most citizens in Northern Norway will not find available data in their city or neighborhood. This makes it harder, if not impossible, to raise awareness around poor air quality and simple measures to improve it.

2.2 The prototype air:bit

Figure 2.1 shows the prototype air:bit used in the pilot project[1]. Its total cost was 41 USD and it consisted of an Arduino Uno, a NEO6M GPS module, a Sharp GP2Y1010AU0F optical dust sensor, a DHT11 sensor measuring humidity and temperature, and microSD card reader/writer. The prototype created a single data log file on the memory card, appending data from all available sensors as rows to the data log file, where each line (row) in the file equaled one observation. The sensor data must be written to the data log file in a

1. Design and implementation by Bjørn Fjukstad, Hedinn Gunhildrud and Morten Grønnesby.

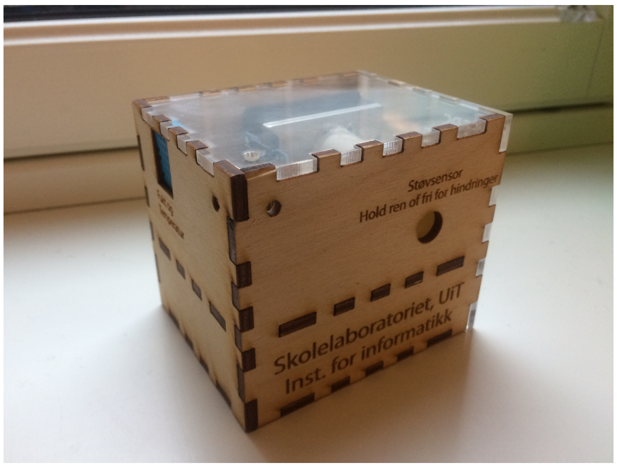


Figure 2.1: The prototype air:bit.

specific order for the prototype backend to insert the data correnctly into the database. The students programmed the prototype using the Arduino IDE, writing code in the C++ like Arduino programming language shown by listing 2.1. During the pilot project, we experienced that the DHT11 sensor was not able to register temperatures below 0°C. We also discovered that the Sharp GP2Y1010AU0F could only measure the total amount of dust particles in the air, not differentiating between PM10 and PM2.5, preventing us from directly comparing the data to official air quality data. Since measuring PM is an essential part of the project, we wanted to differentiate between PM10 and PM2.5. Norwegian winters also, more often than not, involve temperatures below 0°C. We therefore needed to replace the two sensors in the next version of the air:bit.

```
void setup() {
    Serial.begin(9600);      // Start Serial communication to
    dht.begin();                  // receive messages from the Arduino
}                               // and initialize the DHT sensor.

void loop () {
    //Collect data and print them.
    float humidity = dht.readHumidity();
    float temperature = dht.readTemperature();
    Serial.print(temperature);
    Serial.print(", ");
    Serial.print(humidity);
    Serial.print("\n");
    delay(1000);
}
```

Listing 2.1: A simplified code example to collect and print temperature and humidity data from a DHT sensor every second

2.3 The prototype backend web service

I developed the prototype backend storage system as a part of my special curriculum [16] in the fall 2016. For a monthly cost of 23 USD, the backend collected and stored the air quality data gathered by the students. While it served its purpose for the pilot study, it was designed as a proof of concept, not to be used in an operational project. It therefore did not scale, it had a task queue which proved to be unreliable and a database that needed a better structure.

We implemented the backend system in Ruby on Rails 4 and deployed it on the Heroku[2] cloud application platform. It provided a simple web interface for uploading data, allowing the students to upload their data log files directly to the backend system, as well as a simple API for retrieving data within a given time range. Like App Engine that we now use, Heroku is a PaaS, enabling us to focus on writing code, while they take care of infrastructure and maintenance. They also provide "Add-ons", which are fully managed cloud services that are integrated into the Heroku platform. The Add-ons come in different pricing plans, many also include a free plan for development, and simplify installing new services and managing billing, credentials and configurations directly from the Heroku Dashboard or CLI. For our backend storage system, we used three Add-ons: Heroku Postgres[3], Memcachier[4] and Redis To Go[5]. Together with Ruby libraries, the Add-ons respectively provided us with a PostgreSQL database of 10 million free records, a cache store and a Resque task queue.

2.4 Pilot frontend service

To simplify the process of accessing the collected air quality data, we also created a pilot frontend service[6] consisting of a web application and a web server. At the web application, the students and the public could use provided interfaces to explore air quality data from the last 24 hours, see air quality forecasts from luftkvalitet.info (a service provided by NILU), and view and download data from any time period. The web server acted as a means of communication between the web application and the data sources, translating queries and providing the results to the web application.

Figure 2.2 shows a screenshot from the pilot web application, visualizing mea-

2. heroku.com
3. elements.heroku.com/addons/heroku-postgresql
4. elements.heroku.com/addons/memcachier
5. elements.heroku.com/addons/redistogo
6. Designed and implemented by Bjørn Fjukstad

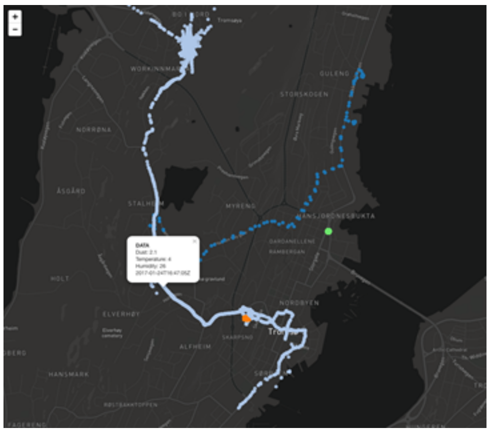


Figure 2.2: Data visualization in the protype web application.

surements from two different air:bits carried around in Tromsø, Norway. The green and orange dots represent data from the two stationary air monitoring stations run by NILU. The web application worked well in the pilot project, but it was tailored to explore and visualize air quality data collected only in Tromsø. Expanding the project to schools in other parts of Northern Norway, would require interfaces and visualization mechanisms that included the new locations. We also wanted to create a more advanced data exploration interface to enable the students to retrieve more specific datasets from the data sources.

2.5 The air:bit - current version

In fall 2017 we created the second, current version of the air:bit[7], shown in figure 2.3. We have replaced the DHT11 sensor with the DHT22, enabling us to register temperatures below 0°C. By replacing the Sharp GP2Y1010AU0F with the Nova SDS011 we can also register both PM2.5 and PM10 concentrations in the air. The air:bit is otherwise very similar to the prototype, in regard to physical components, source code and the format in which the data log files are written (listing 2.2). The order of the data is still important in order for it to be correcly inserted into the platform storage system.

```
Time,     Latitude, Longitude, PM10, PM25, Humidity, Temperature
2018-03-15T06:39:35.000Z, 67.284576, 14.436530, 17.00, 1.10, 28.60, 3.60
2018-03-15T06:39:40.000Z, 67.284576, 14.436518, 15.50, 1.10, 25.70, 2.50
2018-03-15T06:39:45.000Z, 67.284584, 14.436503, 14.60, 1.10, 25.40, 2.40
2018-03-15T06:39:50.000Z, 67.284591, 14.436497, 2.40, 0.70, 25.30, 2.30
2018-03-15T06:39:55.000Z, 67.284591, 14.436492, 3.00, 0.60, 25.10, 2.20
```

Listing 2.2: The air:bit data log file format

7. Designed and implemented by Bjørn Fjukstad, Hedinn Gunhildrud, Morten Grønnesby, Fredrik Rasch and Ken-Arne Jensen.

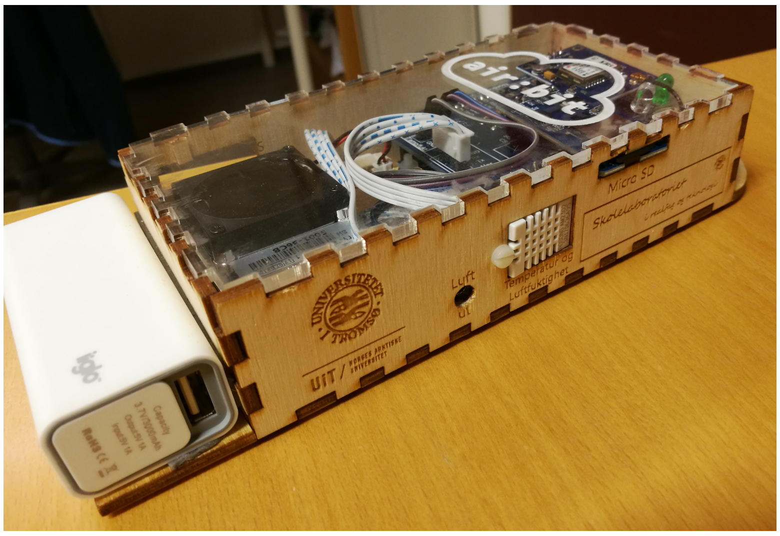
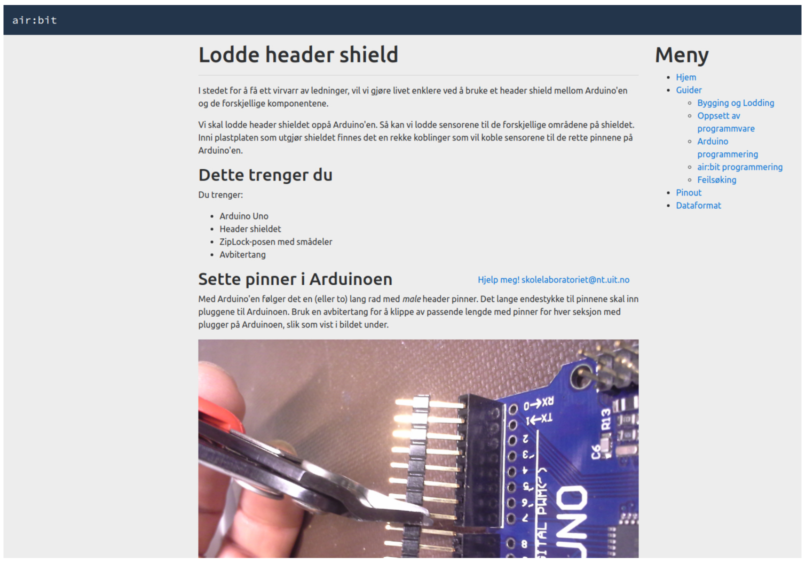


Figure 2.3: The current air:bit.

2.6 The air:bit platform educational resources

Since the participating schools are spread across Northern Norway, we are not able to host every class at UiT and assist the students and teachers at all times. Therefore, we have created educational resources for the project and published them at airbit.uit.no. The resources consist of wiki pages[8] with instructions on how to build and program the air:bits (figure 2.4), guides for installing the required software, an introduction to Arduino programming and help with debugging code. They also include video lectures[9] on air quality monitoring, how air pollution affects our health, and tips and tricks for doing research (figure 2.5).

8. airbit.uit.no/public/wiki/Home.html
9. airbit.uit.no/resources



air:bit

Lodde header shield

I stedet for å få ett virvarr av ledninger, vil vi gjøre livet enklere ved å bruke et header shield mellom Arduino'en og de forskjellige komponentene.

Vi skal lodde header shieldet oppå Arduino'en. Så kan vi lodde sensorene til de forskjellige områdene på shieldet. Inni plastplaten som utgjør shieldet finnes det en rekke koblinger som vil koble sensorene til de rette pinnene på Arduino'en.

Dette trenger du

Du trenger:

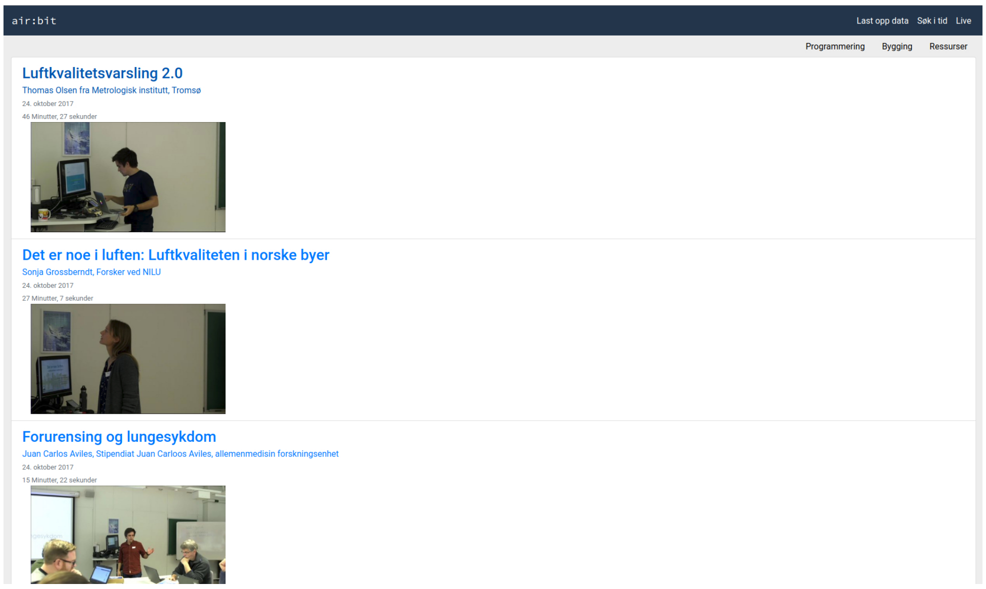
- Arduino Uno
- Header shieldet
- ZipLock-posen med smådeler
- Avbitertang

Sette pinner i Arduinoen

Hjelp meg! skolelaboratoriet@nt.uit.no

Med Arduino'en følger det en (eller to) lang rad med *male* header pinner. Det lange endestykke til pinnene skal inn pluggene til Arduinoen. Bruk en avbitertang for å klippe av passende lengde med pinner for hver seksjon med plugger på Arduinoen, slik som vist i bildet under.

Figure 2.4: The students are provided with detailed instructions on how to build their air:bits.



Luftkvalitetsvarsling 2.0
Thomas Olsen fra Metrologisk institutt, Tromsø
24. oktober 2017
46 Minutter, 27 sekunder

Det er noe i luften: Luftkvaliteten i norske byer
Sonja Grossberndt, Forsker ved NILU
24. oktober 2017
27 Minutter, 7 sekunder

Forurensing og lungesykdom
Juan Carlos Aviles, Stipendiat Juan Carloos Aviles, allemenmedisin forskningsenhet
24. oktober 2017
15 Minutter, 22 sekunder

Figure 2.5: Informative lectures are filmed and posted to the air:bit web application.

3 Architecture

The air:bit platform is a system for storing, exploring and visualizing air quality data from ait:bits and other, external data sources. Figure 3.1 illustrates the air:bit platform architecture, which is three-tiered and consists of i) the air:bit web application, an interactive system for uploads, downloads and visual exploration of air quality data; ii) a frontend web server that translates user interactions in the air:bit web application into queries for storing air:bit data log files, retrieving air:bit data and integrating air quality data from external data sources; and iii) a backend that stores and provides air:bit data for the air:bit web application.

3.1 The air:bit web application

The air:bit web application is the point of interaction with the air:bit platform. It provides users with different interfaces to upload air:bit data log files, and explore and download their collected data through interactive visualizations. The visualizations also integrate the student data with external sources. The web application translates user inputs (air:bit data log files, pressed keys and mouse clicks) from the interfaces into upload requests and data queries, which are sent to the the frontend web server. It also interprets and visualizes the responses and the data that is returned.

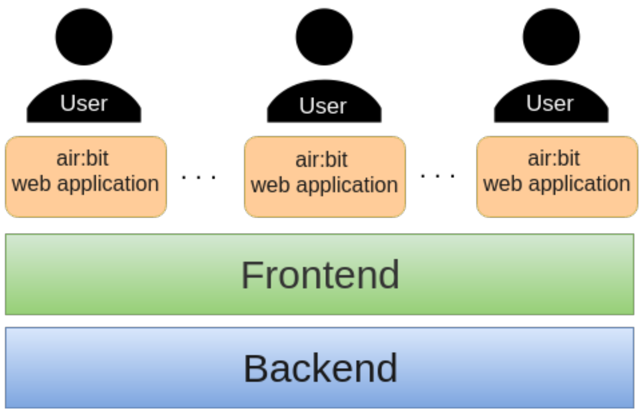


Figure 3.1: The air:platform architecture.

3.2 Frontend web server

The frontend web server works as a middleman between the air:bit web application, and the backend and the external sources. It exposes an API to the web application, enabling multiple users to simultaneously upload air:bit data log files and explore air quality data. The frontend web server translates the upload requests from the web application into upload requests that are understood by the backend. It also translates the web application queries into requests that can be executed by the backend and the external sources. Returned query results are formatted by the frontend according to their purpose in the web application and returned to the web application and the users.

3.3 Backend

The backend is responsible for storing the air:bit data log files uploaded by the air:bit web application users, for making the air:bit data searchable and for providing the web application with air:bit data. It exposes a simple API which the frontend web server uses to send it translated user uploads and queries. The backend retrieves the air:bit data, which it processes according to the queries before returning the result to the frontend web server.

4 Design and implementation

The air:bit platform architecture consists of three components, the air:bit web application, the frontend web server and the backend. We have designed and implemented the three components as two microservices, a frontend service including both the air:bit web application and the frontend web server, and a backend service that is the data management system (figure 4.1). Microservices are separate, autonomous services that communicate via network calls [17], in our case an HTTP REST API. There are several reasons to why this software architecture suits our platform. First, since the microservices are independent of each other, they can be modified and deployed by themselves. Using microservices, we reduce the amount of affected/broken functionality when something goes wrong during or after a deploy, and we minimize the area of code that we need to debug in order to fix the error. Second, breaking the codebase into smaller parts makes it easier to maintain. Third, by communicating using open HTTP APIs microservices allow us to implement each service in the most suitable programming language and technology [17]. Right now we have implemented the air:bit web application in HTML, Javascript and CSS, the frontend web server in Go and the backend in Ruby on Rails in the Google Cloud Platform. The final reason is that microservices allow us to scale the parts of the platform individually. While the frontend service must scale with respect to the number of users on the website, to handle an increasing amount of traffic, the backend must scale according to the amount of data that needs to be processed, to ensure timely database inserts and retrievals for the users.

4.1 Frontend service

The frontend service includes the air:bit web application and the frontend web server[1]. It is responsible for providing the students with services that enable them to store and explore air:bit data, and explore integrated air quality data and forecasts from NILU. We are currently also working on expanding the data exploration interfaces, to enable exploration of integrated precipitation data from MET. The air:bit web application runs in the web browser and is accessible at airbit.uit.no. It is hosted by the frontend web server, which feeds it data from the backend, NILU and MET.

