A scalable COVID-19 screening platform

Cristian Chilipirea, Luciana Morogan and Ștefan-Adrian Toma
Military Technical Academy "Ferdinand I"
Bucharest, Romania
Email: { cristian.chilipirea; luciana.morogan; stefan.toma } @mta.ro

2020 IEEE Globecom Workshops (GC Wkshps) | 978-1-7281-7307-8/20/$31.00 ©2020 IEEE | DOI: 10.1109/GCWkshps50303.2020.9367539

***Abstract*—The COVID-19 pandemic has put a strain on health facilities world-wide. However, remote screening can lessen the burden on medical resources. If done manually, screening does not scale to the large number of people that require it.**

We are constructing an automated remote screening system for Romania. The system needs to support simultaneous use by many persons (presumably a significant part of the population). Considering the urgency, we propose a light-weight web-based platform that can run on low-end server infrastructure. One auxiliary goal of the platform is for it to use machine learning (ML) to be able to adapt its screening results (recommendations) to the evolution of pandemic.

We considered using cloud services, however, due to privacy considerations regarding medical data and legal issues we designed the platform to be able to run, but to not require cloud services. Some of the decisions that we made were to offload computation (ML model inference) to the browser, minimize the number of requests to the server, as well as the code size, and construct a server architecture that uses buffers for writing to the database and which can scale on demand.

I. Introduction

The identification of COVID-19 in China in November 2019 launched global efforts to develop effective tools for epidemiological investigation, diagnosis, and treatment (including a vaccine). Many countries have addressed the COVID-19 pandemic by building mobile or web platform. Although the resulted impact of these platforms varies, for almost all of them, a common issue remained: the low number of people that use the platforms.

We are currently developing a screening platform for Romania. To ensure that it is used by as many individuals as possible we have focused on making it accessible and simple to use.

The proposed platform is web based and will work on both desktop and mobile systems, to be accessible by as many users as possible. The platform will present the user with a set of questions. We limit the number of questions to make sure that the user does not get discouraged while going through the questionnaire. Based on the answers that the user gives on the questionnaire, a machine learning model is used to calculate what is the best recommendation. The recommendations can be to go to a hospital emergency room, contact the family doctor or continue to practice social distancing.

The system we are building, when finished, may be used on a large scale. Following the solution proposed in [1], we will not require user login. There will be no registration, no users, and no passwords. This is meant to remove an important barrier in the way of user acceptance. However, this makes the system more vulnerable to denial of service (DOS) type attacks. We must ensure the system's scalability, to guarantee continuous operation even during such conditions. Furthermore, we must ensure efficient use of the resources so that each server resource can serve as many users as possible.

In this paper, we present the architectural design decisions that were taken to ensure the reliability of our platform. The main features are making use of a distributed, lightweight design, moving as much of the processing to the user and making use of buffers.

II. Survey of the Global Technical Response to COVID-19

At present, most information on specific and predictive clinical features of COVID-19 is related to the hospitalized patients [2], [3]. A recent work [4] highlights the limitations of current diagnostic and prognostic prediction tools.

In response to the lack of information on COVID-19, many countries have developed IT platforms that address the issue. We conducted a short survey of existing IT platform solutions aimed at wide population use. The results of our analysis are summarized in Table I. As can be observed, the main functionality of the platforms varies significantly. In countries like the US, France, or Germany the platforms are non-intrusive, taking the form of Chatbots that gather information on the user to provide recommendations. More intrusive platforms use the phone sensors to determine the individuals with which the user has been in physical contact. This data is called contact tracing data and it can be used to trace the spread of the pandemic. Contact tracing data can take the form of GPS locations, proximity records (gathered when two devices using WiFi or Bluetooth are within a short distance) or presence logs (gathered by manually scanning QR codes). These QR codes are specifically encoded and placed at interest locations.

