*A****bstract— <placeholder>***

**Sleepify: A system towards personalizing and optimising sleeping environments**

**Jeremy Chan, Tsz Ho Ho, Dominic Kwok, Ho Shun Lo, Nathalie Wong**

Department of Electrical and Electronic Engineering, Imperial College London, SW7 2AZ  
Email: {jc4913, thh13, cyk113, hsl113, nw813}@ic.ac.uk

# Introduction

It is

# Sleepify’s Promise

This project aims to provide a better sleeping experience overall from having the room temperature automatically adjust to body and room temperature information from two sensors. Sleepify also promises improved performance and improved machine learning classification accuracy based on prolonged usage of the app. Continued usage of Sleepify is especially important for our machine learning algorithms; thankfully the retention rate of health and fitness apps are the highest among others [1]. Lastly, Sleepify promises to deliver a slick and intuitive app, and web interface for the user to use and interact with, motivating the user to continue using Sleepify regularly; this is crucial to having a low app abandonment rate [2].

# Related Work

There are many sleep trackers on the market that use a variety of ways to track sleep quality. Most sleep trackers monitor the user’s different stage of sleeping, sleeping environment and provide sleep coaching advice. Majority of the trackers are found in the form of software application for iOS and Android. These applications use the accelerometer found in smartphones to track body movement throughout the sleep cycle. Using this data, “Sleep Cycle” wakes the user up during the lightest sleep phase, preventing the feeling of tiredness in the morning. In addition to the accelerometer, “Sleep as Android” [3] records audio through the microphone to detect snoring, speech, and ambient noise. This can be played back to the user the following morning, and can be a good indicator of sleep disturbances and stress [4]. Additionally, some applications also include the feature of playing soothing sound or music to make the user fall asleep peacefully.

Hardware sleep trackers such as “S+ By ResMed personal sleep solution” contain even more features, such as synchronizing the output music with the respiratory pattern of the user to provide a calming effect [5]. Another interesting feature by “Aura Smart Sleep system” includes a red light to induce the user into sleep [6]. “Sense” has a slow wake up light alarm to gradually wake the user up. Most of the aforementioned also have questionnaires for the user to record their daily behavior to help analyze their sleeping pattern.

However, some of the down sides of these applications include inaccuracy in telling whether the user is just lying in bed or actually sleeping. Some drain the battery of both the device or the phone quickly. Some of the applications lack a snooze alarm function.

Sleepify has taken into account the pros and cons of these existing sleep trackers in the market when prioritizing its aims. In addition to the generic functions such as sleep coaching advice and sleep environment monitoring, it has taken an active role to provide a novel edge to sleep tracking - adjusting the sleeping environment. Sleepify analyses the best sleeping temperature and connects to smart heating devices to adjust the optimum sleeping temperature automatically. Manually changing the start time of the sleep record would also be enabled to prevent the problem of false sleep detection.

# System Design and Implementation

## Overall High Level Design

There are many types of software distribution models. Traditionally, users purchase a piece of software either through a retailer or online, and then install it onto their computer. The user then holds the license to use this piece of software, indefinitely. A drawback of this traditional method is that the user normally has to pay an upfront cost, update availability is subject to the package the user bought, and data only exists locally on the user’s machine [7].

Nowadays, the Software-as-a-Service (SAAS) model is the model upon which most companies are building their products around [8]. The SAAS model gives the consumer the ability to use on demand software that is provided by developer via the web or an app. From Heredia et al., as the user normally pays a subscription fee instead of an upfront fee, the SAAS model guarantees that the user will always be using the most updated version of the software as there is no ‘local copy’ of the software to install [9]. Moreover, SAAS removes the burden of having to configure (and control of) infrastructure for the user. However, SAAS solutions often assume that customers will always like new changes as updates are rolled out to all users [10] – this is not applicable to Sleepify as the timeframe of the project means only developing a minimal viable product (MVP).

Following the SAAS model, Sleepify consists of a front-end and a back-end, each consisting of two parts. The front-end is what the user sees and uses, and consists of an iOS application and the web interface. Updates through the App Store and the website ensure the user will always be using the most updated version of Sleepify. Finally, this front-end connects to the sensors for data collection and temperature adjustment. The back-end consists of the servers, databases, APIs, and machine learning modules – these both provide, and accept information from the front-end applications. The user has no information or control on how the back-end is configured; they need not to.