4.1.1 The air:bit web application

The air:bit web application is the students' point of interaction on the air:bit platform. With the use of HTML and JavaScript libraries, it enables the students to interactively upload and query air quality data in a web browser, either from a computer or a smartphone, using the following three interfaces.

Uploading data log files

Figure 4.2 shows the interface for uploading air:bit data log files. It consists of a file input field which opens a file dialog, allowing the students to select data log files from their computer. Once selected, the students can select whether they want to validate the format of the data log files or upload them to the air:bit platform.

Exploration historical data

Figure 4.3 shows the interface for exploring historical data. The students can retrieve the air quality datasets they need to answer their research questions by specifying an area, a time range and the data sources they wish to include data from. When specifying an area, the students either i) select an area from a list of predefined options containing the eight areas where the participating

1. Bjørn Fjukstad, Fredrik Rasch, Morten Grønnesby and Pontus Aurdal contributed to the implementation of the frontend service.

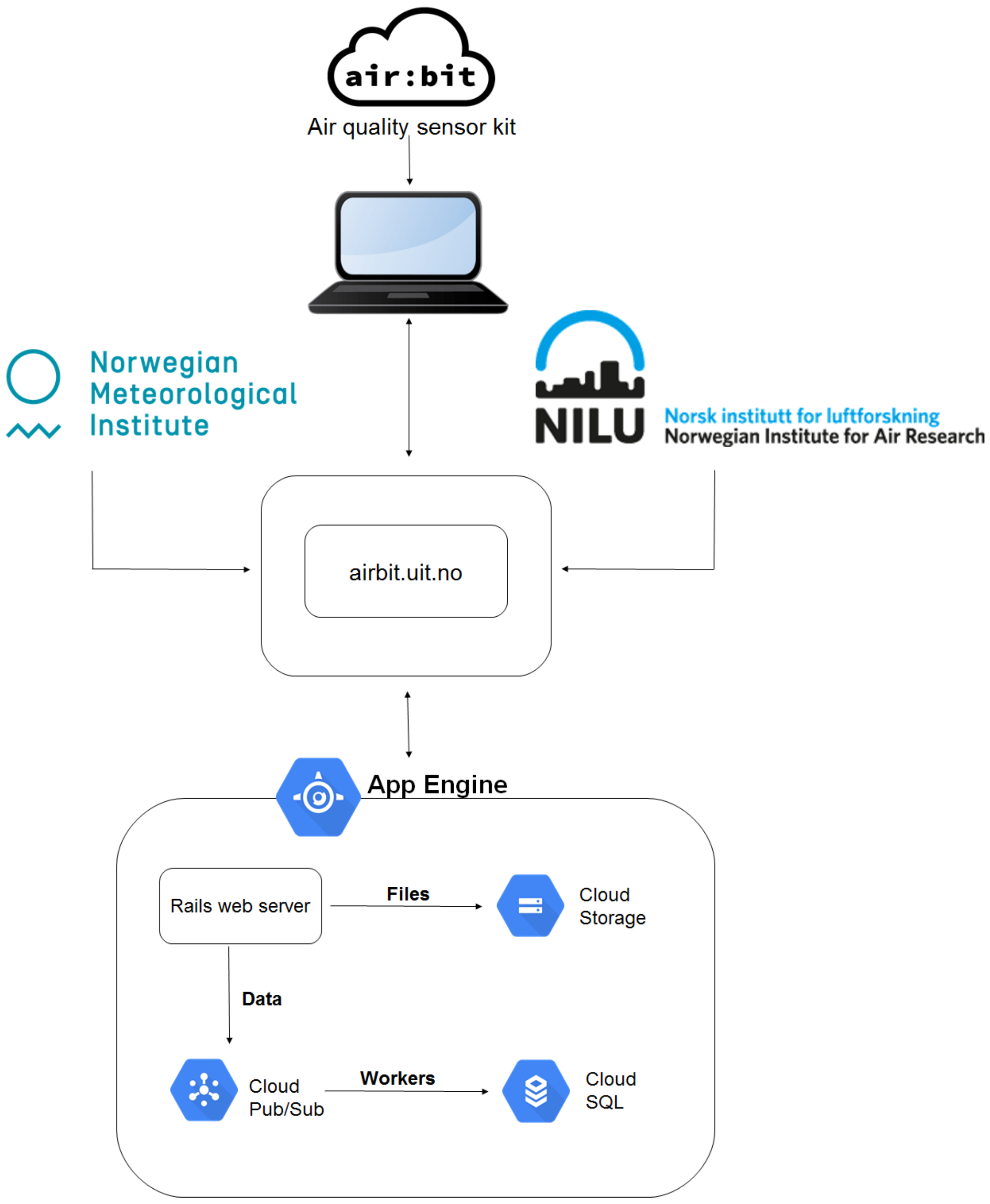
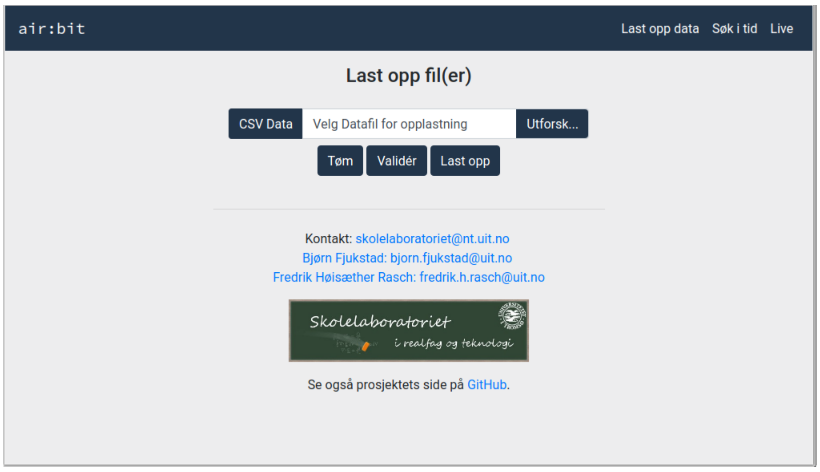


Figure 4.1: The air:platform design.



Kontakt: skolelaboratoriet@nt.uit.no
Bjørn Fjukstad: bjorn.fjukstad@uit.no
Fredrik Høisæther Rasch: fredrik.h.rasch@uit.no

Se også prosjektets side på GitHub.

Figure 4.2: The air:bit web application provides a simple upload interface that also includes data format validation.

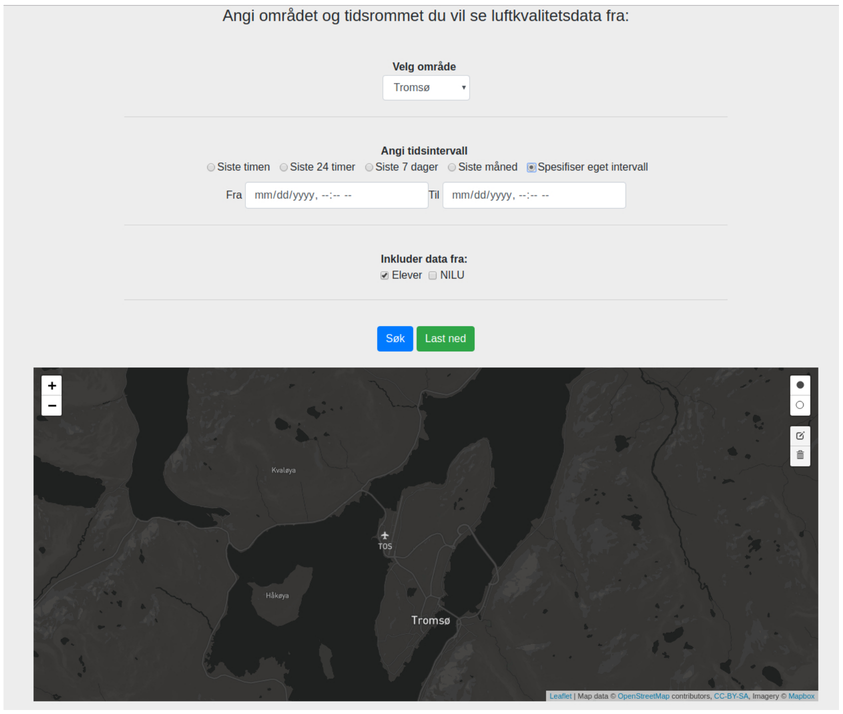
schools are located; or ii) define their own area by drawing a radius circle on the map. The students are provided with four predefined options when specifying a time range: past hour, past 24 hours, past 7 days, past month. They can also use a custom time range by specifying a "to" and "from" time. The students can choose to retrieve datasets from the backend, from NILU or both.

Based on the user request, the retrieved data is either visualized on a map and in charts using the JavaScript libraries D3[2] and Plotly[3] (respectively), as shown by figures 4.4 and 4.5, or downloaded to the local storage of the device that runs the air:bit web application. The visualization tools help the students discover air quality trends related to time and location.

Viewing "live" data

Figure 4.6 shows a screenshot of the interface for retrieving "live" data, i.e. air quality data that has been collected within the past 24 hours. This interface is a simplified version of the interface for exploring historical data, using the same predefined list of areas and the same two data sources. The students can also choose to view an air quality forecast from luftkvalitet.info.

2. d3js.org/
3. plot.ly/


Angi området og tidsrommet du vil se luftkvalitetsdata fra:

Velg område

Tromsø

Angi tidsintervall

Siste timen Siste 24 timer Siste 7 dager Siste måned [✓ Spesifiser eget intervall]

Fra mm/dd/yyyy, --:-- -- Til mm/dd/yyyy, --:-- --

Inkluder data fra:

[✓ Elever] NILU

Søk
Last ned
Kvaløya
TOS
Håkøya
Tromsø
Leaflet | Map data © OpenStreetMap contributors, CC-BY-SA, Imagery © Mapbox

Figure 4.3: With the air:bit web application data exploration interface, the students can query air quality data from the backend and NILU.

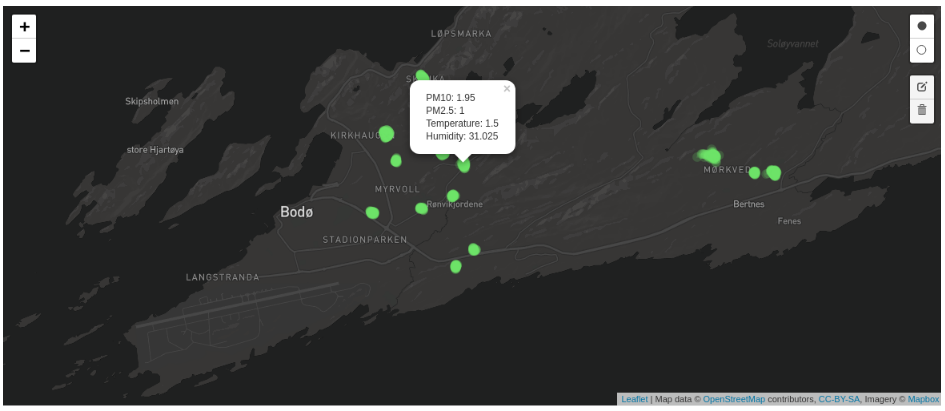

PM10: 1.95
PM2.5: 1
Temperature: 1.5
Humidity: 31.025

Figure 4.4: Map with air:bit measurements collected in Bodø in March 2018.

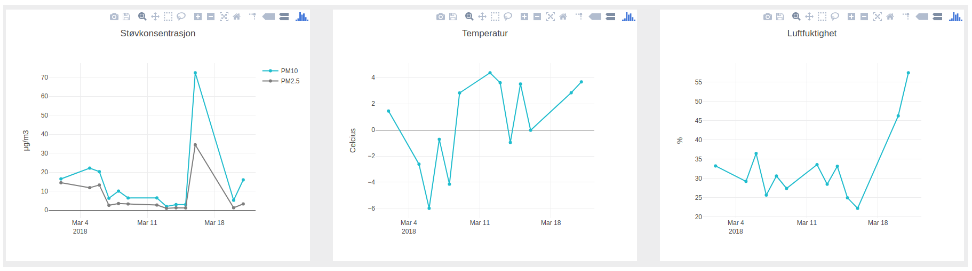
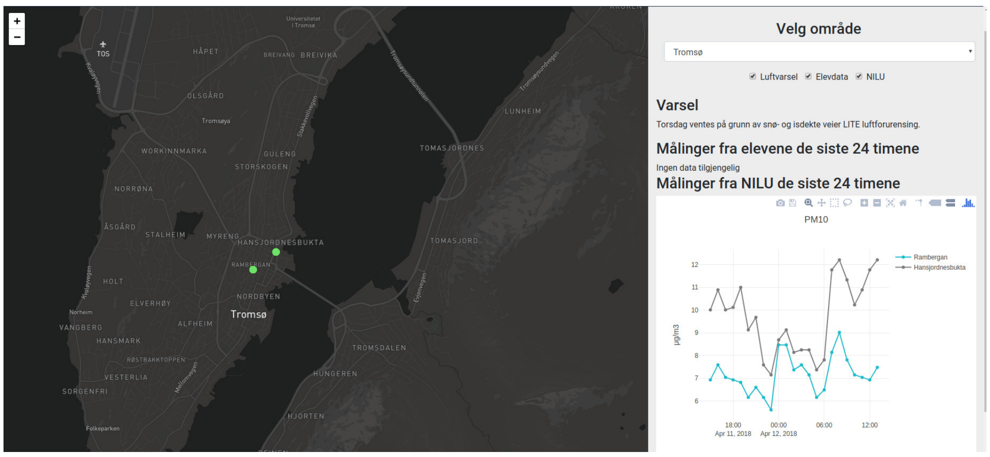


Figure 4.5: Charts visualizing the levels of PM10 and PM2.5 (left), temperature (middle) and humidity (right) in Bodø in March 2018 based on collected air:bit data.



Velg område

Tromsø

✓ Luftvarsel ✓ Elevdata ✓ NILU

Varsel

Torsdag ventes på grunn av snø- og isdekte veier LITE luftforurensing.

Målinger fra elevene de siste 24 timene

Ingen data tilgjengelig

Målinger fra NILU de siste 24 timene

Figure 4.6: With the live data exploration interface, the students can view data collected within the past 24 hours.

4.1.2 Frontend server

The frontend web server hosts the air:bit web application and acts as the translation layer between the web application and the multiple data sources. It is implemented as an API, and in addition to being used by the web application, other visualization tools/apps can also use the API to query and visualize data from the backend, NILU and MET. We provide a detailed description of the the API resources in table 4.1. The frontend server runs in a Docker container on a local server at the Department of Computer Science at UiT. Since the web server is stateless we can scale it out horizontally by replicating the container and placing the containers behind a HTTP load balancer such as nginx[4].

4.2 air:bit backend data management system

The air:bit backend stores air:bit data log files uploaded by the students and processes data queries generated by the data exploration interfaces at the air:bit web application. It is run by a Ruby on Rails application deployed to App Engine, a Platform as a Service delivered by the Google Cloud Platform[6] (GCP). The application in App Engine consists of two services, a web service and a worker service. When we deploy the two services to App engine, the code is run on instances, i.e computing units used to automatically scale the application [18]. Instances running the web service each run their own web server and handle HTTP requests from the frontend web server, while instances running the worker service process the uploaded air:bit data log files. The application uses three GCP products in the process of storing the uploaded data log files: Cloud Storage[7], Cloud Pub/Sub[8] and Cloud SQL[9] (figure 4.7).

4.2.1 Cloud Storage

Google Cloud Storage is a microservice that allows us to store binary data in a high-level container for storing binary objects, known as a *bucket* [19]. Each bucket has three properties that are specified when the bucket is created: a name, a location and a storage class describing the availability and minimum storage duration of the bucket contents [20]. Using the Cloud Storage storage classes, it is possible to implement a storage system similar to Facebook's f4 and Haystack, with designated buckets for hot and cold data [21]. However, since

4. docker.com/what-container
6. cloud.google.com/
7. cloud.google.com/storage
8. cloud.google.com/pubsub/
9. cloud.google.com/sql/

Resource	Description
GET /niluaqis?area=area&from=[...]&to=[...]& component=component Parameter format: from, to: [yyyy-mm-ddThh:mm:ss.sssZ]	Retrieves live air quality data from NILU and returns GeoJSON[5] data points for map visualization. The component parameter describes the air pollutant to search for, and is either "PM10" or "NO_2"
GET /historical?area=area&from=[...]&to=[...]& component=component Parameter format: from, to: [yyyy-mm-ddThh:mm:ss.sssZ]	Retrieves historical data from NILU based on query parameters, for charts and download
GET /forecast?area=area	Retrieves air quality forecast from luftkvalitet.info
GET Predefined area: /studentaqis?area=area&from=[...]&to=[..]& plotmap=true Specified area on map: /studentaqis?within=[..]&from=[...]&to=[..]& plotmap=true Parameter format: within: [latitude,longitude,radius (in kilometers)] from, to: [yyyy-mm-ddThh:mm:ss.sssZ]	Retrieves air:bit data points based on a query and returns GeoJSON data points for map visualization.
GET Predefined area: /student?area=area&from=[...]&to=[..] Specified area on map: /student?within=[..]&from=[...]&to=[..] Additional parameters: plotchart: true Parameter format: within: [latitude,longitude,radius (in kilometers)] from, to: [yyyy-mm-ddThh:mm:ss.sssZ]	Retrieves air:bit data based on query parameters, and returns it in a CSV format for visualization in charts and download. By adding the additional parameter plotchart, the backend will return aggregated data for the graphs. Otherwise, raw data is returned. Depending on the duration of the time frame, the aggregated data is: A minutely mean if (duration <1 hour) An hourly mean if (1 hour <duration <7 days) A daily mean if (7 days <duration <1 month) A monthly mean if (duration >1 month)
GET /precipitation	Retrieves precipitation from MET This functionality has been implemented, but we have not yet integrated it into the visualization tools.
POST /sendfile	Uploads data log files to the air:platform
GET /	Renders views for index page
GET /live	Renders views for live page
GET /history	Renders views for data exploration page
GET /upload	Renders views for upload page
GET /resources	Renders views for resources page

Table 4.1: The RESTful interface of the frontend web server.

