

# P21015: Fall Risk Mobile

Matt Krol  
mrk7339@rit.edu

Jacob DeFord  
jwd5062@rit.edu

Paul Kelly  
pjk2563@rit.edu

Doyle Bartholet  
cdb3120@rit.edu

**Abstract**—Stroke is one of the leading causes of death and disability in the United States. Stroke survivors tend to have reduced motor functions, which puts them at higher risk of severe fall related injuries. At the time of writing, methods for assessing the fall risk of post stroke patients have been proposed, however, not many tools exist to collect the necessary data needed for these assessments in a practical and safe manner. In this paper, we propose a solution to the later that will allow post stroke patients to safely gather the data needed for fall risk evaluation from the location of their choice. Furthermore, our solution additionally allows post stroke patients to be assessed remotely once the necessary data has been gathered, eliminating the need for travel.

## I. INTRODUCTION AND PROBLEM STATEMENT

In the United States, someone has a stroke every 40 seconds and someone dies from a stroke every 4 minutes [1]. Among the many challenges that post stroke patients face, fall related injuries are among the most common and can occur at any point of a post stroke patients daily routine. Studies have shown that chronic stroke patients fall at rates between 23% and 50%, with many patients reporting significant injury after such falls [2]. For this reason, many post stroke patients are assisted by caretakers or family members which can cause emotional stress and financial problems, among others. Therefore, it is critical that a post stroke patient and their family have all of the tools necessary to make an informed decision about how the post stroke patient is to be cared for. At the time of writing, many researchers are attempting to develop a quantitative fall risk scale. This fall risk scale would be an important tool for post stroke patients and their families to determine the appropriate precautions and care needed.

There are many challenges associated with developing such a fall risk scale. One of these challenges being that data must be collected from post stroke patients in order for them to be evaluated. While survey data is easy to obtain via a variety of different methods, the motion sensor data needed for many fall risk models would require specialized equipment and supervision, which is likely only available at a medical facility. For many post stroke patients, leaving their home or care facility can put them at an even higher risk of falling—not to mention—collecting data in a different location can result in inaccurate or unrealistic data used as input to the fall risk model which reduces the quality of the result. Another challenge is the delivery of the fall risk results. In many cases, a fall risk evaluation can help prevent injury or even death. This is why it is important that the fall risk result is communicated in an efficient and streamlined manner to all necessary parties as soon as the results become available.

In this work, we will focus on solving the problem of collecting and transmitting the data necessary for a fall risk evaluation and the problem of communicating the fall risk results back to the post stroke patient and other necessary parties. Our work is independent of the actual fall risk model. We hope that our framework will be widely adopted by different researchers developing different models for fall risk evaluation.

We will first discuss the proposed solution and design, then the feasibility of the design and benchmarks. We will follow up with concluding remarks and future development opportunities.

## II. PROPOSED SOLUTION AND DESIGN

In this section, we will discuss the overall architecture. Then, we will discuss each of the three components in detail. The three components are the mobile application, the web server, and the Matlab API.

### A. High Level Solution

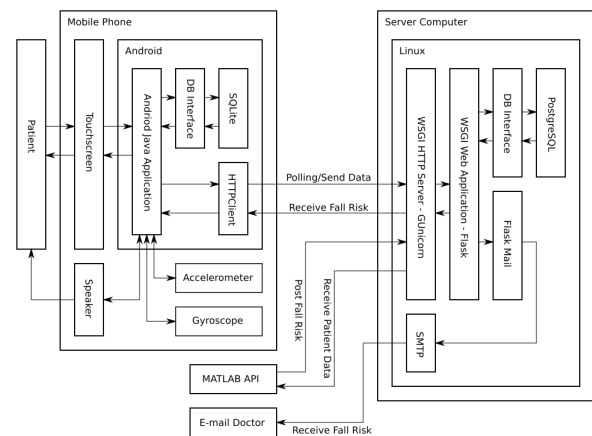


Fig. 1. System Level Design

Collecting and transmitting fall risk data requires several technologies. First, the data must be collected using motion sensors attached to the patient. The motion sensors should be easily controllable by the patient so that no training is required to take the assessment. Next, the data must be transmitted digitally to a storage device that can be accessed by the researchers. The storage device should be easily accessible from Matlab, since the fall risk metrics will be calculated using

Matlab. The storage device should also be able to hold data from many different patients in an organized manner.