The analysis of data gathered using these platforms has already showed interesting results. Data gathered with a platform developed in the UK [5], in which individuals tracked COVID-19 related symptoms, was

Table I: National IT platforms in response to COVID-19

Country	Beneficiary (Developer)	Technology	Reference	Results/Number of Users
Hong Kong	CHP, Department of Health	App limit quarantine movement	StayHomeSafe	10k+ - imposed
Singapore	Government Technology Agency	Phone alerts + Whatsapp + Tracing bluetooth	TraceTogether	500k+
Taiwan	NHI	Health card + colors + rations	-	NA - imposed
China	(Alibaba)	App trace QR + Colors + Limit access	AliPay QR extension	NA - imposed
South Korea	Government	Public maps traces of infected	coronamap.site + Co100	-
Australia	DTA	App screening + Notifications of quarantine	Coronavirus Australia	500k+
New Zealand	Ministry of Health NZ	App trace QR	NZ COVID Tracer	5+
India	NIC eGov Mobile Apps	Trace Bluetooth + GPS	Aarogya Setu	100M +
United Kingdom	(Zoe Global Limited)	Self record symptoms	COVID Symptom Study	1M+
Germany	DRK (Tyntec, Future of Voice)	Whatsapp Chatbot	-	-
France	Gouvernement	Whatsapp Chatbot	-	-
Norway	Folkehelseinstituttet	Contact tracing GPS + Bluetooth + Public database of infected	Smittestopp	100k+
SUA	(IBM, CDC, Microsoft, Apple, Google)	Chatbot Screening symptoms	Apple COVID-19, projectbaseline, CDC Coronavirus Self-Checker	-
Brazil	Ministério da Saúde	Whatsapp Screening Chatbot + population education	-	-

analyzed and enabled the identification of anosmia as a possible diagnostic predictor of COVID-19 [6].

In the last column of Table I we have estimated the adoption rate of the platforms based on publicly available data (number of installs on Google Store). Some countries introduced mandatory use: China restricts access to shops and inter-city transportation; In Hong Kong, all quarantined people are obliged to use the application and alerts are set if they leave their home. When considering countries where the use of the platform is not mandatory, the adoption rates are small. India has the highest number of users. However, even in this case, users remain a small percentage of the total population.

We focus our efforts in developing a non-intrusive, accessible self-screening solution. Our goal is to develop a platform that will be used by many people, and, with user agreement, will collect a significant amount of data. This is key to improving the accurate clinical recognition of the pathology of COVID-19 and predicting its evolution. However, due to the low world-wide adoption rate of COVID-19 related platforms, we believe user acceptance to be a primary concern. We aim for our platform to support and encourage the use by a large fraction of the Romanian population. Unlike other COVID-19 platforms [7] we need to insure that we offer a scalable solution. The benefit of this platform is both at the individual user and at the societal level, by improving the management of the COVID-19 pandemic.

III. Scope and General Description of the Platform

The platform will consist of a component exposed to the user as a web application and an administrative component to be used by operators (eg. medical specialists). The administrative component will allow the operator to re-estimate the parameters of the machine learning model, to visualize statistics and interactive maps as well as be alerted based on configured triggers. The operator will be able to start mass notifications, via personalized SMSs (e.g. to all people in a certain city with certain symptoms or who are over a certain age).

A user accessing the platform will be greeted with a set of questions. These will be displayed consecutively, along with possible answers. The answers given by the user will be used to generate a recommendation. The questionnaire will address the following topics:

- Demographic data
- Geographical location
- Environmental risk factors
- Direct or indirect contact with infected persons
- Medical history
- Information on user recent testing, confirmation, and evolution of an already known COVID-19 disease

Based on the answers, the application offer one of the following recommendations:

- Practice social distancing (GREEN)
- Contact family doctor by phone (YELLOW)
- Call emergency services (RED)