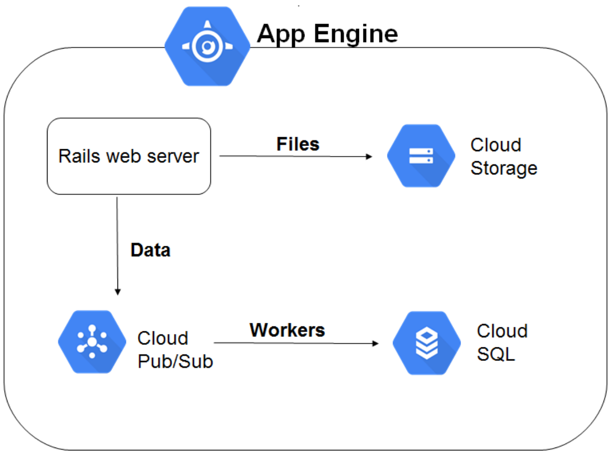


Figure 4.7: air:bit backend design.

we will not reach the amounts of data that Facebook has, we make use with one bucket designed for high frequency access. When deploying an application in the GCP, three additional buckets are created by default: i) a default application bucket where the first 5GB of storage are free; ii) a bucket for storing backups of the application's Docker container images; and iii) a bucket for storing temporary files (used for staging and test purposes). For the air:bit platform, we use Cloud Storage to store the raw data log files. We store the data in the default application storage bucket, which is multi-regional (i.e redundant in at least two locations) and located in Europe. Due to incomplete Cloud Storage documentation for the Ruby programming language, we initially created a new bucket for the data log files which had a monthly cost per GB stored. This was a multi-regional bucket located in the US, providing us with redundant storage in at least two locations. We later discovered the default application bucket and chose to transfer the data log files there since it too is multi-regional, allows us to store the data for free and is located in Europe.

We store the data log files in Cloud Storage in case the database insertion process fails. Based on experiences from the pilot project and software development in general, things will fail. Since we are storing data from novice programmers and failures may be due to wrongly formatted data log files, it is important that we can inspect the files to find the cause of the error and inform the teachers.

4.2.2 Cloud Pub/Sub

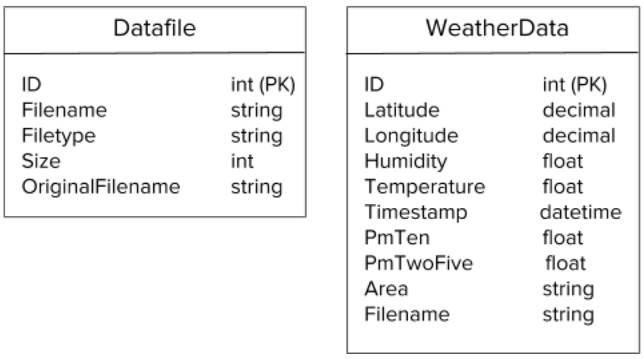
Google Cloud Pub/sub is a service that provides a secure, many-to-many, asynchronous messaging [22]. Our platform uses Cloud Pub/Sub to enqueue the data log files in a message queue, so that we can asynchronously parse and insert their contents into our database using background worker processes.

To use this GCP service with our Rails application, we use the activejob-google_pub_sub library[10], which is a Google Cloud Pub/Sub adapter and worker for ActiveJob[11] [12]. The library creates a topic (a queue) to which the data log files are published, and creates a pull subscription[13] which the background workers collect data log files from. This enables fast uploads and asynchronous data processing, allowing us to quickly inform the students about successful or failed uploads. An upload is technically successful once the data log file contents have been completely inserted into our database, but this might take some time (depending on the number of parallel uploads and the size of the data log files). However, the Cloud Pub/Sub durable messaging, guarantees that a correctly formatted file added to the message queue, will be processed and have its contents inserted into the database in the near future. Thus we can deem a data log file upload successful once the file is successfully inserted into the message queue and send a response message back to the student. Also, should the system fail and the data log file is taken out of the message queue before it is processed, the storing in Cloud Storage enables us to manually re-insert the data log file into the message queue.

4.2.3 Cloud SQL

In order to enable students to query the air:bit data, the air:bit measurements are stored in a relational Google Cloud SQL Second Generation MySQL database. The GCP offers two types of MySQL instances, the First Generation MySQL and the Second Generation MySQL. While the former is the legacy instance type, the latter is the newest instance type and the recommended option for a Cloud SQL database. In addition to being a fully-managed database service, it also offers scalability [23] and simple mechanisms for changing the virtual machine type of the database instance. The air:platform database is currently run on an instance with 1 virtual CPU (vCPU), 3.75GB memory and 25 GB SSD, and is located in Belgium.

10. github.com/ursm/activejob-google_cloud_pubsub
11. ActiveJob is Rails' built-in framework for performing asynchronous queuing
12. edgeguides.rubyonrails.org/active_job_basics.html
13. cloud.google.com/pubsub/docs/subscriber



Datafile	
ID	int (PK)
Filename	string
Filetype	string
Size	int
OriginalFilename	string

WeatherData	
ID	int (PK)
Latitude	decimal
Longitude	decimal
Humidity	float
Temperature	float
Timestamp	datetime
PmTen	float
PmTwoFive	float
Area	string
Filename	string

Figure 4.8: The air:bit database contains two tables, one for storing data log file metadata and one for storing air:bit measurements.

The database currently holds two tables: Datafile and WeatherData (figure 4.8). The Datafile table stores metadata about the uploaded data log files, such as filename, type, size and the original filename. The WeatherData table stores the air:bit measurements. In addition to the sensor data (latitude, longitude, humidity, temperature, timestamp, PM10 and PM2.5), we also store the name of the data log file which the measurement derive from. Mapping the measurements to their derived data log files has been very helpful as it enables us to remove faulty data from the database at the students' request. For example, this spring some students discovered that they had switched the columns for PM10 and PM2.5 in several of their data log files. This manifested in the database and affected everyone's data query results. Since we had mapped the database measurements to filenames, their teacher could give us the name of the affected files and we were able to identify and remove the faulty data from the database. We also assign an area to the measurement if it is within an 8 km radius of any of our eight predefined areas from the data exploration interfaces at the air:bit web application. We do this to optimize database queries and we have also added indexes on the table columns *area, latitude, longitude and timestamp* for the same effect.

4.2.4 Data privacy

Since a big part of the air:bit project involves students collecting air quality data in their local whereabouts, data privacy becomes a topic. If not anonymized correctly, the air:bit data can reveal private information about the student that collects it, such as where they live, where they go and who they spend time with. We have been granted permission by the The Norwegian Data Protection Authority to run the project only if we can ensure that the air:bit data cannot be used to trace individual students. Our most effective measure to achieve this is by preventing personal information from entering the air:bit platform.

Resource	Description
POST /api/upload	Uploads a data log file to the air:bit data management system

GET /api/data?totime=[...]&fromtime=[...]
 	
Additional parameters:
area=area
within=[latitude,longitude,radius (in kilometers)]
plotmap=true
plotchart=true
 	
Parameter format:
totime, fromtime: [yyyy-mm-ddThh:mm:ss.sssZ]
	Returns all of the air:bit measurements satisfying the given query
	
	Additional parameters:
	area: Returns air:bit measurements that are mapped to the given area
	
	within: Returns air:bit measurements collected within the radius circle that is made of the parameter values.
	
	plotmap: Returns aggregated data for visualization on map. Database entries are grouped by coordinates and for each key the mean of the values are calculated.
	
	plotchart: Returns aggregated data for visualization in charts. Database entries are grouped by timestamp and for each key the mean of the values are calculated

Table 4.2: The RESTful interface of the air:bit backend system.

We have informed the students that they should not collect data as they are leaving or arriving their home, or spending time at locations related to their health, religion, etc. They should also not include their names or group name in the air:bit data log files (or the filenames) that they upload to the air:bit platform, since we store the files and filenames in our systems. This information is, however, only visible to the administrators of the air:bit platform backend data management system. In terms of data privacy, we guarantee that the air:bit data which is returned to the air:bit platform after a query, can never with complete certainty be traced to a specific air:bit or student based only on the data itself.

4.2.5 air:bit backend data processing

The main responsibilities of the air:bit backend include processing air:bit data, whether it is parsing data log files and inserting measurements into the database, or retrieving data from the database based on a web query and aggregating the data. The air:bit backend exposes an HTTP REST API, described in table 4.2, enabling the frontend web server, as well as other applications and users, to send air:bit data log files and web queries for it to process.

Database insert

When the backend receives a data log file, it creates a new entry in the Datafile table. Since Cloud Storage does not have a way of handling duplicate filenames other than overwriting files, we query the database for data log files with the same name. Based on the result, we either add the appropriate version number to the filename or we do not, before storing the data log file in Cloud Storage and adding the filename to the message queue.

A worker process polls the message queue, fetches a filename and proceeds to search for the file in Cloud Storage. Sometimes the worker might pull the filename from the queue before the file has been stored, in which case we allow the worker 30 attempts before marking the file as non-existent and aborting the processing process. If a worker cannot find the file in Cloud Storage, it is most likely due to the file having an invalid filename. We are sanitizing the filenames at the frontend web server and thus we can ensure that data log files uploaded through the website have valid filenames. However, we keep the limited amount of tries in case someone uploads a data log file using the HTTP REST API directly, in order to avoid eternal search loops to Cloud Storage and hogging of worker resources in App Engine.

When the worker successfully retrieves the data log file from Cloud Storage, it parses the contents into air:bit measurements. Each measurement goes through a simple validation (checking for null values, invalid timestamps and invalid coordinates), in order to remove those that will cause the database insertion to fail. As a means to optimize database queries later, we also assign an area to the measurements if they are located within an 8 km radius of one of our predefined areas at the air:bit web application. Finally we do a bulk insert[14] of all the data log file measurements into the WeatherData table, i.e. inserting all measurements into the database in one transaction. Some of the students' air:bit data log files contain more than 44,000 measurements and bulk inserts greatly optimizes the insert process compared to doing the it the default way of one database transaction per measurement.

Database retrieval

Queries that are generated by the data exploration interfaces in the air:bit web application contain parameters specifying a time range (predefined options are translated into a "to" and "from" time at the frontend web server) and an area. If the students select an area from the predefined list of areas, the name of the area is included in the query and later translated into coordinates by the backend. If they specify an area on the map, the query will contain coordinates and a radius. The queries may also contain a parameter specifying if the data will be visualized in charts or on the map. If this is not provided, the data will be downloaded by the students.

14. github.com/zdennis/activerecord-import

To process a data query, the web service starts by collecting all WeatherData table entries that have been collected within the given time range. If the query contains the parameter “area”, the web server will traverse the set of collected entries, use a database index to find all entries that have been mapped to the parameter value (e.g. Tromsø), and create a new subset of those. If the query contains the parameter “within”, the web service will traverse the set of collected entries, calculate the distance between each entry and the parameter value (e.g. [69.68795, 18.944174, 4.752]) and generate a new subset of those entries. If the data is to be downloaded, we return the latest subset of data as JSON objects without any further processing, to provide students with raw data.

Data that is to be visualized in charts or on the map needs to be aggregated before it is returned to the frontend web server. Most of the data is very fine-grained (one measurement per 2-5 seconds) and data queries might return tens (sometimes hundreds) of thousands of measurements. Returning large amounts of data points and having the web application visualize them all, causes both the web application and backend to run out of memory and crash. We reduce the number of data points returned to frontend web server, by grouping the data based on GPS coordinates (map) or timestamp (charts) and calculating an average for each of the remaining sensor data columns. The result set is returned as JSON objects to the frontend web server.

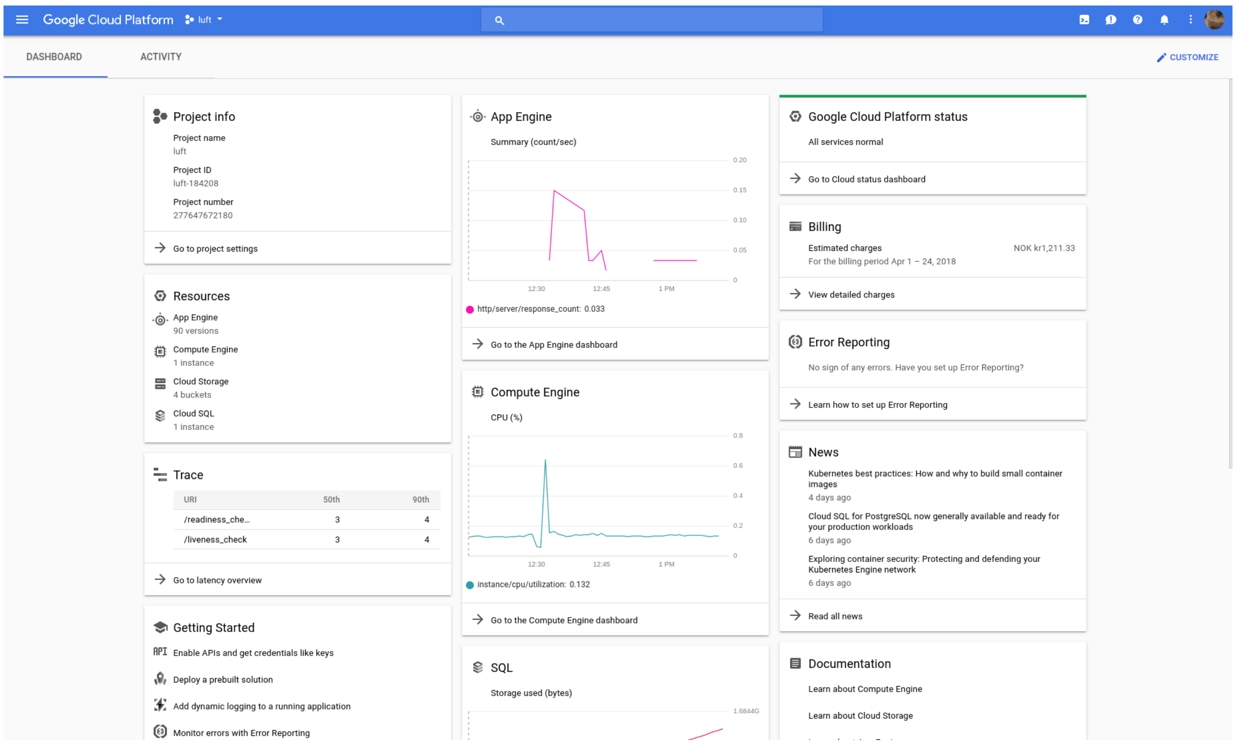
4.2.6 App Engine

Applications in App Engine can run the App Engine flexible environment[15], the App Engine standard environment[16] or both. There are pros and cons to both environments, but we chose to deploy the backend web application to the App Engine flexible environment since it allows background processes and modification of the runtime. It is also the only environment that supports Ruby applications.

App Engine itself helps us satisfy the platforms maintainability and scalability requirements. Being a PaaS, it provides a fully managed environment, abstracting away the infrastructure. Using the GCP Cloud Console[17] (figure 4.9) we can easily manage and get insights into the services used by our application. In the App Engine flexible environment we can specify how App Engine should automatically scale the application according to the amount of data being processed, and we can configure the resource settings of the application to

15. cloud.google.com/appengine/docs/flexible/
16. cloud.google.com/appengine/docs/standard/
17. cloud.google.com/cloud-console/

ensure we have enough memory, disk space and vCPUs for our needs.



Project info
Project name
luft
Project ID
luft-184208
Project number
277647672180
Go to project settings

Resources
App Engine
90 versions
Compute Engine
1 instance
Cloud Storage
4 buckets
Cloud SQL
1 instance

Trace

URI	50th	90th
/readiness_che...	3	4
/liveness_check	3	4

Go to latency overview

Getting Started
Enable APIs and get credentials like keys
Deploy a prebuilt solution
Add dynamic logging to a running application
Monitor errors with Error Reporting

App Engine
Summary (count/sec)
● http/server/response_count: 0.033
Go to the App Engine dashboard

Compute Engine
CPU (%)
● instance/cpu/utilization: 0.132
Go to the Compute Engine dashboard

SQL
Storage used (bytes)

Google Cloud Platform status
All services normal
Go to Cloud status dashboard

Billing
Estimated charges NOK kr1,211.33
For the billing period Apr 1 – 24, 2018
View detailed charges

Error Reporting
No sign of any errors. Have you set up Error Reporting?
Learn how to set up Error Reporting

News
Kubernetes best practices: How and why to build small container images
4 days ago
Cloud SQL for PostgreSQL now generally available and ready for your production workloads
6 days ago
Exploring container security: Protecting and defending your Kubernetes Engine network
6 days ago
Read all news

Documentation
Learn about Compute Engine
Learn about Cloud Storage

Figure 4.9: The GCP Cloud Console Dashboard, providing us with an insight into the services used by our application.

Deployment configurations

```
env: flex
runtime: custom
automatic_scaling:
  min_num_instances: 2
  max_num_instances: 4
  cpu_utilization:
    target_utilization: 0.5
resources:
  memory_gb: 1.5
```

Listing 4.1: An extract from the app.yaml configuration file.

The web service and the worker service are deployed with the same source code, but with an individual configuration files. Listing 4.1 shows an extract of the configuration file for the web service. We have configured the service to always run on two instances, with the possibility of scaling to maximum four instances if the average CPU usage across all instances exceeds 50%. CPU usage typically increases when handling data queries from the frontend web server, especially when the queries are parallel and complex. We have configured the web service to run on machine types that provides us with one virtual CPU (vCPU) and 10 GB disk (default values), and at least 1.5 GB of memory (from the default 0.6 GB) to avoid running out of memory while processing the data queries. We initially had only one web instance with 1 vCPU, 1GB of RAM and 10GB persistent disk. However, we experienced that the web server would periodically crash and not restart, taking down the backend. We therefore added a second default web instance for redundancy in mid-March.

The worker service always run one instance and can scale to a maximum of 15 instances (can be increased in need be) if the average CPU usage exceeds 40%. It is configured like the web service, but runs on machine types that provides one vCPU, 1 GB of RAM and 10GB persistent disk. Worker instances spend all of their computational power on parsing and inserting data log files into the database, and the CPU usage typically increases when the workers are processing multiple data log files containing tens of thousands of measurements.

5 Air:bit education project

The ait:bit project is a collaborative project between the School Laboratory at the Faculty of Science and Technology at UiT - the Arctic University of Norway, NILU and MET. It is offered to upper secondary schools across Northern Norway that are participating in the Lektor2 initiative. The Lektor2 initiative is funded by the Norwegian Ministry of Education, to enable educators to collaborate with the industry to create learning resources that motivate students and raise their interest in science and technology subjects[1]. In our pilot project in spring 2017 only one school class of about 30 students participated, building and programming eight air:bit prototypes.

In 2018 there were 16 participating classes from 11 upper secondary (videregående) schools spread across Northern Norway. The 174 students built and programmed 62 air:bits during the project and figure 5.1 shows the distribution of students and air:bits by the eight areas in which the students are located. The 16 teachers also built and programmed 15 air:bits during a two-day workshop in the fall 2018, in order to be able to guide their students later in the project. This raises the total number of air:bits that have been built and programmed during the 2018 air:bit project to 77.