Figure 1 shows the proposed overall system level design. The solution is broken down into three main components: a mobile application, a web server, and a Matlab API. The smart phone application was chosen for its rapid development, ease-of-use, and built-in motion sensors. Modern smart phones provide both the motion sensors required and the easy-to-use interface required by the patient-side device. Most patients will likely have a smart phone, and a free mobile application is accessible by anyone with an Android phone. The web server was designed to provide a universal API that is easily accessible from any device while still providing some security. The API provides an interface to a database that stores all of the patient data in an organized manner. The API's simple design allows for the Matlab API to also be very simple. All the Matlab script needs to do is format HTTP requests. This design effectively collects patient data in an easily-controlled manner, sends it to an easily-accessible storage device, and provides the information to virtually any internet-connected computer, including the researcher's Matlab device.

### B. Android Mobile Application

The mobile application development was broken up into two parts: the front end and the back end. This section will first discuss the front end, then the back end.

#### 1) Application GUI:

#### 2) Data Management and Networking:

### C. Flask Web Server

The HTTP REST API implemented on our web server is a key subsystem in our overall system design. This is because the HTTP REST API is what will handle bidirectional communication between the research team and the clients on their mobile application. The term HTTP refers to the application level internet communication protocol. The HTTP internet communication protocol has been a mainstay of the internet since the beginning of its existence and is still widely in use today. The term REST refers to a standardized interface for creating web services. We will follow the convention of using the JSON text format for sending and receiving information via the REST API. The main feature of a REST API is that it is stateless, which greatly simplifies the implementation. Finally, in our web server we must also utilize a relational database to store user profile information, survey data, tests data, and also fall risk results.

The main function of our REST API is to create a bidirectional communication loop between the clients on the mobile application and the research team. The database will store the necessary profile, tests, and survey information. A typical usage scenario of the REST API is as follows:

- 1) Client creates a profile via the REST API
- 2) Client requests a temporary login token via the REST API.
- 3) Client collects survey and test data via the mobile application.

- 4) Client sends the survey and test data to be stored in the web server database via the REST API.
- 5) Researchers pull survey and test data from the web server database via the REST API.
- 6) Researchers analyze data and evaluate fall risk.
- 7) Researchers post fall risk results to the database via the REST API.
- 8) Client pulls fall risk evaluation data from the database via the REST API.
- 9) Repeat if necessary by going back to step 2.

In addition to implementing the REST API calls needed for the steps above, we also will implement REST API calls for the clients to:

- Edit their profile information, i.e., change their e-mail address, password, etc.
- Delete their profile, survey, and test information.
- Reset their password in case they forget it.
- Revoke a temporary access token.

For our database in the web server we can use one of the standard SQL relational database implementations. We will use a single table with the following columns:

- `id`: The unique integer id of a user.
- `firstname`: The string first name of a user.
- `lastname`: The string last name of a user.
- `email`: The unique string email address of a user.
- `admin`: A boolean flag indicating if a user has admin privileges or not.
- `creation_timestamp`: A datetime object containing the creation date of the user account.
- `modification_timestamp`: A datetime object containing the date that the user account was last modified. Defaults to the `creation_timestamp` value.
- `survey`: A JSON string containing a the survey data of a user.
- `survey_timestamp`: A datetime object containing the date of the last survey that the user took. This value can be `None`.
- `tests`: A binary blob containing the serialized Java class that should contain the tests data of a user. This value can be `None`.
- `tests_timestamp`: A datetime object containing the date of the last tests that the user took. This value can be `None`.
- `result`: A JSON string containing the fall risk result. This value can be `None`.
- `result_timestamp`: A datetime object containing the date of the last result that was posted.
- `password_hash`: A string containing the user hashed password.
- `token`: A string that contains a user temporary access token.
- `token_expiration`: A datetime object that contains the expiration date of a user token.