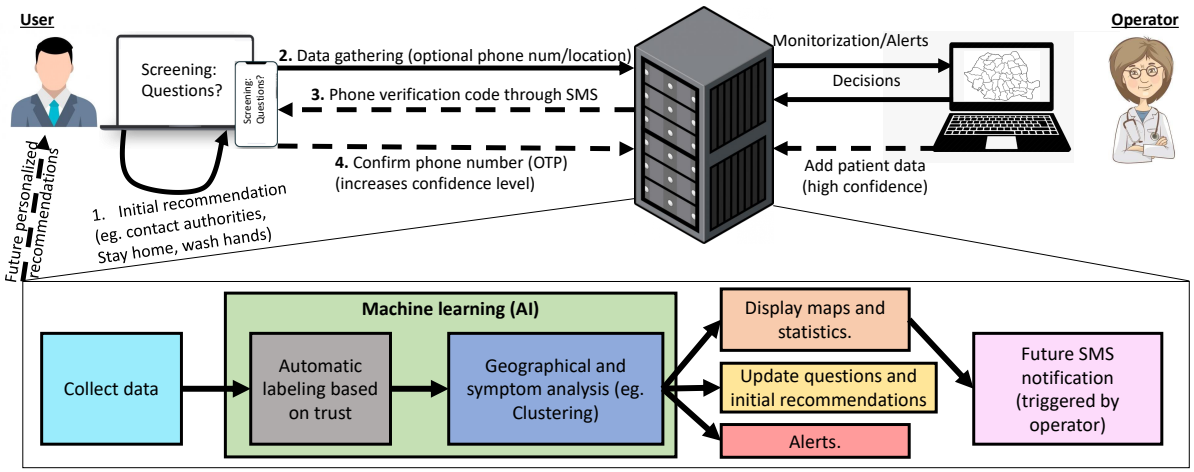

Figure 1: General architecture

The questionnaire will also include items for verifying the veracity of information. The aim is to label the data with different confidence levels. Furthermore, the confidence levels will increase if the phone number has been verified and it will automatically decrease when the data is identified as having a malicious source (eg. multiple entries with the same source, DOS).

At the beginning of the questionnaire the user will be informed that based on his willingness to share his phone number and data, he will receive notifications about the next steps he should follow (e.g. reusing the platform, consulting a doctor). The phone number will be verified using a unique code, One Time Password (OTP) sent via SMS. To those people who have reported symptoms suggestive of COVID-19, automatic notifications are sent guiding them to answer a different set of questions on the evolution of the disease.

The platform has the following components:

- **Screening application** will be designed as both a smartphone app and a web page, using cross-platform technologies. The application displays the questionnaire (which can be configured) and collects the answers.
- **Data collection module** will provide simultaneous usage by a large number of users. Scalability will be ensured by using Docker containers [8]. The data coming from the users will be temporarily stored in a Redis server [9] before being moved to a MySQL database. The Redis server is a type of in-memory data storage, assuring fast reads and writes. In the event of a sudden increase in traffic, the Redis server will act as a buffer, storing answers before they can be written to the database. If this is not sufficient, new Docker containers with Redis servers will be started.
  With the consent of the user, location data can be collected by using GPS (directly from the browser via HTML Geolocation API); IP-based geo-location (via an IP Lookup service); or user input.
  The website uses no cookies and the user can opt out of sending any data. This insures we comply with GDPR guidelines. Because the inference is done in the browser (see next bullet point) the platform remains completely functional for those who opt out from data collection. This ensures that the platform can reach a wider audience (even those who are concerned about their privacy) and important data can be collected from those willing to share it.
- **Machine Learning Module (AI)** will take the form of a neural network which will determine the risk of a possible infection based on the user answers. Based on the level of risk a specific recommendation is displayed. To collect training data, we plan to launch the platform with an expert system (decision tree) designed by a medical team. The collected data (with possible changes to the recommendations determined by the operator) will be split in a training and test set. After the neural network is trained the accuracy is measured on the test set and the result is presented to the operator. Depending on how it compares to the expert system, the operator can then choose to use the trained neural network. Regardless of the model used, it will be encoded in a JSON format and sent to the browser, which will conduct the inference on the user's answers. The neural network can be retrained as needed. Furthermore, we will use machine learning to detect the misuse of the platform, by labeling data that is likely to come from malicious sources as untrustworthy. Clustering of locations will be used to build interactive maps.
- **User Notification Module** will send customized messages, on demand. The messages will be sent via SMS to those who gave their consent and whose phone numbers have been verified.