1. lektor2.no/c1336841/artikkel/vis.html?tid=2181301

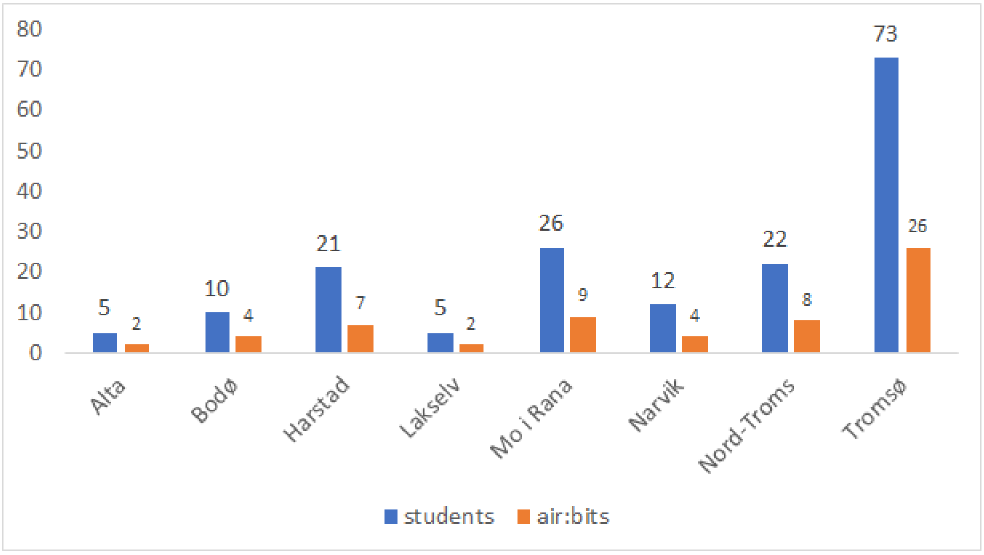


Figure 5.1: Distribution of students and air:bits by area.

5.1 air:bit project schedule

The air:bit project runs from August to May/June. In August, the schools are invited to participate in the project and in later in the fall the participating teachers are invited to the Department of Computer Science at UiT for a two-day workshop to learn how to build and program the air:bits. In January the students begin building and programming the air:bits in groups of 2-3. When the students are close to finishing building and programming the air:bits, we arrange for them to come to the Department of Computer Science at UiT for a day. There they are given a lecture on tips and tricks when doing scientific research and we assist them with any difficulties they might experience regarding their air:bits.

At some point early in the spring semester, the students develop their own air quality related research questions, such as "at what time during the day do our local kindergartens experience the highest level of air pollution?". From February to April/May, the students collect air quality data using the air:bits. In the air:bit web application the students upload their data to the air:bit platform and retrieve the datasets they need to analyze and answer their own research questions. The project is done in May/June when the groups of students present their findings and results to their fellow students and teacher.

The cost of participating in the air:bit project is based on the number of air:bit kits each school buy. The schools are billed per air:bit kit and the cost of each kit is 1,500 NOK. This covers the cost of the air:bit components, the air:bit workshop at UiT and support when they experience issues during the project. The teachers pay a participation fee to attend the two-day workshop in the fall. The fee is 3500 NOK per teacher, with a 500 NOK discount for each additional teacher from the same school. The participation fee also includes an air:bit kit.

6 Evaluation

In this chapter we evaluate the implementation of the air:bit platform with regard to latency, resource usage, scalability, and cost. Specifically, we evaluate the following six questions:

1. Data Log Upload Latency: What is the data log file upload time to the air:bit platform?

2. Data Query Latency: What is the air:bit platform query time for air quality data?

3. GCP Resource Usage: What is the resource usage on GCP with regards to storage, computation, and the number of instances?

4. GCP Operation Cost: What are the operating costs of hosting the air:bit platform backend management system on the GCP?

5. GCP Scaling Resource Usage: How does the air:bit platform scale with regards to the number of parallel data log file uploads and queries?

6. GCP Scaling Cost: What is the scale-up cost of the GCP during high-traffic periods?

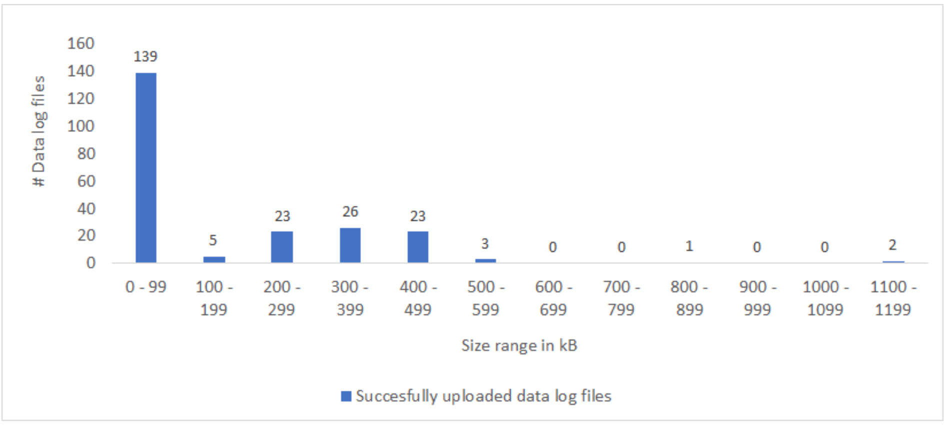


Figure 6.1: The size distribution of stored data log files in the backend database, for files uploaded from February through April 2018.

6.1 Data Log Upload Latency

To evaluate the upload time for data log files to the air:bit platform, we define an upload as the process that starts when a data log file is sent to the backend and ends when the file contents are completely parsed and inserted into the database. In the case of parallel uploads, latency time is measured from the first data log file is sent to the backend to the last data log file is completely inserted into the database. This is because Cloud Pub/Sub performs out-of-order delivery and we cannot guarantee that a specific data log file has been processed until all the parallel uploaded files are inserted into the database.

To measure upload latency we have generated two synthetic datasets[1]. We performed these tests in February, before the students started uploading their data. We then assumed that an average-sized data log file would be 717kB, containing 10,080 measurements, and that a "worst case"-sized data log file would be 3.2MB, containing 44,640 measurements. We chose these numbers since they respectively equal per-minute measurements for a week and per-minute measurements for a month. However, we overestimated the size of the students' data log files, since 97.2% of the uploaded air:bit data log files are less than 500kB in size (figure 6.1). For each of the datasets, we measured the latency of uploading one data log file, as well as the latency of parallel uploads (6, 12, 24, 48 and 96 data log files). For each number of parallel uploads we ran the benchmark six times. We configured the backend to scale to a maximum of 15 worker instances, and a target CPU of 40%, since these seemed reasonable for our testing purposes.

1. All benchmark datasets and scripts are stored online in github.com/ninaangelvik/luft .

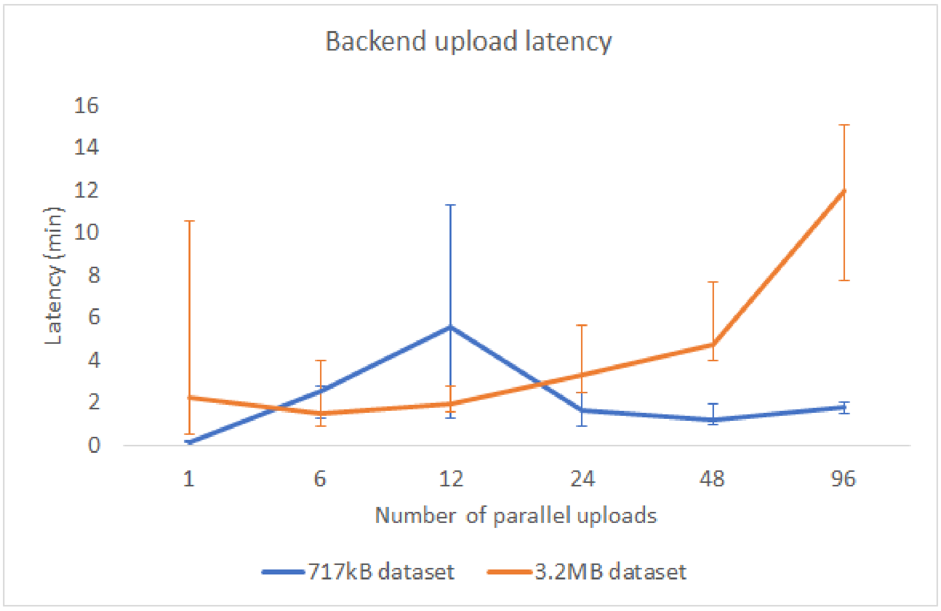


Figure 6.2: The average upload time in minutes for the two synthetic datasets, including the maximum and minimum upload time for each number of parallel uploads.

In each test we sent the data log files as individual HTTP POST requests to the backend, the same way the frontend web server sends data log files. To evaluate when the data log files were completely inserted into the database, we queried the database every three seconds for the number of unique filenames in the WeatherData table. We truncated the database periodically, but not after every test. Thus the size of the database might have affected the querying of filenames in the database, and by that the upload latency test results.

Figure 6.2 shows the average upload latency in minutes for each of the datasets. For each number of parallel uploads, we have marked the maximum and

# parallel uploaded files	Number of workers
1	1
6	1-3
12	4-5
24	5-9
48	9-15
96	15

Table 6.1: The number of active worker instances during the six benchmark runs for the 717kB dataset. An increase in workers during the runs is denoted by a dash between the lowest and highest number of active instances.

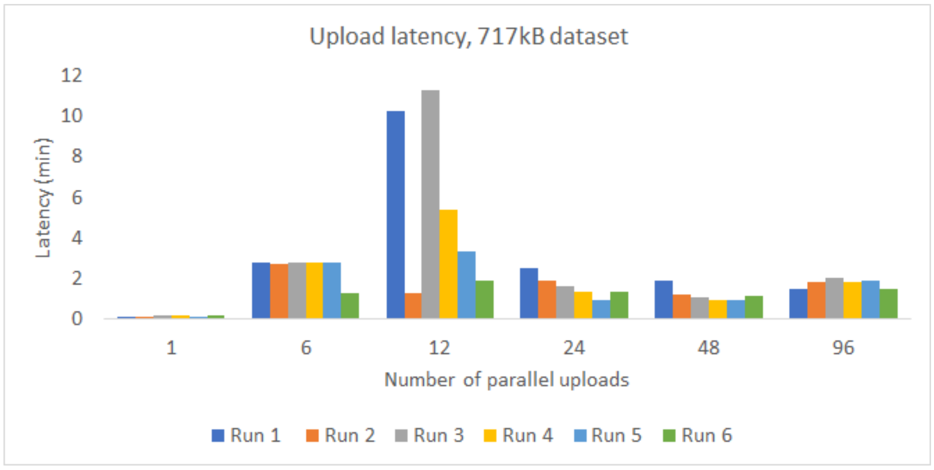


Figure 6.3: The upload time for each of the six benchmark runs for the 717kB dataset.

minimum upload time within the six benchmark runs. The upload latency of the 717kB dataset (blue line) is over-all low and acceptable. In table 6.1 we see how the backend managed to keep the latency low by adding more worker instances as the workload increased. Figure 6.3 shows the upload time for each of the six benchmark runs for each number of parallel uploads. They are generally small, except for when parallel uploading 12 data log files, at which the latency peaks and the upload time varies greatly from 1.8 - 11.3 minutes.

The orange line in figure 6.2 shows the upload latency for the 3.2MB dataset. The upload latency is low until we reach 12 parallel uploads, at which we reach max utilization of workers (table 6.2) and are unable to scale any more. This is, however, easily solved by increasing the number of maximum workers. Our only limit in terms of available instances is our quota on in-use addresses, i.e. available ip-addresses for our instances, which can be increased by requesting additional quota from GCP. The upload times for the six benchmark runs for each of number of parallel uploads (figure 6.4) show large variations, especially when uploading one file. While the first five runs completed in 30 seconds, the last one took 10 minutes to complete, which greatly affected the average upload latency. We believe it is caused by some files held up in the task queue for a longer period before being processed. We inspected the source code of the Google Cloud Pub/Sub adapter library[2] which enqueues and polls the task queue, but could not find the reason for the variation.

Since the majority of the data log files uploaded by the students are much smaller than the datasets generated for this evaluation, we believe that the

2. github.com/ursm/activejob-google_cloud_pubsub

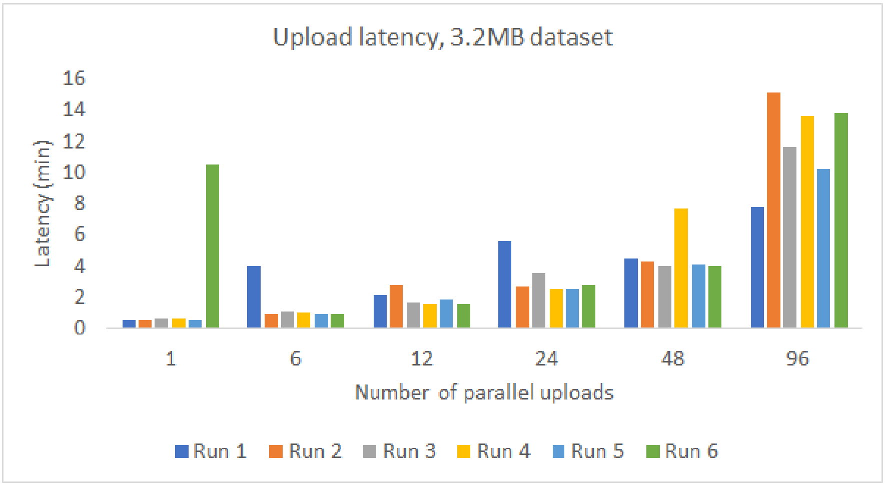


Figure 6.4: The upload time for each of the six benchmark runs for the 3.2MB dataset.

# parallel uploaded files	Number of workers
1	1-3
6	18
12	15
24	15
48	15
96	15

Table 6.2: The number of active worker instances during the six benchmark runs of the 3.2MB dataset. An increase in workers during the runs is denoted by a dash between the lowest and highest number of active instances.

upload latency experienced by the students is most comparable with the latency of the 717kB dataset. We cannot guarantee that a data log file will not be held up in the queue, thus variations in the upload latency will also apply for the students.

6.2 Data Query Latency

To evaluate the data query latency of the air:bit platform, we measure the time from the frontend web server receives the selected interface options from the air:bit web application (i.e when a student clicks the "search" button), until the data results are ready for visualization or download. To do this we generate six HTTP requests (table 6.3) to the frontend web server API, which will be translated and sent to the backend for data retrieval and processing. In addition to a time range, each request contains i) a parameter specifying if the

Visualization tool/ download	Area/ GPS	Path
Chart	GPS	http://airbit.uit.no/student?to=2018-04-29T11:39:48.000Z&from=2018-03-01T11:39:48.000Z&within=69.6747430330008,18.94003089717284,5.838591143813072&plotchart=true
Chart	Area	http://airbit.uit.no/student?to=2018-04-29T11:39:48.000Z&from=2018-03-01T11:39:48.000Z&area=Tromsø&plotchart=true
Map	GPS	http://airbit.uit.no/studentaqis?to=2018-04-29T11:39:48.000Z&from=2018-03-01T11:39:48.000Z&within=69.6747430330008,18.94003089717284,5.838591143813072&plotmap=true
Map	Area	http://airbit.uit.no/studentaqis?to=2018-04-29T11:39:48.000Z&from=2018-03-01T11:39:48.000Z&area=Tromsø&plotmap=true
Download	GPS	http://airbit.uit.no/student?to=2018-04-29T11:39:48.000Z&from=2018-03-01T11:39:48.000Z&within=69.6747430330008,18.94003089717284,5.838591143813072
Download	Area	http://airbit.uit.no/student?to=2018-04-29T11:39:48.000Z&from=2018-03-01T11:39:48.000Z&area=Tromsø

Table 6.3: The six data queries used to measure the air:bit platform query time.

backend should process data for visualization on map or visualization in charts. If parameter is absent, the data is meant for download; and ii) a parameter specifying if the student searches for data within a predefined area or has defined their own area by drawing a circle on the map. In case of a predefined area, the backend will use a database index to find the air:bit measurements mapped to the area. In case of a student-defined area, the backend will have to calculate distances between each air:bit measurement and the parameter value, to decide if it is within the defined circle.

For each query, we limit the number of elements processed by the backend (aggregated if required and converted to JSON) to 500, 1000, 5000, 10,000 and 50,000. Aggregating elements (grouping database entries by time or location and calculating averages for the air:bit sensor data values) and converting them to JSON are the most time-consuming parts of the data queries, and we therefore evaluate how the number of processed elements affects the query latency. We limit the number of processed entries by collecting all air:bit measurements that match the data query and selecting only the the first n entries of the group (n = 500, 1000, 5000, 10 000 or 50,000). We have configured the backend to scale to a maximum of 4 web instances, using a target CPU of 40%.

For each number of processed elements, we ran each query 20 times back-to-back in the following order:

1. Chart, GPS
2. Map, GPS

3. Download, GPS
4. Chart, area
5. Map, area
6. Download, area

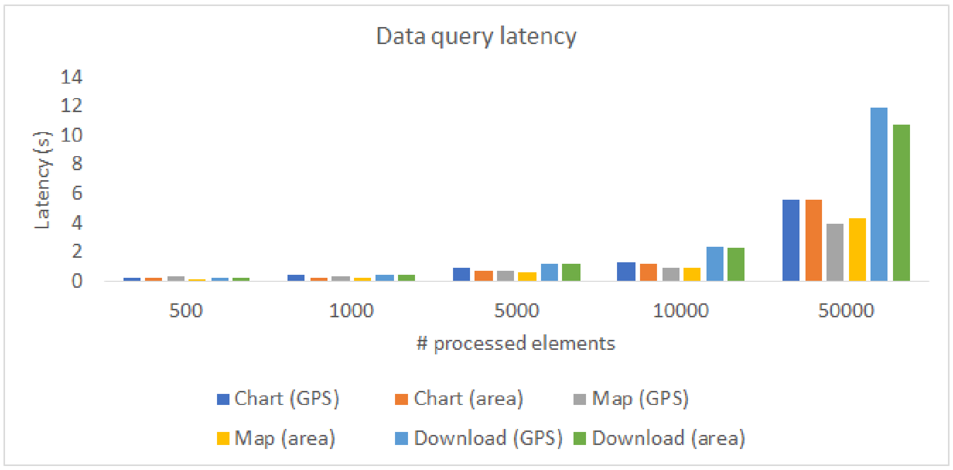


Figure 6.5: Average query time for each of the six queries.

Figure 6.5 shows the average query latency for each of the six queries. All queries have a latency of less than 1.5 seconds when processing up to 5000 elements. Passing 5000 elements, there is a significant increase in latency when downloading the data compared to visualizing it. This is, however, expected since the data is not aggregated, thus the number of elements that need to be converted into JSON is larger. There is also an increase in latency when visualizing the data in charts compared to on the map. This might be due to one retrieving more data points than the other, but we have not investigated this hypothesis. In some cases there are differences in the latency of querying data using in a predefined area compared to using GPS coordinates and a radius, but these are not consistent enough to draw any conclusions on which is faster.