Each user will be stored in a row of the database. The above fields will allow us to store the necessary user profile

information, survey data, tests data, and fall risk results. Several timestamps allow us to determine whether or not a user has posted recent tests and survey information so they can be evaluated.

Lastly, our web service must have some sort of authentication system. The REST API endpoints that are used by each respective user via the mobile application should be private, i.e., users should only be able to modify or see resources in the database that belong to them. Secondly, all REST API endpoints used by the researchers should be sufficiently locked down and they should not be publicly accessible. Effectively, the researcher accounts will be administrator accounts. In the database, administrator accounts will be identified by the admin flag. This flag can only be set manually by a server maintainer for security purposes. Each of the REST API endpoint categories described will utilize a temporary access token that will expire after one hour. So users must first create an account, and then request a temporary access token in order to further obtain or modify resources in the database.

#### D. MATLAB Java API

### III. DESIGN FEASIBILITY AND BENCHMARKS

This section will discuss the specifics of the implementation and benchmarks.

#### A. Android Mobile Application

#### B. Flask Web Server

There are several so called web stacks for implementing web services. Given that our team has more experience using Python, we decided to use a Python based web stack. A Python web stack consists of a HTTP server, application server, Python web framework, and a relational database. We decided to use the very popular Flask web framework, along with PostgreSQL for our database and gunicorn for our combined HTTP and application server. All of these tools are open source and have been vetted by the open source community and industry, not to mention, they have been in use for around a decade. Additionally, all of these tools are widely documented and follow all of the standard protocols such as HTTP, PEP, TCP/IP, REST, etc. This not only allows us to focus on our higher level design, but it lowers our design to scale if needed to include other standardized services. Finally, we also needed a computer to run our services on. We were able to obtain a virtual machine from RIT to run our web service on. This virtual machine is running the Linux operating system which is ideal for deploying our web service. We utilized Docker to automate the deployment process.

In particular, our implementation was highly influenced by [3]. The author in [3] also developed many Flask extensions that we used to implement e-mail services (Flask-Mail), token and basic HTTP authentication (Flask-HTTPAuth), among others. The author in [3] is widely known in the Python web service community, and his Flask extensions have been vetted by the open source community. Hence, because we are utilizing standardized and community vetted tools, we feel that our implementation is feasible. Mobile applications and

web services are so common these days that many resources and tools exist to aid developers and we have tried to take advantage of this. At the time of writing we have been able to implement the main design requirements of our HTTP REST API, namely, the communication loop. While most of the testing has been manual, we believe that this is strong evidence that our design is feasible and will meet the demands of our client.

#### C. MATLAB Java API

### IV. CONCLUSION

#### REFERENCES

- [1] S. S. Virani, A. Alonso, E. J. Benjamin, M. S. Bittencourt, C. W. Callaway, A. P. Carson, A. M. Chamberlain, A. R. Chang, S. Cheng, F. N. Delling *et al.*, "Heart disease and stroke statistics—2020 update: a report from the american heart association," *Circulation*, vol. 141, no. 9, pp. e139–e596, 2020.
- [2] J. E. Harris, J. J. Eng, D. S. Marigold, C. D. Tokuno, and C. L. Louis, "Relationship of balance and mobility to fall incidence in people with chronic stroke," *Physical therapy*, vol. 85, no. 2, pp. 150–158, 2005.
- [3] M. Grinberg, *The New and Improved Flask Mega-tutorial*. Amazon Fulfillment, 2018.