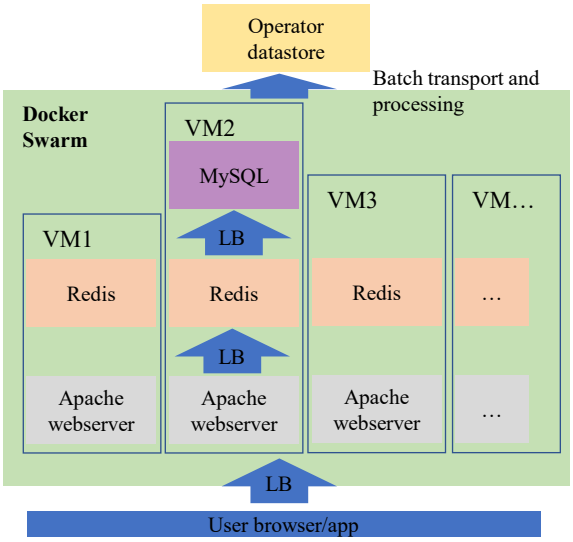

Figure 2: User facing architecture
(LB - Load balancer; VM - Virtual machine)

IV. User Facing System Design

As we showed in Section II, a screening platform is effective only if it is used by a large part of the population. As such, we aimed to build a platform that can scale on demand. To facilitate its extension with more physical or cloud resources, we decided to make use of *Docker* containers as is shown in Figure 2.

Docker Swarm is configured to run over multiple virtual machines (VM). When deployed, it is possible to add physical machines, however, the use of virtual machines allows us to add resources faster and to make use of cloud systems. The number of machines in our platform is limited only by the *Docker* system and the load balancer. Currently we are using the default load balancer provided by *Docker Swarm*.

Due to the design of our platform, when we serve the user webpages, we do not extract anything from the database. Furthermore, we only make one insertion when the user submits the answers. We know that *MySQL* databases can be slow when handling insert requests, therefore, we added a buffer between the database and the webserver in the form of *Redis* in-memory data store. When we need to serve many simultaneous users, the buffer will gather all the data. Eventually the data will be offloaded to the database.

Between each layer (user, webserver, buffer, database) we use a load balancer. This means that we can scale each layer independently and that requests are spread over the entire infrastructure. Furthermore, having a load balancer ensures that if any of the machines stops functioning others will automatically take over.

The data is stored in the *MySQL* database only temporarily. Once per night, a batch service will move and process the data. This is also the moment in which we create backups. We have not yet considered using a backup *MySQL* database due to the costs implied (the rest of the architecture uses open source solutions) and because data needs to be stored in *MySQL* for only one day.

If hardware resources from multiple sources are required to serve a high number of users, we plan to use the DNS system on top of the first load balancer, the one placed between the users and the webservers. DNS type A records can support multiple IP addresses and the browser should randomly choose one when making a request. This allows the resources to be completely separated, both in physical location and control plane.

On top of designing the architecture to support heavy load, even with few resources, we model the front-end of the questionnaire so that it makes a minimum number of requests. This helps to maximize the number of users that we can serve simultaneously.

As previously described, the website displays the questions one at a time, and it allows the user to answer by choosing from a list of possibilities. Finally, a score is computed, and a message is displayed. A naive approach would have the web client make a request to the server after every question, sending the answer and receiving the next question. Finally, when all questions have been answered the server would compute a score and send the recommendation to be displayed by the client.