We believe that a query latency of more than 2 seconds (except for when downloading data), is too much. Figure 6.6 shows the actual query latency at the backend in April (retrieved from the GCP Cloud Console tracing tool), and the majority of the requests have a higher latency than what we deem acceptable. An investigation of the requests that has produced highest latency show that they involve processing more than 330 000 elements. We must therefore, in future work, optimize queries as discussed in chapter 9.

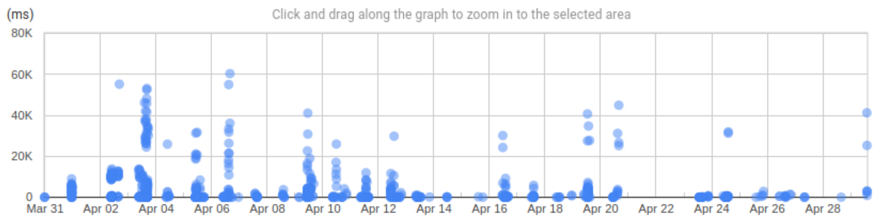


Figure 6.6: The actual student query latency in April 2018, retrieved from the GCP Cloud Console tracing tool.

	App Engine			Cloud SQL
	# virtual CPUs (vCPUs)	Core hours	GB hours	Core hours
February	2	1488	1488	659
March	2/3	1944	2964	743
April	3	2160	3600	720

Table 6.4: The monthly resource usage of the default App Engine instances and the Cloud SQL database instance from February through April 2018.

6.3 GCP Resource Usage

To evaluate the resource usage of hosting the backend on the Google Cloud Platform, we measure how much resources are needed to keep the backend able to serve requests, and process file uploads and data queries.

The backend runs three App Engine instances at all times, two web instances and one worker instance. The web instances serve HTTP requests and the worker instance is ready to process incoming data log files. The instances have one vCPU and 10GB persistent disk each. The web instances also have 2GB RAM, while the worker instance has 1GB RAM. The backend also continuously runs one instance for the Google Cloud SQL database. We use a db-n1-standard-1 instance, which and has one vCPU, 3.75 GB RAM and 25 GB SSD. 25 GB SSD is the minimum storage capacity for master instances.

Table 6.4 shows the monthly resource usage for the App Engine instances and the Cloud SQL database instance from February through April 2018. The number of App Engine vCPUs indicate the total number of virtual CPUs that run in App Engine at all times. We initially started out with one web instance and one worker instance with one vCPU each, but added a second web instance in mid-March for redundancy. The number of core hours describe number of hours the vCPUs have run combined (e.g. 2 vCPUs running for 1 hour equals 2 core hours). Similarly, the GB hours describe the number of gigabytes used in the course of an hour (e.g. 2 web instances with 2GB RAM each running for

Billing rates in USD, App Engine instances	
CPU	RAM
0.0579 / core hour	0.0078 / GB hour

Table 6.5: Billing rates for App Engine instances located in Belgium.

Billing rates in USD, Cloud SQL Instance (db-n1-standard-1)	
CPU (30% discount)	RAM
0.0676 / hour	0.17 / GB month

Table 6.6: Billing rates for the db-n1-standard-1 Cloud SQL instance.

1 hour, equals 4 GB hours). In table 6.4 we see how the second web instance created an increase in the number of App Engine core hours and GB hours. The increase in GB hours is also due to our increasing the web instances' RAM from 1GB to 2GB in March, to stop them from running out of memory during compute-intensive data queries. The increase in Cloud SQL core hours for March and April is due to the length, days, of each month.

6.4 GCP Operation Cost

The minimum GCP operation cost is the cost of hosting the backend on the GCP when no one is uploading or querying data, and no data is stored. Table 6.5 shows the billing rates of the App Engine instances in USD. Instances in the App Engine flexible environment are billed on a per-second basis with a one-minute minimum usage cost, using individual billing rates for VCPU usage and RAM [24]. The billing rate for vCPU in Belgium (where our instances are located) is $0.0579 US per core hour, while the billing rate for RAM is $0.0078 US per GB hour (May 2018). Table 6.6 shows the billing rates of the Cloud SQL database instance. Similar to the App Engine instances, the vCPU usage and RAM usage of the database instance is billed individually. The initial cost of the database vCPU is $0.0965/hour, but since the database instance runs continuously throughout the month, we are eligible for Google's Sustained Use Discounts[3] and get a 30% discount. For the 25 GB of database SSD, we are billed $0.17/GB.

We calculate the operation cost by multiplying the resources usage described in table 6.4 with the billing rates in this subchapter. Table 6.7 shows the monthly cost of the default App Engine instances and the Cloud SQL database from February through April 2018, both as separate services and combined. It is

3. cloud.google.com/compute/docs/sustained-use-discounts

	Cost in USD				
	App Engine		Cloud SQL		Total cost
	CPU	RAM	CPU	RAM	
February	86	12	44	4	146
March	113	23	50	4	190
April	125	28	49	4	206

Table 6.7: The monthly costs of the default App Engine instances and the Cloud SQL database instance from February through April 2018.

apparent that the majority of the operation costs is related to the App Engine instances' resource usage. The monthly operation cost of the backend in the Google Cloud Platform, i.e. the minimum cost of keeping the air:bit platform running online, is currently at about $206 US, which is acceptable for this service. The alternative would be to have a dedicated staff for IT operations, which would come to a higher cost.

6.5 GCP Scaling Resource Usage

To evaluate how the air:bit platform scales with regard to the number of data file uploads and queries, we investigate the backend's resource usage relative to the students data file uploads and queries so far.

To measure the resource usage for file uploads, we count the data log files that were processed by the platform from February through April and measure their size. The air:bit platform database shows that as of May 1st 2018, the students have uploaded in total 343 data log files, of which 222 were successfully parsed and inserted into the database. In terms of air:bit measurements, this comprises 481,186 entries in the WeatherData table. The remaining files have most likely been rejected due to wrongly formatted file contents. In table 6.8, showing the distribution of successfully uploaded files and and air:bit measurements per month from February through April, we see that the majority of data log files (66.6%) and air:bit measurements (84,3%) were uploaded during March. Figure 6.7 shows the distribution of the successfully uploaded data log files by size. We see that 62.6% of the data files are < 100kB and that 97.2% are < 500kB in size. A 100kB file contains about 1500 measurements.

Since the students decide for themselves the time interval at which their air:bits register data, the size of the data log file alone cannot say anything about the length of the measurement periods. Some 100kB files contain days of coarse-grained measurements, while other contain hours of fine-grained

	February	March	April	Total
Successfully parsed and inserted files	31	148	43	222
Successfully parsed and inserted measurements	5,522	405,648	70,016	481,186

Table 6.8: The number of successfully parsed and inserted data log files per month from February through April 2018.

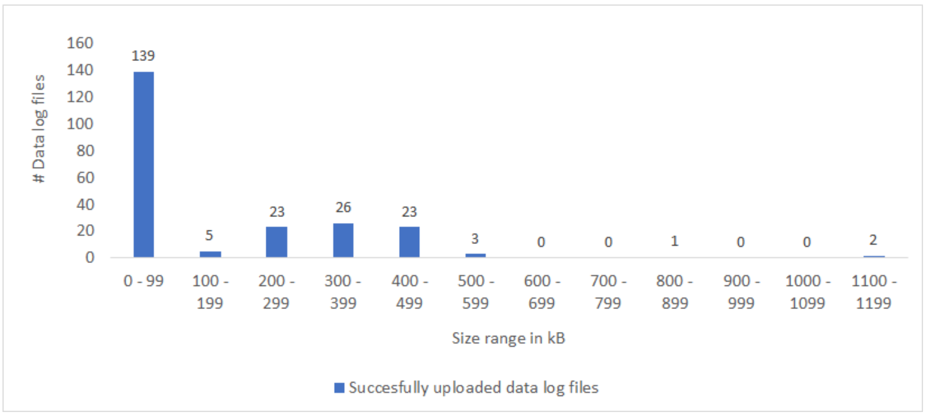


Figure 6.7: The size distribution of successfully parsed and inserted data log files per month from February through April 2018.

measurements.

Figure 6.8 shows the distribution of the 481,186 air:bit measurements in the database by area per 01.05.2018. The majority (69%) of the measurements are collected by the students in Bodø. This averages to 83,054 measurements per air:bit. In comparison, the students Tromsø, who have collected the second largest portion (16.6%) of the air:bit measurements, have an average number of 3,073 measurements per air:bit. The third largest portion of database entries are undefined, meaning they are not collected within an eight km radius of one of our predefined areas. Some entries have an undefined area due to latitude and longitude being wrongly formatted (e.g. as integers and not as floats). However, we also know that a portion of the undefined entries are due to the students in Mo i Rana collecting data in areas far away from Mo i Rana. To reduce the number of undefined entries, we could have increased the radius used to map measurements to an area. However, undefined entries are the reason why we implemented the radius circle search in the data exploration interface. Since the latency tests show that querying data using GPS coordinates is equally fast as (and sometimes even faster than) retrieving data using a predefined area, we do not consider the undefined entries to be a problem for now.

To measure the resource usage related to data queries, we count the number of queries processed by the backend. Unfortunately, we do not have data on

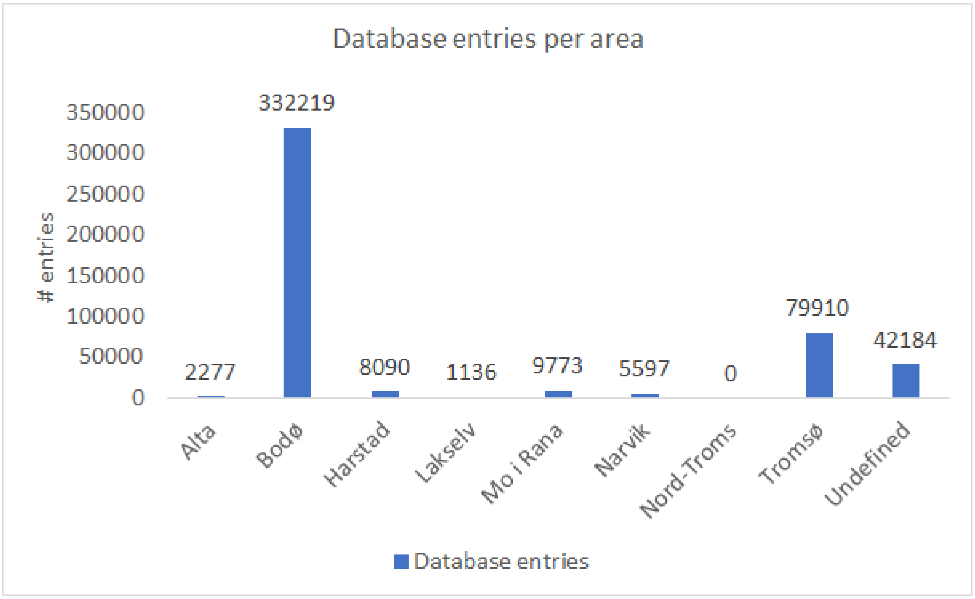


Figure 6.8: The distribution of air:bit database entries per area.

	Data uploads	Data queries
April	43	959

Table 6.9: Upload and data query statistics from April 2018.

the total number of processed queries during the course of the project, since we did not monitor the traffic to the backend using Google Analytics and the GCP monitoring tools only provide traces and logs for the past 6 weeks. We have the traces and logs from April 2018, and will base our evaluation of the scaled resource usage on these. Using the GCP Cloud Console tracing tool and the GCP monitoring service Google Stackdriver[4], we find that the backend processed 959 data queries in April. According to some teachers, their students started analyzing air quality data to answer their research question in April. We therefore believe that the number of processed student data queries is higher in April than in February and March.

Figure 6.9 shows the App Engine instance counts in April, i.e. the number of active App Engine instances run throughout the month and the resources used to process the 43 data log file uploads and 959 data queries in April (table 6.9). We see that the uploads and data queries caused periodically increases in the number of instances, meaning that the average CPU usage across either the web instances or worker instances periodically exceeded 40%. However, the average number of additional instances rarely exceeds one. This implies that the two

4. cloud.google.com/stackdriver/

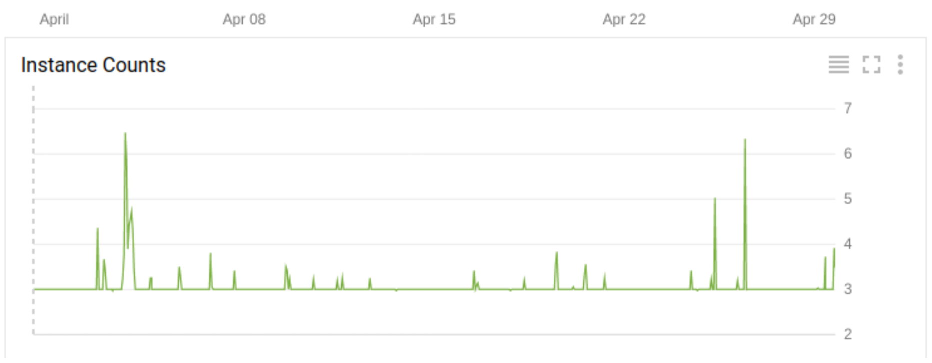


Figure 6.9: The App Engine instance counts in April, displaying the number of active App Engine instances run throughout the month.

		Core hours		
	Default vCPUs	Default instances	**Scaled instances**	Total
February	2	1488	**73**	1561
March	2/3	1944	**55**	1999
April	3	2160	**30**	2190

Table 6.10: The number of scaled App Engine core hours from February through April 2018.

default web instances and the one worker instance manage to cover much of the air:bit platform activities without starting additional instances. When the additional number of instances exceeds 1, it peaks at 4. We have configured the backend to scale to a maximum of 15 worker instances and 4 web instances. This tells us that, during April, the students' data log file uploads and data queries never used more than 36% of the available instances. This implies that the backend is provided with a sufficient amount of resources, but since we do not know which of the services scaled, we cannot say for sure.

To find the exact number of core hours and GB hours used by scaled instances from February through April, we examine the monthly billing transactions in the GCP Cloud Console and subtract the resource usage described in chapter 6.3. During the course of the project we have altered instances configurations and performed optimizations on the data retrieval methods, which have affected the usage of scaled core hours and scaled GB hours. We will come back to these changes and how they affect the scaled resource usage later in this subchapter.

Tables 6.10 and 6.11 show the total number of App Engine core hours and GB hours, as well as the distribution between default and scaled instances (highlighted in bold), from February through April 2018. Table 6.10 shows how

		GB hours		
	Default vCPUs	Default instances	**Scaled instances**	Total
February	2	1488	**73**	1561
March	2/3	2964	**109**	3073
April	3	3600	**57**	3657

Table 6.11: The number of scaled App Engine GB hours from February through April 2018.

the monthly scaled core hour usage has decreased each month, after peaking in February. February is the month in which we did the extensive performance tests of upload latency, during which we maximized the utilization of worker instances for a longer period of time. This resulted in a large number of scaled core hours. In the beginning of April, we optimized the methods that calculate the distance between two coordinates (used to collect measurements within a radius circle). This optimization might also have affected the number of scaled core hours in April.

Table 6.11 shows that the number of scaled GB hours peaks in March. However, since we do not know what types of instances were scaled, we do not know the exact cause. The students uploaded the most data log files in March and it may have caused the worker service to scale. However, since the amount of GB hours used by scaled instances is doubled the amount of scaled core hours, it seems like the majority of scaled instances were web instances, which implies that the air:bit platform experienced compute-intensive data queries. Due to the very uneven distribution of air:bit measurements per area (figure 6.8), some data queries involve processing several hundreds of thousands of elements. In those cases, the backend will have to scale the number of web instances to balance the workload.

Based on these findings, we know that the backend is able to scale with regard to the number of data file uploads and queries. For now we will not increase the maximum available instances for the worker service, but we should increase the number of available web instances in case of extra compute-intensive data queries.

6.6 GCP Scaling Cost

To evaluate the cost related to scaled resources, we subtract the baseline costs provided in chapter 6.4 from the total backend cost. The total monthly cost of the backend, as well as the costs of the App engine instances (CPU and RAM)

	Cost in USD		
	February	March	April
App Engine CPU	91	116	126
App Engine RAM	12	24	29
Cloud SQL	48	54	53
Other	1	2	3
Total platform cost	152	196	211

Table 6.12: Total backend cost.

	Cost related to scaled core hours in USD	Total core hour cost in USD
February	**4**	91
March	**3**	116
April	**2**	126

Table 6.13: Cost related to scaled core hours (highlighted in bold).

and the Cloud SQL database, are collected from the monthly billing transactions in the GCP Cloud Console and shown in table 6.12. We have also included a row for "other costs", which includes all smaller costs related to requests to and data retrieval from the storage bucket in Google Cloud Storage and storing data in Google Cloud Storage. For the remaining part of this chapter we will not discuss the cost of the Cloud SQL database, as it remains the same as in chapter 6.4 until we exceed the storage limit of 25GB.

Tables 6.13 and 6.14 show the monthly cost of the scaled core hour usage and GB hour usage (highlighted in bold) from February through April. The cost is calculated based on the numbers provided in tables 6.10 and 6.11 in chapter 6.5 and the billing rates in chapter 6.4. In tables 6.13 and 6.14 we also compare the scaled resource usage cost to the total resource usage cost, illustrating how the scaled instances are responsible for only 2-5% of the total core hour cost and 1-5% of the total GB hour cost. In table 6.15 we show the monthly total cost related to the scaled instances from February through April (highlighted in bold), and compare it to the monthly total cost of the backend. We have included the costs related to "other" in table 6.12 to the scaled costs, since they are generated by the students' data uploads and data queries. For now, costs related to the scaled resource usage are very small compared to the operation costs of the air:bit platform. This is because we only pay for the scaled resources we use and for now this usage is low.

	Cost related to scaled GB hours in USD	Total GB hour cost in USD
February	**1**	12
March	**1**	24
April	**1**	29

Table 6.14: Cost related to scaled GB hours (highlighted in bold).

	Total cost of scaled resources in USD	Total backend cost in USD
February	**6**	152
March	**6**	196
April	**6**	211

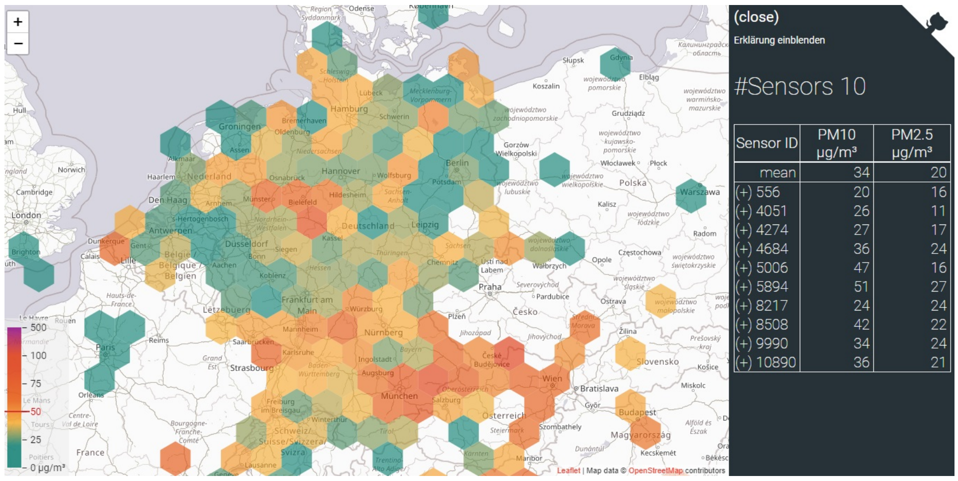
Table 6.15: Total cost of scaled resource usage (highlighted in bold).

7 Related work

There are many companies and recent projects that aim to create awareness about air pollution and to engage the public to collect air quality data using a citizen science approach using various air quality kits [25, 26, 27]. Some of the kits are 'out of the box' [28], meaning that they can be used immediately and no assembling or programming is required by the user. Others are based on a microcontroller/tiny computers and low-cost micro-sensors [29, 30, 31], and have to be built and/or programmed by the user before collecting data. Through online air pollution data platforms and mobile apps the projects or companies make the air quality data available to the public, enabling them to learn about - and reduce their exposure to - air pollution. There also exist frameworks for creating data collection tools, both with and without additional software development [7]. In this chapter we present some of these frameworks, projects and companies, as well as their data platforms and air quality sensor kits.

7.1 luftdaten.info

The German project luftdaten.info is a citizen science project focusing on particulate matter [26]. People around the world can participate by installing their self-built fine dust sensors outside their home, which collects and uploads



(close)
Erklärung einblenden

#Sensors 10

Sensor ID	PM10 µg/m³	PM2.5 µg/m³
mean	34	20
(+) 556	20	16
(+) 4051	26	11
(+) 4274	27	17
(+) 4684	36	24
(+) 5006	47	16
(+) 5894	51	27
(+) 8217	24	24
(+) 8508	42	22
(+) 9990	34	24
(+) 10890	36	21

Figure 7.1: The luftdaten.info particulate matter map. By clicking one of the coloured hexagons, a more detailed view (to the right) is provided.

data to the luftdaten.info site[1] over WiFi. Like our air:bits, their sensor kits measure dust particles using the Nova SDS011 and humidity and temperature using the DHT22, but they have use the NodeMCU ESP8266 microcontroller instead of an Arduino. At their website, they provide instructions on how to build the fine dust sensor and how to install the pre-programmed firmware, and they also provide continuously updated particulate matter map[2] based on the uploaded data is generated and displayed, shown by figure 7.1.

Unlike luftdaten.info, we have decided to store the air:bit data measurements to an on-board memory card, rather than uploading it over WiFi. This is mostly because it i) simplifies the software; and ii) makes it possible to use the kits without WiFi coverage. Since our aim is to introduce the students to computer science, and most of them (as well as the teachers) do not have any programming experience prior to this project, we want to keep the software as simple as possible. luftdaten.info also does not provide an interface or API for querying or downloading, which is one of the main services of the air:bit platform.

7.2 Hackair

The hackAIR project is supported through the EU programme on "Collective Awareness Platforms for Sustainability and Social Innovation" [25]. By enabling

1. luftdaten.info/
2. deutschland.maps.luftdaten.info

citizens and organizations to engage in generating and publishing outdoor air quality data, hackAIR aims to raise collective awareness about the daily levels of human exposure to air pollution. They have developed two versions of a home sensor[3] [4] that can be used to collect air quality data, and at their website they provide all the information (list of materials and tools, online retailers, etc) and instructions necessary to build one yourself. The first version only measures PM and is based on an Arduino Uno. For the second version, they have replaced the Arduino Uno with a Wemos D1 mini and added a sensor that measures temperature and humidity. Both home sensors use the same air quality sensor as our air:bit (the Nova SDS011) and the second version also use the same temperature and humidity sensor (DHT22). Similar to the air quality sensor used in Luftdaten.info, both hackAIR home sensors upload air quality data to the online hackAIR platform[5] over WiFi. hackAIR also provide the software for the sensors as a downloadable library, relieving the user from programming the sensor kit to collect data and communicate with the platform. They have, however, not open-sourced their platform. The platform integrates data from Luftdaten.info, as well as another open air quality and metadata platform powered by the open-source project OpenAQ[6], but they do not enable data querying or data download.

7.3 Plume Labs

Plume Labs is a French environmental technology company that aim to equip citizens around the world with information about their local air quality [32]. They have created the air pollution forecast app, Plume Air Report[7], which provide citizens with local air quality data (both historical and live) and a 24-hour air quality forecast. These forecasts are delivered by AI models, that learn from atmospheric data, satellite-based estimates, weather information and data from government-owned environmental monitoring stations [33, 34]. They also plan to add local, crowdsourced, sidewalk-level data to their data sources, when they launch their smart air quality tracker Flow in July 2018 (figure 7.2). The air quality tracker is said to measure real-time concentrations of NO_2, PM10, PM2.5 and VOC [27, 28], collecting both indoor and outdoor air quality data. It is available for pre-order for $139 and will have a final price of $199. As a part of a three year long test phase, Flow was taken to the streets of London by 100 beta testers in summer 2017 [35], but to our knowledge no specific test results have been published. We plan to test it once it becomes

3. hackair.eu/hackair-home/
4. hackair.eu/hackair-home-v2/
5. platform.hackair.eu/
6. openaq.org/
7. plumelabs.com/en/products/air-report

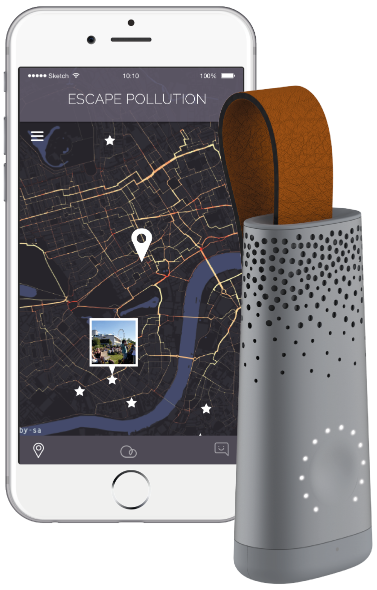


Figure 7.2: The Flow air quality tracker. Source: https://plumelabs.com/en/press.

available.

In August 2017 Plume launched the Plume API, opening up their air quality platform to third-party companies and organizations. Access to the API is tiered, with prices ranging from \$499 - \$2999 per month. However, academic researchers, non-governmental organizations and local community projects can apply for free access. We applied for access to the API in fall 2017, hoping to integrate their data with the student data on our platform, but have not yet gotten a response from Plume Labs and therefore have not been able to evaluate it.

7.4 ResearchKit and ResearchDroid

ResearchKit[8] is an open-sourced framework by Apple that researchers and developers can use to create apps for medical research [36]. It is designed for the IOS platform and makes use of the sensors and capabilities of iPhone (and Android when using the ResearchDroid[9] library) to record data, track

8. researchkit.org/
9. blog.appliedinformaticsinc.com/researchdroid-an-android-forms-and-consent-library/

movement and take measurements. The framework provides customizable modules helps creating surveys and creating active tasks (i.e. tasks in which iPhone sensors are used to actively collect data while the participants perform activities). We could have used this framework to create an app where the students could answer surveys to register e.g weather, as a supplementary means to the air:bit. Since iPhones (and smartphones in general) do not have all of the sensors that are necessary for this project (e.g. a particulate matter sensor) and we want the students to learn programming, we cannot replace the air:bits with a smartphone. We also do not want the students' ability to participate to depend on having a smartphone.

8 Conclusion

This thesis describes the air:bit platform, a scalable, cloud-based data management platform for citizen science education projects, that we designed and implemented in the air:bit project. Students use the air:bit platform to store, explore, visualize and download air quality data from ait:bits and other data air pollution related data sources.

The air:bit platform consists of three components: i) the air:bit web application, an interactive system for uploading, downloading and visual exploration of air quality data; ii) a frontend web server that translates user interactions in the air:bit web application into queries for storing air:bit data log files, retrieving air:bit data and integrating air quality data from external data sources; and iii) a backend that stores and provides air:bit data for the air:bit web application. We have deployed the backend to the Google Cloud Platform (GCP) to make the air:bit platform scalable in regard to computational resources and storage. This also makes the backend easier to maintain, since the GCP manages infrastructure and maintenance on our behalf.

We have evaluated the backend with regard to resource usage, latency, scalability and cost. The results show that the backend scales well as the workload increases and that our scaling configurations are adequate for the students' use of the air:bit platform. While the backend performs timely uploads of the students' air:bit data log files, the data query latency is too high. We suggests data query optimization strategies in chapter 9. The current monthly cost of the backend is about $211 US, which is acceptable for the service.

We have demonstrated that the air:bit platform successfully serves its purpose in the air:bit project. In the spring 2018 it was used by 174 students from 11 upper secondary (videregående) school classes across Northern Norway. Guided by the air:bit instructions in the air:bit web application, they successfully built and programmed 62 air:bits, which they used to collect data. From February through April 2018 they uploaded 222 air:bit data log files to the air:bit platform, comprising 481,186 air quality measurements.

We believe the air:bit platform is a valuable service, not only for the air:bit project, but also for other education projects. It relieves other air quality related citizen science projects from implementing their own data management solution. Being an open air quality database, it provides others with air quality data for analyses. We have also open-sourced the codebase for the air:bit platform, with the aspiration of guiding other who wish to create similar platforms for their citizen science education projects.

The air:bit web application is available at airbit.uit.no. We have open-sourced the code for the frontend service at github.com/fjukstad/luft and the code for the backend at github.com/ninaangelvik/luft.

9 Future Work

During the course of the air:bit project in 2018, the air:bit platform successfully provided students with services for storing and exploring air quality data from air:bits and NILU. Based on current knowledge of the use of the air:bit platform, we have identified areas of improvement that have not been implemented due to time constraints. We also present future features that we plan to add to the air:platform to improve the services provided to the students.

9.1 Data query optimization

The most urgent area of improvement for the air:bit platform is optimizing data queries. To do this, the first step would be to identify the bottlenecks of the backend using performance tools such as Bullet[1]. From there we would look into caching and alterations of the source code to make the caching most efficient. In chapter 6.2 we discussed how the number of processed elements in a data query heavily affects the latency. A quick way to achieve faster queries while waiting for a more permanent solution, would be to select a smaller randomized subset of elements from the query result set if it contains too many elements.

1. github.com/flyerhzm/bullet

9.2 Monitoring the backend

The evaluation process of the backend revealed the importance of adding monitoring to a system as soon as it becomes operative. Due to the lack of analytical data, we were not able to fully evaluate the actual upload and data query latency of the backend as experienced by the students. We will therefore add Google Analytics to the backend before the 2019 air:bit project.

9.3 Expand integration with external sources

An important feature of the air:bit platform is enabling students to query integrated data from external data sources. Currently, the only external data source that the students can query is NILU. We have implemented the functionality for retrieving precipitation data from MET, and are in the middle of implementing the functionality for visualizing the data. In the future we also aim to exchange with other data sources such as hackAIR. However, since they have not yet open-sourced their platform or exposed a platform API, a data exchange will require an integration process in close collaboration with the hackAIR team.

Bibliography

[1] The European Commission, *White Paper on the Future of Europe*. The European Commission, 2017.

[2] The European Commission, *Proposal for a Council Recommendation on Key Competences for Lifelong Learning*. The European Commission, 2018.

[3] European Schoolnet, *Computer programming and coding*. European Schoolnet, 2015.

[4] B. Fjukstad, N. Angelvik, M. W. Hauglann, J. S. Knutsen, M. Grønnesby, H. Gunhildrud, and L. A. Bongo, "Low-cost programmable air quality sensor kits in science education," in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, SIGCSE '18, (New York, NY, USA), pp. 227–232, ACM, 2018.

[5] E. Law, K. Z. Gajos, A. Wiggins, M. L. Gray, and A. Williams, "Crowdsourcing as a tool for research: Implications of uncertainty," in *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*, CSCW '17, (New York, NY, USA), pp. 1544–1561, ACM, 2017.

[6] Azavea and SciStarter, *Citizen Science Data Factory. A Distributed Data Collection Platform for Citizen Science*. Azavea and SciStarter, 2014.

[7] Azavea and SciStarter, *Citizen Science Data Factory. A Distributed Data Collection Platform for Citizen Science*. Azavea and SciStarter, 2014.

[8] Guerreiro, Christina and de Leeuw, Frank amd Foltescu, Valentin, *Air Quality in Europe - 2013 report*. 2013.

[9] WHO, *World Health Assembly closes, passing resolutions on air pollution and epilepsy*. 2015.

[10] R. Beelen, O. Raaschou-Nielsen, M. Stafoggia, Z. J. Andersen, G. Weinmayr,

B. Hoffmann, K. Wolf, E. Samoli, P. Fischer, M. Nieuwenhuijsen, P. Vineis, W. W. Xun, K. Katsouyanni, K. Dimakopoulou, A. Oudin, B. Forsberg, L. Modig, A. S. Havulinna, T. Lanki, A. Turunen, B. Oftedal, W. Nystad, P. Nafstad, U. D. Faire, N. L. Pedersen, C.-G. Östenson, L. Fratiglioni, J. Penell, M. Korek, G. Pershagen, K. T. Eriksen, K. Overvad, T. Ellermann, M. Eeftens, P. H. Peeters, K. Meliefste, M. Wang, B. B. de Mesquita, D. Sugiri, U. Krämer, J. Heinrich, K. de Hoogh, T. Key, A. Peters, R. Hampel, H. Concin, G. Nagel, A. Ineichen, E. Schaffner, N. Probst-Hensch, N. Künzli, C. Schindler, T. Schikowski, M. Adam, H. Phuleria, A. Vilier, F. Clavel-Chapelon, C. Declercq, S. Grioni, V. Krogh, M.-Y. Tsai, F. Ricceri, C. Sacerdote, C. Galassi, E. Migliore, A. Ranzi, G. Cesaroni, C. Badaloni, F. Forastiere, I. Tamayo, P. Amiano, M. Dorronsoro, M. Katsoulis, A. Trichopoulou, B. Brunekreef, and G. Hoek, "Effects of long-term exposure to air pollution on natural-cause mortality: an analysis of 22 european cohorts within the multicentre escape project," *The Lancet*, vol. 383, no. 9919, pp. 785 – 795, 2014.

[11] R. D. Brook, S. Rajagopalan, C. A. Pope, J. R. Brook, A. Bhatnagar, A. V. Diez-Roux, F. Holguin, Y. Hong, R. V. Luepker, M. A. Mittleman, A. Peters, D. Siscovick, S. C. Smith, L. Whitsel, and J. D. Kaufman, "Particulate matter air pollution and cardiovascular disease," *Circulation*, vol. 121, no. 21, pp. 2331–2378, 2010.

[12] O. Raaschou-Nielsen, Z. J. Andersen, R. Beelen, E. Samoli, M. Stafoggia, G. Weinmayr, B. Hoffmann, P. Fischer, M. J. Nieuwenhuijsen, B. Brunekreef, W. W. Xun, K. Katsouyanni, K. Dimakopoulou, J. Sommar, B. Forsberg, L. Modig, A. Oudin, B. Oftedal, P. E. Schwarze, P. Nafstad, U. D. Faire, N. L. Pedersen, C.-G. Östenson, L. Fratiglioni, J. Penell, M. Korek, G. Pershagen, K. T. Eriksen, M. Sørensen, A. Tjønneland, T. Ellermann, M. Eeftens, P. H. Peeters, K. Meliefste, M. Wang, B. B. de Mesquita, T. J. Key, K. de Hoogh, H. Concin, G. Nagel, A. Vilier, S. Grioni, V. Krogh, M.-Y. Tsai, F. Ricceri, C. Sacerdote, C. Galassi, E. Migliore, A. Ranzi, G. Cesaroni, C. Badaloni, F. Forastiere, I. Tamayo, P. Amiano, M. Dorronsoro, A. Trichopoulou, C. Bamia, P. Vineis, and G. Hoek, "Air pollution and lung cancer incidence in 17 european cohorts: prospective analyses from the european study of cohorts for air pollution effects (escape)," *The Lancet Oncology*, vol. 14, no. 9, pp. 813 – 822, 2013.

[13] M. Pascal, M. Corso, O. Chanel, C. Declercq, C. Badaloni, G. Cesaroni, S. Henschel, K. Meister, D. Haluza, P. Martin-Olmedo, and S. Medina, "Assessing the public health impacts of urban air pollution in 25 european cities: Results of the aphekom project," *Science of The Total Environment*, vol. 449, pp. 390 – 400, 2013.

[14] P. S. A. B. S. M. P. Hagen, JA; Nafstad, "Associations between outdoor air pollutants and hospitalization for respiratory diseases," *Epidemiology*, vol. 11, pp. 136 – 140, 2000.

[15] EPA, "Health and Environmental Effects of Particulate Matter (PM)." `https://www.epa.gov/pm-pollution/health-and-environmental-effects-particulate-matter-pm`. [Online; Accessed: 2018-04-16].

[16] Angelvik, Nina, "Low-cost portable air quality sensor kit and cloud service.".

[17] S. Newman, *Building Microservices*. O'Reilly Media, Inc., 1st ed., 2015.

[18] Google Cloud Platform, "How Instances are Managed." `https://cloud.google.com/appengine/docs/standard/java/how-instances-are-managed`. [Online; Accessed: 2018-04-17].

[19] Google Cloud Platform, "Using Cloud Storage with Ruby." `https://cloud.google.com/ruby/getting-started/using-cloud-storage`. [Online; Accessed: 2018-04-17].

[20] Google Cloud Platform, "Storage Classes." `https://cloud.google.com/storage/docs/storage-classes`. [Online; Accessed: 2018-04-17].

[21] S. Muralidhar, W. Lloyd, S. Roy, C. Hill, E. Lin, W. Liu, S. Pan, S. Shankar, V. Sivakumar, L. Tang, and S. Kumar, "F4: Facebook's warm blob storage system," in *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, (Berkeley, CA, USA), pp. 383–398, USENIX Association, 2014.