Our approach is to move as much as possible of the computational load to the client. All the questions, the possible answers and the model used to make the inference and decide what recommendation to display, as well as all possible recommendations, are sent to the browser in the initial set of requests (requests for the HTML, CSS, JavaScript and image files). The sum of all files totals to under 60KB, making the requests very short. The browser displays all the questions, one after the other, and determines which recommendation message to display to the user. When finished, all the answers are submitted to the server in one request.

An important advantage of using this technique resides in the fact that all the content which is given to the client in the initial requests is somewhat static. We say "somewhat static" because we expect that the model used to determine the recommendation message will change in time. However, we do not expect this change to undergo more frequently than once per day. This means that the file holding both the questions and the model can be cached and, in the case of a heavy load, it can be served by a content delivery network with an expiration timer of one or several hours. All other files (Images, JavaScript, CSS, HTML) are completely static, making them trivial to cache. Based on the work in [10], we decided to pre-load the machine learning model file used for the screening procedure and to employ the CPU, rather than the GPU when running the inference process.

The application collects the phone numbers of will-

ing participants, to be able to message them at a later time (asking them to take another questionnaire or to send specific recommendations). The phone number can also serve as user identification. However, before we send any messages, we want to confirm that the phone number belongs to the person using our application. To confirm the phone numbers, we make use of one-time passwords (OTP), sent over SMS.

Classically, a system that confirms phone numbers uses multiple requests to the server. The initial request sends the phone number and triggers the generation of a One Time Password (OTP). In the browser, the user is asked to enter the OTP (received by SMS). Once the user fills in the OTP field and presses a confirmation button, a request is made to the server and the OTP is compared to the generated one. If they do not match, the user is asked to re-enter it and the procedure is repeated. Every time the procedure is repeated a new request is made.

In our platform, the phone numbers are meant for later use. This means they do not need to be immediately verified. Similarly, to the classical model, the phone number is sent to the server and a OTP is generated and sent to the user via SMS. However, when the OTP is generated, the value is hashed, and the hashed value is sent in the reply to the browser. When the user enters the OTP in the browser, we can calculate its hash and make the verification in the browser. If it is incorrect, we can alert the user and have him enter it again without making any requests to the server. The OTP is stored in the browser and sent with all the other answers to be verified again by the server.

We considered brute force attacks for discovering the OTP (e.g. generating all possible combinations and comparing them to the given hash). However, this approach would require both significant computing resources and time. We believe that this balances with the small security impact of having a recommendation SMS sent to the wrong recipient. Furthermore, the OTP expiration time frame remains small.

V. Results

We tested our platform using three virtual machines. These virtual machines had assigned 2 *Intel Xeon CPU E5-2640* cores and 2 GB of RAM each. The VMs have limited specifications in order to both ensure that our solution is efficient even when only few resources are available and to make the results easily reproducible. As previously described, on top of these VMs, we ran the *Docker Swarm* infrastructure. The alert and the monitoring components are separated from the questionnaire web service and the data gathering platform. We plan to move the data in batches from the user facing part of the platform to the operator facing one, at a specific hour of the night. As such, we do not consider the operator part in our analysis.

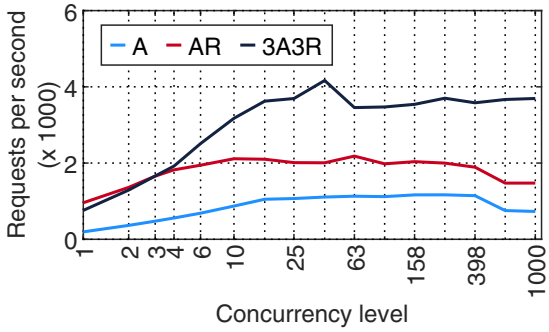

Figure 3: Comparison between different architecture choices (A - Apache, R - Redis, 3X - three X containers)

To validate our architecture design, we compared its performance with a classical one as well as with intermediate ones (further presented). This allows us to validate that each individual component increases the performance of the overall system and is not superfluous.

To measure the efficiency of our platform we consider the number of requests per second that the platform can serve for the writing operations. These operations take place when the user submits the questionnaire's answers.

To count the number of requests per second which can be served by the platform, we used Apache Benchmark[1]. For each measurement we made 10000 POST requests. We used multiple concurrency levels ranging from 1 to 1000 (on a log scale). The benchmarking application was run on a 4th virtual machine inside our infrastructure. This was done to measure the performance regardless of the limitations imposed by the bandwidth. It is quite likely that the values will change dramatically when considering the bandwidth limitations or adding components such as load balancers (external to the *Docker Swarm* one) or firewalls. The results of our measurements are presented in Figure 3.

We can observe that by using a *Redis* server as a buffer between *MySQL* and *Apache*, we can serve twice as many requests. Moreover, when we ran three web servers and three *Redis* data stores, the number of requests per second that we can serve grows even higher.

To determine what is the biggest performance impact when it comes to scaling, we made the same measurements using different number of *Apache* and *Redis* containers. The results are presented in Figure 4. One can observe that the biggest impact is brought by increasing the number of *Apache* containers. This is because *Apache* processes the user requests. Furthermore, when we increase the number of *Redis* containers, but keep just one *Apache*, the load balancer forwards two thirds of the requests from the server running the *Apache* container to

[1]https://httpd.apache.org/docs/2.4/programs/ab.html (Accessed on 10-Jul-2020)

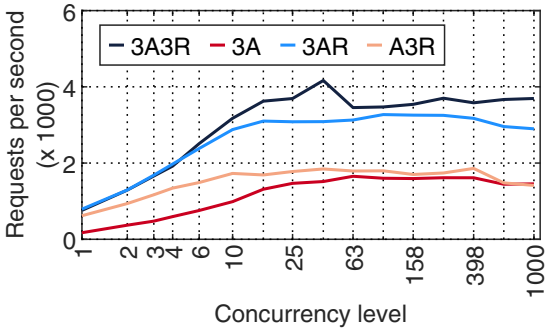

Figure 4: Comparison of performance when scaling different parts of the application
(A - Apache, R - Redis, 3X - three X containers)

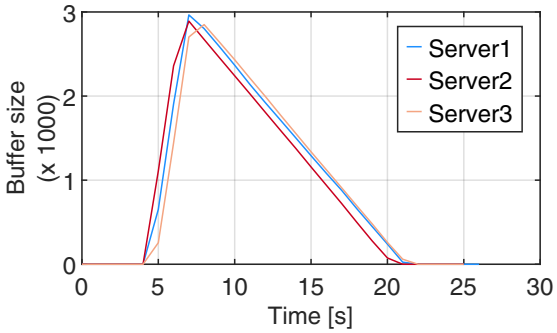

Figure 5: Load on buffers is evenly split
If the buffers were not needed, the number of elements in them would remain small

the other two servers. This has a significant cost. Using a location aware load balancer may improve the overall performance.

To confirm that the buffer layer works as expected, we recorded the change in the number of the elements in the buffer while we ran *Apache benchmark* with 10,000 write requests and a concurrency level of 1000. The number of elements in the buffer was recorded using scripts added to the *Redis* containers.

As can be observed in Figure 5, soon after the request started the number of elements in the buffer increased. After reaching the peak, even though more requests came in, the script managed to move data to the database. This is due to the way that *Apache benchmark* limits the requests. Once the maximum number of concurrent requests is created it waits for requests to end before it creates new ones.

The fact that the change to the buffer size follows the same pattern in all three servers confirms that the load balancer works as expected. We do note that the default *Docker Swarm* load balancer forwards requests in a round robin fashion. This would be problematic if the requests varied significantly in size or processing time. However, we expect requests to be similar and the *Redis* server performs the same operation for all.