[22] Google Cloud Platform, "What Is Cloud Pub/Sub?." `https://cloud.google.com/pubsub/docs/overview`. [Online; Accessed: 2018-04-17].

[23] Google Cloud Platform, "Cloud SQL." `https://cloud.google.com/sql/`. [Online; Accessed: 2018-04-17].

[24] Google Cloud Platform, "App Engine Pricing." `https://cloud.google.com/appengine/pricing#flexible-environment-instances`. [Online; Accessed: 2018-05-01].

[25] hackAIR, "About hackAIR." `http://www.hackair.eu/about-hackair/`. [Online; Accessed: 2018-03-22].

[26] luftdaten.info, "Home." `https://luftdaten.info/en/home-en/`. [Online;

Accessed: 2018-03-22].

[27] Plume Labs, “Meet Flow, your smart mobile air quality tracker.” `https://blog.plumelabs.com/2017/01/03/meet-flow-your-smart-mobile-air-quality-tracker/`. [Online; Accessed: 2018-03-29].

[28] Plume Labs, “Flow.” `https://flow.plumelabs.com/`. [Online; Accessed: 2018-03-29].

[29] hackAIR, “hackAIR home v1 sensor.” `http://www.hackair.eu/hackair-home/`. [Online; Accessed: 2018-03-22].

[30] hackAIR, “hackAIR home v2 sensor.” `http://www.hackair.eu/hackair-home-v2/`. [Online; Accessed: 2018-03-22].

[31] luftdaten.info, “Fine dust sensor - construction manual.” `https://luftdaten.info/en/construction-manual/`. [Online; Accessed: 2018-03-22].

[32] Plume Labs, “About Us.” `https://blog.plumelabs.com/about-us/` . [Online; Accessed: 2018-03-28].

[33] Plume Labs, “Clean air—now there’s an API for that.” `https://blog.plumelabs.com/2017/08/01/clean-air-now-theres-an-api-for-that/`. [Online; Accessed: 2018-03-28].

[34] Plume Labs, “Frequently Asked Questions (FAQ).” `https://plume.io/en/faq`. [Online; Accessed: 2018-03-28].

[35] Plume Labs, “Flow pre-orders now open.” `https://blog.plumelabs.com/2017/09/26/flow-pre-orders-now-open/`. [Online; Accessed: 2018-04-17].

[36] ResearchKit, “ResearchKit Framework Programming Guide.” `http://researchkit.org/docs/docs/Overview/GuideOverview.html`. [Online; Accessed: 2018-04-16].

Appendices

A
air:bit Poster

This appendix includes a resized version of the poster presented at SIGCSE2018.

The full-size version is available at
github.com/ninaangelvik/sigcse2018/blob/master/poster_nano16.pdf

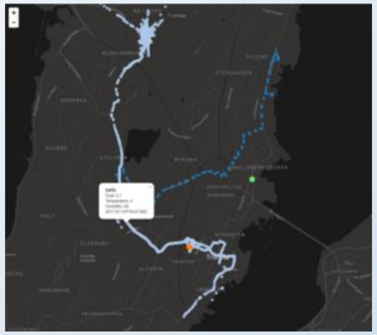
UiT
THE ARCTIC
UNIVERSITY
OF NORWAY

Air pollution data analysis platform for computer science education projects

Nina Angelvik
nan016@post.uit.no
Department of Computer Science, UiT – The Arctic University of Norway

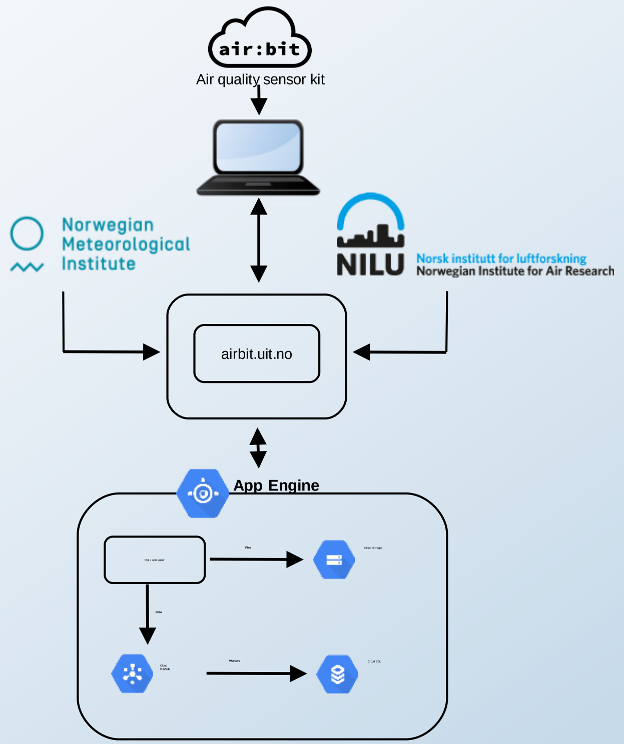
Students build and code their own air quality sensor kits

The air:bit platform is used in science education projects where upper secondary school students build and code their own air quality sensor kits, the air:bit, before investigating a research question by analyzing their collected data. The air:bit periodically registers time, location, temperature, humidity and dust particles, and stores the data as a CSV file on an on-board memory card.



Web interface for analysis of localized air quality patterns

We have built a web application where the students upload, download, query and visualize their data. It also integrates the student data with climate data from external sources, such as The Norwegian Institute for Air Research and the Norwegian Meteorological Institute.


air:bit
Air quality sensor kit
Norwegian Meteorological Institute
NILU Norsk institutt for luftforskning Norwegian Institute for Air Research
airbit.uit.no
App Engine

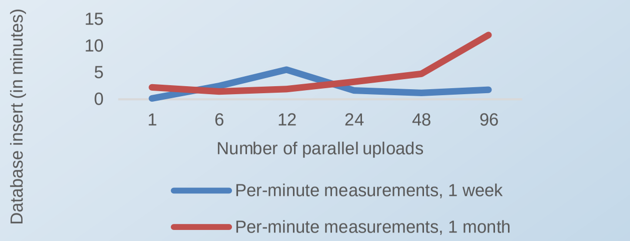
Scalable cloud based data analysis platform

The uploaded data files and web application queries are sent to our backend using HTTP REST API. The backend is deployed to the App Engine Flexible Environment, a Platform as a Service delivered by the Google Cloud Platform (GCP). It is implemented in Ruby on Rails and consists of multiple GCP services, including Google Cloud Storage, Google Cloud Pub/Sub and Google Cloud SQL. Upon receipt, the file is stored as an entity in Cloud Storage, before its content is enqueued in the Cloud Pub/Sub task queue. In an asynchronous manner, the queue is polled and the data is parsed and inserted into the Cloud SQL database. The client will receive a response message once the file has been successfully/unsuccessfully added to the queue, while query results are returned to the web application as JSON objects.

Visit our platform at: http://airbit.uit.no

ACKNOWLEDGEMENTS
A big thank you to the Department of Computer Science at UiT – the Arctic University of Norway and the Centre for Science Education at UiT – The Arctic University of Norway for funding this poster presentation.

Database insert times for parallel uploads of data logs



Preliminary performance results and cost estimates

The App Engine Flexible Environment automatically scales the platform up and down based on a configurated CPU target. To test the scalability of the platform we have measured the data insert time for uploading a variable number of data files containing 1 week of per-minute measurements (10080 measurements, 717kB) and 1 month of per-minute measurements (44640 measurements, 3.2 MB). Initial performance results show that the average insert time for up to 96 parallel uploads is less than 6 minutes for files containing 1 week of data and 14 minutes for files containing 1 month of data. This is acceptable for our usage.

Our platform is deployed to virtual machines that are billed on a per-second basis. The main cost are the three vCPUs for the web server, a background worker and the database. The cost of the three vCPUs is ~$115/month. The extra cost of handling bursts in uploads using additional vCPUs is low. For example the cost of using 15 vCPUs for 96 3.2 MB file uploads is $0.25. Including the cost of memory, persistent disk and the occasional use of the extra vCPUs, the total cost is about $135/month.

11 schools in Northern Norway are using the platform

During spring 2018, 11 school classes in Northern Norway are using the platform to study air quality patterns in their local environment.

/B

air:bit Paper

Low-Cost Programmable Air Quality Sensor Kits in Science Education

Bjørn Fjukstad
Department of Computer Science
UiT The Arctic University of Norway

Nina Angelvik
Department of Computer Science
UiT The Arctic University of Norway

Maria Wulff Hauglann
Department of Computer Science
UiT The Arctic University of Norway

Joachim Sveia Knutsen
Kongsbakken Videregående Skole

Morten Grønnesby
Department of Molecular Biology
UiT The Arctic University of Norway

Hedinn Gunhildrud
Science Centre of Northern Norway

Lars Ailo Bongo
Department of Computer Science
UiT The Arctic University of Norway

ABSTRACT

We describe our citizen science approach and technologies designed to introduce students in upper secondary schools to computational thinking and engineering. Using an Arduino microcontroller and low-cost sensors we have developed the *air:bit*, a programmable sensor kit that students build and program to collect air quality data. In our course, students develop their own research questions regarding air quality before using their own air quality sensor kit to answer their respective questions. This project combines electronics and coding with natural sciences providing a truly interdisciplinary course.

We have open-sourced the teaching materials including the building and coding instructions. In addition, students can contribute to our web-based platform for storing, visualizing, and exploring the collected air quality data. It also provides an open API for anyone to download air quality data collected by the students. Through the website, available at airbit.uit.no, students are motivated to contribute air quality data open to the public.

We describe lessons learned from our pilot project in a Norwegian upper secondary school and how we are deploying it in 10 schools across Northern Norway. In the pilot, students successfully built and coded the air:bits, and after two months of data collection they could correctly describe local patterns in the air quality. We believe that by combining electronics and coding with the natural sciences we motivate students to engage in all scientific disciplines.

ACM Reference Format:
Bjørn Fjukstad, Nina Angelvik, Maria Wulff Hauglann, Joachim Sveia Knutsen, Morten Grønnesby, Hedinn Gunhildrud, and Lars Ailo Bongo. 2018. Low-Cost Programmable Air Quality Sensor Kits in Science Education. In *SIGCSE '18: The 49th ACM Technical Symposium on Computer Science Education, Feb. 21–24, 2018, Baltimore, MD, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3159450.3159569

INTRODUCTION

From a recently published Norwegian Official Report on the future of education in Norway, the authors argue that technology affects all subjects in the Norwegian school system and that digital skills should be expressed in these.[1] While countries such as the UK, Finland, and Estonia have already, or are now, introducing programming and computational thinking in school curricula from primary school and upwards[2], Norway is falling behind. This is both due to the lack of mandatory courses, teaching materials, tools, and courses for teachers without any programming background.

We have developed a course that aims to introduce computer programming and engineering to Norwegian upper secondary schools. Our course combines engineering and computer programming with natural sciences. We developed a maker-inspired citizen science approach focused on air pollution data collection monitoring. We believe this engages both male and female students that are not primarily interested in programming and engineering. It also teaches scientific thinking since students themselves have to collect and verify data. As part of the course we developed an air pollution sensor kit that we have named *air:bit* to collect data. We provide teaching materials including the design, assembly instructions, example source code for the sensors, and a cloud based service for students to upload and explore their collected datasets.

The air:bit shown in Figure 1 measures air quality using low-cost sensors and the Arduino UNO microcontroller.[1] It measures i) dust particles, ii) temperature, iii) air humidity, iv) location, v) time and date, and log these to a memory card in an open format. The total cost of each kit is $40 USD. The cloud service provides data storage and visualization that can be used by students and the public to view current and previous trends in air quality. The cloud service also facilitates download of the air quality datasets through an open API.

The rest of this paper gives an introduction to air quality and related projects that aim to create awareness and engagement; describe our course and how it is structured; gives an overview of

[1] arduino.cc

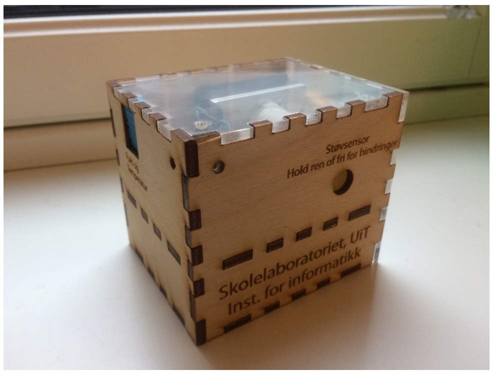


Figure 1: The air:bit.

the air:bit and the computational resources; discuss our experiences from a pilot project; and lastly we provide future directions for the project.

BACKGROUND

We provide a background in air quality and describe related projects that aim to create public awareness and information on air quality.

Air pollution

Air pollution is a global issue since it reduces quality of life in polluted areas and causes diseases. The WHO has termed air pollution as the largest single environmental health risk[3], and has both health, environmental and climate effects.[4] Both short and long term exposure to poor air quality as a result of air pollution is contributing to respiratory disease, cardiovascular disease, and certain cancers.[4–8] In Norway, both European and Norwegian legislation ensure that the air quality is monitored and that air quality forecasts are available to the public. Air pollution originate from a range of different sources. From chemical emissions in factories, chemicals used as a fertilizer in agriculture, exhaust from combustion engines burning fossil fuel, to micro-particles from cars driving on snow-free roads with studded tires[4, 9]. This wide range of sources generate different pollutants, e.g. nitrogen oxides (NO_x), Ozone (O_3), Particulate Matter (PM), and Carbon Monoxide (CO).

In Northern Norway the air quality is rapidly changing in the winter months, especially while the seasons are changing from winter to spring, and fall to winter mainly due to the use of spiked tires on dry roads that generate dust particles in the air, and the emissions from diesel powered cars. Reducing the use of cars will improve air quality in these months, and by creating awareness on the local conditions and their impact on health we believe that it is possible to reduce car usage without enforcing it. However, the monitoring is typically done using stationary equipment that provides high-quality data, but due to their high cost are not affordable to locate throughout all populated areas. Therefore citizens may not find available data on the air quality in their city or neighborhood and it's not possible to increase awareness around poor air quality and simple measures to improve it.

Recent initiatives to use low-cost, easy-to-use micro-sensors for air quality monitoring[10–13] to replace the expensive instruments currently to gather high-quality air quality data. These projects are often targeted towards crowd sourcing air quality data, and must therefore provide an easy-to-use platform that anyone can use. In our project we focus on building the sensor kits to give the students a deeper understanding on how the technology works and not just how to use it. We have focused on measuring PM since it is the major cause of poor air quality in Tromsø, and there are simple and reasonably priced sensors available to measure dust densities in the air.

Air Pollution Engagement Projects

There are many recent projects that aim to engage the public to collect air quality data using a citizen science approach using various air quality kits.

Friskby Bergen is a Norwegian project with a similar aim as our project to develop a project for school classes to build air quality monitoring base stations and collect air quality data. Their technology is based on the Raspberry Pi computer and use low-cost sensors to measure air quality. Their monitoring stations are stationary and log data directly to their open web services over wifi. While the Friskby sensor kits provide live monitoring of air quality by directly uploading air quality measurement over wifi, our air:bit is equipped with a memory card to store measurements before being uploaded by the students afterwards. This simplifies both the software and makes it possible to use the kits without any wifi coverage.

The aim of the hackAIR project is to develop a simple DIY kits that enable citizens and organizations to engage in air quality data generation and sharing.[14] We share the same technical platform as the hackAIR project, and we have initiated a collaboration to share both data and experiences between the two projects.

The CITI-SENSE project has developed 'citizens observatories' for citizens to contribute to and participate in environmental governance[11]. Citizens use the Little Environmental Observatory (LEO) sensor packs to measure NO, NO_2, O_3, temperature and relative humidity. Data is uploaded to a online service for users to view and explore, similar to our approach.

Another project that share our technological platform is the Luftdaten project which aims to promote air quality awareness in Germany.[15] Through the luftdaten.info site they provide instructions on how to build and code the air quality sensor, in addition to interactive maps with live data. The Luftdaten project also host frequent meetups for citizens who need help with the air quality kits or want to learn more about the project. We share the approach with the Luftdaten project, but we target high-school students as the main audience.

Computer Science Education

There is a wealth of platforms and software tools to introduce programming and technology. One such approach is to combine electronics and coding through the cheap Arduino microcontroller architecture and electronic components such as LEDs, buttons or other sensors.[16, 17] An example of one such approach was to create or hack existing toys or build new ones to make "noise"[18].

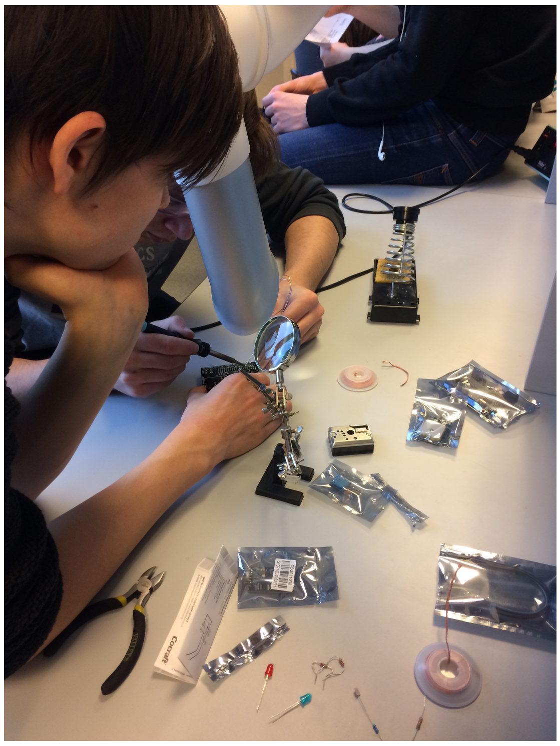


Figure 2: Students soldering different components of the air quality sensor kits to the custom PCB Arduino Shield.

Another is to use Arduinos in Physics experiments in the areas of Optics, Thermodynamics, and Waves.[19]

COURSE CONTENTS

Our course has been given once in spring 2017 at UiT The Arctic University of Norway to science students at a local High School, and is planned again in spring 2018 across 10 upper secondary schools across Northern Norway. We host the course at the university both to recruit new students but also as a part of the outreach program at the university.

To make the course fit into different subjects in the upper secondary school in Norway, we have surveyed the relevant subjects and their specific learning goals and requirements. By creating a project where students build, code and use a sensor kit to investigate air pollution we cover learning outcomes from the subjects Technology and Research Learning (Teknologi og Forskningslære), Physics (Fysikk), Chemistry (Kjemi), Information Technology (Informasjonsteknologi), and Mathematics (Matematikk for realfag).

The course is run over the duration of a semester and given in four segments; first an introduction to air quality and research on the topic; a hands-on introduction to electronics and coding, as well as assembling and coding the air:bit; air quality data collection; and finally an evaluation where students summarize their work in a written report and/or a presentation. We followed the students' presentations both to get an overview of their learning outcome and get feedback on the project.