VI. Conclusion

In this paper we presented the architecture for a scalable COVID-19 screening platform (CovShield), for Romania. We have shown that based on the proposed architecture, we can serve more than 3,500 POST requests per second. This is done using only three virtual machines with limited resources.

We expect to deploy the platform on an infrastructure with additional computational resources. As such, we anticipate a significant increase in capacity. Furthermore, due to the use of the *Docker* containers, the platform can be easily extend.

With 3,500 POST requests per second we can potentially collect data from more than 200,000 individuals every minute. This data can be analyzed to eventually provide new insights on the evolution of COVID-19 in Romania, as well as provide insights on the symptoms which can be present in the general population.

Acknowledgment

This work was supported by a grant of the Ministry of Education and Research, UEFISCDI, project number PN-III-P2-2.1-SOL-2020-2-0360, within PNCDI III.

References

[1] M. Krausz, J. N. Westenberg, D. Vigo, R. T. Spence, and D. Ramsey, "Emergency response to covid-19 in canada: Platform development and implementation for ehealth in crisis management," *JMIR Public Health Surveill*, vol. 6, no. 2, p. e18995, May 2020. [Online]. Available: http://publichealth.jmir.org/2020/2/e18995/

[2] C. Huang, Y. Wang, X. Li, L. Ren, J. Zhao, Y. Hu, L. Zhang, G. Fan, J. Xu, X. Gu *et al.*, "Clinical features of patients infected with 2019 novel coronavirus in wuhan, china," *The lancet*, vol. 395, no. 10223, pp. 497–506, 2020.

[3] W. Liang, H. Liang, L. Ou, B. Chen, A. Chen, C. Li, Y. Li, W. Guan, L. Sang, J. Lu *et al.*, "Development and validation of a clinical risk score to predict the occurrence of critical illness in hospitalized patients with covid-19," *JAMA Internal Medicine*, 2020.

[4] L. Wynants, B. Van Calster, M. M. Bonten, G. S. Collins, T. P. Debray, M. De Vos, M. C. Haller, G. Heinze, K. G. Moons, R. D. Riley *et al.*, "Prediction models for diagnosis and prognosis of covid-19 infection: systematic review and critical appraisal," *bmj*, vol. 369, 2020.

[5] D. A. Drew, L. H. Nguyen, C. J. Steves, C. Menni, M. Freydin, T. Varsavsky, C. H. Sudre, M. J. Cardoso, S. Ourselin, J. Wolf *et al.*, "Rapid implementation of mobile technology for real-time epidemiology of covid-19," *Science*, 2020.

[6] C. Menni, A. M. Valdes, M. B. Freidin, C. H. Sudre, L. H. Nguyen, D. A. Drew, S. Ganesh, T. Varsavsky, M. J. Cardoso, J. S. E.-S. Moustafa *et al.*, "Real-time tracking of self-reported symptoms to predict potential covid-19," *Nature medicine*, pp. 1–4, 2020.

[7] T. Schinkøthe, M. R. Gabri, M. Mitterer, P. Gouveia, V. Heinemann, N. Harbeck, and M. Subklewe, "A web- and app-based connected care solution for covid-19 in- and outpatient care: Qualitative study and application development," *JMIR Public Health Surveill*, vol. 6, no. 2, p. e19033, Jun 2020. [Online]. Available: http://publichealth.jmir.org/2020/2/e19033/

[8] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[9] J. L. Carlson, *Redis in action*. Manning Publications Co., 2013.

[10] Y. Ma, D. Xiang, S. Zheng, D. Tian, and X. Liu, "Moving deep learning into web browser: How far can we go?" in *The World Wide Web Conference on - WWW 19*. ACM Press, 2019. [Online]. Available: https://doi.org/10.1145%2F3308558.3313639