Figure 3: Students browsing their collected air quality data on the web site.

The course is designed for students in Norwegian upper secondary school, typically around 17-18 years old. Students may have some experience with basic circuits and electronics through science classes, but since there are no mandatory courses on computer programming so we do not expect any prior experience with coding. We can expect them to have basic computer skills and access to individual laptops.

As for the teaching staff at the individual institutions we cannot expect them to have more knowledge on air quality, microcontrollers, or extensive coding experience. Prior to the project we therefore invite teachers for an intensive two-day workshop that take the teachers through the four segments of the course. We host this two-day workshop at our department.

Following the two-day workshop teachers will return to their schools with parts and instructions to complete the course at their respective institutions. We provide an online forum for questions and answers, both from students and teachers, if they encounter any difficulties with assembling, coding or collecting data. The project is typically run in the spring months to capture the changing air quality from winter to spring.

In the course, teachers can themselves choose if they want to provide research questions to the students or if they want students to develop these themself. The imporant point is that the students all contribute to a large database with air quality measurements, but each student group each decide on what aspect of air pollution they want to investigate. For example, one student group investigated the relationship between snow coverage of roads and dust particles in the air.

SENSOR KIT AND CLOUD COMPUTE INFRASTRUCTURE

In this section we describe our air quality measurement kit, its relevant documentation, the backend storage system, and the frontend visualization platform. The students collect air quality data and upload it to the backend storage system. To view and download data, users access a frontend web application. This application interfaces with both student-collected data in our local backend storage system, as well as open air quality data from the Norwegian

Table 1: A list of the different components in the air:bit along with their cost (as of August 2017).

Component	Cost (USD)
Arduino Uno microcontroller	$3.14
NEO6MV2 GPS module	$8.19
Sharp GP2Y1010AUOF optical dust sensor	$5.99
DHT11 temperature and humidity sensor	$1.00
SD Card reader and 16GB memory card	$4.74
Portable power bank	$15.00
Two indicator leds and resistors	$1.00
Custom PCB circuit board	$2.00
Total:	$41.06

Institute of Air Research (NILU) and the Norwegian Meteorological Institute (MET).

air:bit

We have designed the air quality sensor kit as a small microcontroller based data logger that collects measurements of dust particles, air temperature, air humidity, location, and time and date. The kit is enclosed in a laser cut box, equipped with an external power source that makes the kit portable and withstanding of different weather conditions. Figure 1 shows the first prototype. The kit was built as simple as possible to facilitate use in an educational setting. Table 1 lists the different components and their respective cost.

To simplify the assembly and soldering of the components to the microcontroller we have designed a custom PCB circuit board. The circuit board has pre-defined pins for each sensor, and fits on top of the popular Arduino UNO board. Figure 4a shows the underlying circuit and Figure 4b shows the custom designed PCB circuit board.

Students assemble the kit soldering the components to the custom PCB circuit board. The sensors and circuit board is then mounted on top of an Arduino UNO before enclosed in the pre-cut box.

To program the air:bit we use the standard Arduino IDE together with additional libraries to interface with the different sensors. Through the Arduino IDE students code and upload programs to the Arduino. Arduino programs are written in C++ and by the end of the project students will end up with a working solution at about 150 lines of code. While this is the final version, we expect students to write at least 500 lines of code during the project to test sensors and experiment with different solutions. The Arduino IDE features a console for monitoring communication from the Arduino, allowing students to read data in real time from the different sensors before taking it outside.

Since the kits are programmed in upper secondary school classes where both teachers and students have little or no coding experience, we aim to keep the code as simple as possible. In the project we distribute example code to interface and collect data from the individual sensors[2]. These are small $\tilde{1}00$ lines of code examples that typically take 20 minutes each to implement. The students must write their own program that collects data from all sensors simultaneously and write them to a memory card. We do not put any restrictions on how their code should look like, the only restriction is on the format of the output data written to the memory card.

We designed a simple data exchange format for the air:bit based on the simple CSV file format. Sensor kits create a single log file and append to it as it collects new measurements. A line (row) in the output file is an observation and consist of measurements for all available sensors. Listing 2 shows an example data file. Using simple Arduino libraries students write the data files to memory cards and they can view them using standard applications, such as Excel, on their laptops.

Listing 1: A simplified code example to collect and print temperature and humidity data from a DHT sensor every second.

```
void setup() {
  Serial.begin(9600); // Start Serial communication to
  dht.begin();        // receive messages from the Arduino
}                     // and initialize the DHT sensor.

void loop() {
    // Collect data and print them.
    float humidity = dht.readHumidity();
    float temperature = dht.readTemperature();
    Serial.print(temperature);
    Serial.print(", ");
    Serial.print(humidity);
    Serial.print("\n");
    delay(1000);
}
```

Listing 2: Example data file. Every line in the file is a measurement.

```
Time and date, Latitude, Longitude, Dust, Temperature, Humidity
26/10/2016 18:48:41, 69.682121, 18.978985, 88.98, 21.00, 20.00
26/10/2016 18:48:46, 69.682114, 18.978952, 95.70, 22.00, 17.00
26/10/2016 18:48:51, 69.682114, 18.978891, 99.06, 22.00, 17.00
26/10/2016 18:48:55, 69.682106, 18.978865, 98.22, 22.00, 17.00
                                ...
```

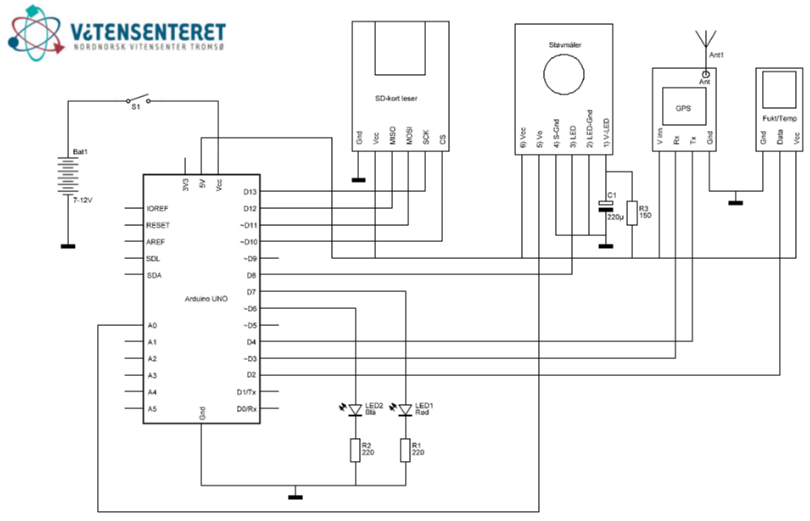
Backend data storage

We built a backend cloud based data storage system to handle the large quantities of student data. Students upload data directly from their memory cards to the backed system. The backend also serves queries for air quality data within given time intervals. Since we want to support a query-like interface to the collected air quality data, we used a relational database to build the backend storage system. Student access a lightweight web interface to upload data files from the air:bits, which are parsed and inserted into the database represented by a record for each measurement. Records are indexed using a combination of the time and date, and location. Invalid records are rejected and users will receive an error message to indicate any invalid data or formatting. The backend data storage system exposes a small REST API to allow the frontend (and other applications) to retrieve air quality data. The API will accept queries to retrieve all data within a given time interval. There is no limitation on the length of the time interval, and the API will return a single file with all measurements within the time period.

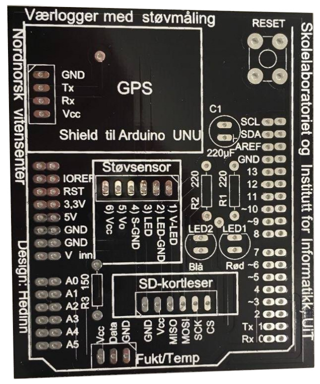
Frontend visualization

To simplify the process of accessing the collected air quality data, we built a web application. Students and the public can explore air quality data from the last 24 hours, or view and download data from any time period. The web application is built around a

[2]Available online at airbit.uit.no

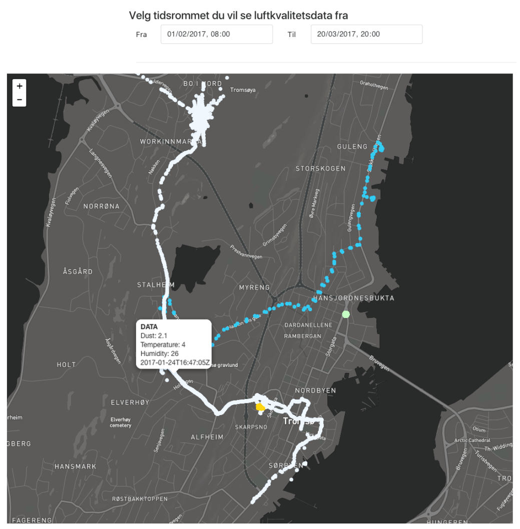


(a) The Air Quality Sensor Kit circuit diagram.



(b) The custom PCB Arduino shield.

Figure 4: air:bit circuit diagram and custom PCB.


Velg tidsrommet du vil se luftkvalitetsdata fra

Fra 01/02/2017, 08:00 Til 20/03/2017, 20:00

DATA
Dust: 2.1
Temperature: 4
Humidity: 26
2017-01-24T16:47:05Z

Figure 5: Screenshot from the web application for exploring collected air quality measurements.

paneled visualization consisting of a map with measurement points plotted at their geographical location, in addition to other line plots visualizing the different measurements over time. Figure 5 shows a screenshot from an early version the web application with measurements from two different air:bits carried around in Tromsø, Norway.

EXPERIENCES

Our course has been delivered once in spring 2017 at the University of Tromsø to science students at a local High School (Kongsbakken Vidergående Skole). The class consisted of 26 students divided into groups of 3-4 students. The teacher formed groups consisting of students at the same level of experience. The students had varying previous experience with soldering, and no one except one student had coding experience.

The assembly of the kit went almost without any issues. The groups had to present their soldering to one in the staff before moving on to the coding. This allowed us to verify their soldering. We observed that students took turns soldering, and offered each other help. It was clear that some students had done more prior soldering than others.

Coding the kit was a more challenging task. Since most students had little experience writing software, it was oftentimes not clear that they understood the structure of a program and where to start debugging it. The Arduino IDE helped a bit, but most groups had difficulties completing the coding without any help from the instructors. We believe that this is related to the minimal code examples and instructions we provided the students. An interesting point made by one of the students was that it would be helpful with some tool (test suite) that could verify that their implementation was correct.

The students collected data over a 2 month period, each group deciding on when and where to bring their kits on their own. This resulted in measurements from different areas over different time periods. We believe that this stems from our instructions not being clear enough that they should focus on collecting data at the same locations and time of day to make the interpretation simpler. One group ended up with mounting the kit outside their house to simplify data collection. Another group also photographed the road while they were collecting dust data to quantify the amount of snow and ice later.

Due to the scattered data points, the students experienced some difficulties interpreting the data. Because of this, groups with 'good' and complete datasets shared their data with other groups to help them answer their respective research questions. However due to the very simple dust sensor, groups had difficulties with comparing their data with official PM measurements.

After the project was complete we collected oral feedback from both the students and the teacher. Students enjoyed the interdiciplinary nature of the project, especially in groups where students could assign tasks evenly according to interest. The coding was by far the most difficult part of the project for the students as well as the teacher. Because the data interpretation was difficult because of the sensors and scattered data, students had to reflect on the study design and quality of the data before making any conclusions. While we are going to improve the data quality by using improved sensors, the teacher enjoyed the student's reflection on study design and data collection.

FUTURE WORK

While we successfully deployed the air:bit project in a upper secondary school class of 26 students, we have identified areas for improvement and some future directions for the project.

We will improve data quality from the air:bit by using an air quality sensor that can measure both PM2.5 and PM10 concentrations in the air. The current Sharp GP2Y1010AU0F optical dust sensor measures only the total amount of dust particles and cannot be directly compared with the air quality stations from the NILU. We will use the Nova SDS011[3] that both measures PM2.5 and PM10. We will also replace the DHT11 temperature and humidity sensor with the DHT22 temperature and humidity sensor since its readings more accurate and can read temperatures below 0 °C. With these new sensors we are aiming to perform a formal evaluation of their accuracy by sampling air quality data at one of the air quality measurement stations of NILU. We estimate that the new sensors will increase the total cost of each kit by USD $15.

We are also re-writing both the interface for students to explore their collected data, as well as redesigning both the backend storage solution, and the frontend to facilitate users from different parts of Norway. The current backend does not provide the low latencies required by the frontend visualization interface.

To make it possible to scale the project to 10 new schools we are improving the teaching materials that we distribute to the classes. Since we cannot host every class ourselves, we provide video lectures on the background material such as air quality and climate, as well as instructional videos on assembly of the kit and soldering. We are also re-writing the example code that we distribute to the students with better documentation and clearer code. We believe this will simplify and make the coding-part of the projects more pedagogical.

CONCLUSIONS

We have shown how we designed the air:bit, a simple programmable sensor kit for measuring air quality, and how we built a upper secondary school course for students to build and program the kit. We believe that by introducing students to electronics and coding in a citizen science project we motivate students to engage in both technology and the environment.

[3]aqicn.org/sensor/sds011

ACKNOWLEDGEMENTS

We would like to thank MET for providing open weather observations through data.met.no and yr.no. We would like to thank NILU for providing air quality measurements through their luftkvalitet.nilu.no portal. We would also like to thank Thomas Olsen, Torkjel Sandanger, and Juan Carlos Aviles Solis for their presentations and input on the projects.

REFERENCES

[1] "NOU 2015: 8 - fremtidens skole." https://www.regjeringen.no/no/dokumenter/nou-2015-8/id2417001.

[2] European Schoolnet, "Computing our future. computer programming and coding - priorities, school curricula and initiatives across europe." http://www.eun.org/c/document_library/get_file?uuid=521cb928-6ec4-4a86-b522-9d8fd5cf60ce&groupId=43887, 2014.

[3] W. H. Organization, "World health assembly closes, passing resolutions on air pollution and epilepsy." http://www.who.int/mediacentre/news/releases/2015/wha-26-may-2015/en, 2015. [Online; Accesssed: 02.06.2017].

[4] C. Guerreiro, "Air quality in europe: 2013 report," 2013.

[5] R. Beelen, O. Raaschou-Nielsen, M. Stafoggia, Z. J. Andersen, G. Weinmayr, B. Hoffmann, K. Wolf, E. Samoli, P. Fischer, M. Nieuwenhuijsen, *et al.*, "Effects of long-term exposure to air pollution on natural-cause mortality: an analysis of 22 european cohorts within the multicentre escape project," *The Lancet*, vol. 383, no. 9919, pp. 785–795, 2014.

[6] R. D. Brook, S. Rajagopalan, C. A. Pope, J. R. Brook, A. Bhatnagar, A. V. Diez-Roux, F. Holguin, Y. Hong, R. V. Luepker, M. A. Mittleman, *et al.*, "Particulate matter air pollution and cardiovascular disease," *Circulation*, vol. 121, no. 21, pp. 2331–2378, 2010.

[7] O. Raaschou-Nielsen, Z. J. Andersen, R. Beelen, E. Samoli, M. Stafoggia, G. Weinmayr, B. Hoffmann, P. Fischer, M. J. Nieuwenhuijsen, B. Brunekreef, *et al.*, "Air pollution and lung cancer incidence in 17 european cohorts: prospective analyses from the european study of cohorts for air pollution effects (escape)," *The lancet oncology*, vol. 14, no. 9, pp. 813–822, 2013.

[8] M. Pascal, M. Corso, O. Chanel, C. Declercq, C. Badaloni, G. Cesaroni, S. Henschel, K. Meister, D. Haluza, P. Martin-Olmedo, *et al.*, "Assessing the public health impacts of urban air pollution in 25 european cities: results of the aphekom project," *Science of the Total Environment*, vol. 449, pp. 390–400, 2013.

[9] J. A. Hagen, P. Nafstad, A. Skrondal, S. Bjørkly, and P. Magnus, "Associations between outdoor air pollutants and hospitalization for respiratory diseases," *Epidemiology*, vol. 11, no. 2, pp. 136–140, 2000.

[10] J. Dutta, C. Chowdhury, S. Roy, A. I. Middya, and F. Gazi, "Towards smart city: Sensing air quality in city based on opportunistic crowd-sensing," in *Proceedings of the 18th International Conference on Distributed Computing and Networking*, p. 42, ACM, 2017.

[11] N. Castell, M. Kobernus, H.-Y. Liu, P. Schneider, W. Lahoz, A. J. Berre, and J. Noll, "Mobile technologies and services for environmental monitoring: The citi-sense-mob approach," *Urban climate*, vol. 14, pp. 370–382, 2015.

[12] A. Antonić, V. Bilas, M. Marjanović, M. Matijašević, D. Oletić, M. Pavelić, I. P. Žarko, K. Pripužić, and L. Skorin-Kapov, "Urban crowd sensing demonstrator: Sense the zagreb air," in *Software, Telecommunications and Computer Networks (SoftCOM), 2014 22nd International Conference on*, pp. 423–424, IEEE, 2014.

[13] D. Oletic and V. Bilas, "Design of sensor node for air quality crowdsensing," in *Sensors Applications Symposium (SAS), 2015 IEEE*, pp. 1–5, IEEE, 2015.

[14] hackAIR, "The hackair project." http://www.hackair.eu/pages/about-hackair, 2017. [Online; Accesssed: 16.08.2017].

[15] Luftdaten, "luftdaten.info – feinstaub selber messen." http://luftdaten.info, 2017. [Online; Accesssed: 16.08.2017].

[16] J. D. Brock, R. F. Bruce, and S. L. Reiser, "Using arduino for introductory programming courses," *Journal of Computing Sciences in Colleges*, vol. 25, no. 2, pp. 129–130, 2009.

[17] L. Buechley, M. Eisenberg, J. Catchen, and A. Crockett, "The lilypad arduino: using computational textiles to investigate engagement, aesthetics, and diversity in computer science education," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 423–432, ACM, 2008.

[18] E. Brunvand and N. McCurdy, "Making noise: Using sound-art to explore technological fluency," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, pp. 87–92, ACM, 2017.

[19] C. A. Petry, F. S. Pacheco, D. Lohmann, G. A. Correa, and P. Moura, "Project teaching beyond physics: Integrating arduino to the laboratory," in *Technologies Applied to Electronics Teaching (TAEE), 2016*, pp. 1–6, IEEE, 2016.

/C

Source Code

We have open-sourced the codebase for the air:bit platform on github:

The air:bit web application and frontend server:
github.com/fjukstad/luft

The backend:
github.com/ninaangelvik/luft