## Sensors

## Backend (Server, Database, API)

The backend is responsible for interfacing with the front-end, in accepting and providing it with the information it needs. It consists of a server on which a database resides, and a Representational State Transfer (RESTful) API which allows the iOS app, web interface, and machine learning sections to communicate with the server and by extension, the database.

### The Server

As the database, the web interface, and the API all reside upon the server, a smart choice needed to be made regarding how the server would be implemented.

Our group had prior experience in setting up a server running a LAMP stack (Linux, Apache, MySQL, PHP) in hacking together a simple custom API and website, but this was judged to be inadequate for Sleepify as trying to hand code PHP without a web framework when creating any sort of advanced web app would take an extremely long time. Laravel, and Yii, both modern PHP frameworks, were initially shortlisted as using a modern web framework would shorten development times drastically. However, the verbose and sometimes confusing syntax of PHP mean getting-things-done is more important than code readability [11]. As Sleepify’s development may continue in the future, reusability and code readability meant the group decided not to go with a PHP framework. In hindsight, further research from Srinivasan et al. also showed PHP to suffer from more security issues compared to other web frameworks [12].

Emphasising code readability and rapid development meant the choice was narrowed down to two programming languages: Python, and Ruby. The most well-known web framework in Python is Django, and in Ruby, Ruby on Rails. Both offer extremely fast prototyping and development, extensive documentation, security measures against common attacks, and multiple libraries to assist development. The final decision was to use Django, the Python web framework, as the ease of use of Python (especially considering the group had extensive C++ and Python experience) and the ample documentation on Django meant decreasing the time needed to create the MVP.

Multiple Django libraries were leveraged to add extra functionality, the most notable being: django-rest-framework, a library which provides the skeleton of the RESTful API; django-rest-auth, which extends Django’s already excellent authentication system with the API; django-bootstrap3, a library which simplifies styling a website using Twitter bootstrap.

### The Database

The main design decision when choosing the database was whether to go with a Structured Query Language (SQL) or a NoSQL database.

SQL databases are known as relational databases, where databases are linked together by keys and values [13], held in entries in database tables. In SQL databases, all the incoming data must match the format of the database table, whilst NoSQL operate on the premise that the incoming data is of a large volume and of a rapidly changing format [14].

The most well-known NoSQL database is MongoDB, and it offers several advantages over SQL databases. MongoDB claims scalability and performance improvements in [15], claiming that NoSQL databases are horizontally scalable (add more servers) instead of vertically scalable (have to make the one server more powerful). However, the flexibility of NoSQL data means there exists consistency issues when dealing with many similar data objects – unacceptable for user data. Another important advantage a NoSQL database has is its data format. Nayak et al. goes into more detail, showing the data being held in a binary Javascript Object Notation (JSON) object [16]; this can be accessed using object oriented methods. However, this advantage is nullified with Django as it has its own object oriented wrapper for any type of database. Django supports its own ‘Models’, which abstract away the complicated SQL statements needed to modify the database [17] in favour of treating database tables as objects, nullifying yet another advantage of NoSQL databases.

Hence, the final decision was to use SQL databases. Having a SQL database means structured data (sensor data is of a set format anyway), and relational relationships mean data can be linked with user profiles very easily. There are a few popular SQL databases, the most popular 3 being SQLite, MySQL, and PostgreSQL. The following table shows the pros and cons of each (taken from [18]); the final decision was made to use SQLite, a SQL database that comes shipped with Django by default, with the main justification coming from portability (copy and paste the database across testing machines, committable on Git), and it supporting enough features to not be considered bloated. Scalability issues have been moved down in priority as according to SQLite, they only occur at high volumes of data [19], an unrealistic target for Sleepify. Lastly, NoSQL support is not part of the official Django development effort, and is only supported via third party forks [20].

With regards to user security, the SQLite database is not encrypted now as the database is not reachable through the internet. However, in the future, a Django library known as django-fernet-fields can be utilised to encrypt database fields (unfortunately, floating point numbers are not supported yet).

Table 1: Comparison of SQL Databases

|  |  |  |
| --- | --- | --- |
| Database | Pros | Cons |
| SQLite | * Extremely portable as the whole database is one file * Feature rich for development and testing | * No in-built user management * Slow for large number of writes |
| MySQL | * Scalable, more feature rich than SQLite, tested * High security | * Reliability issues and discontinued development * Not fully SQL compliant |
| PostgreSQL | * Reliable in terms of data integrity * Supports complex queries | * Overkill for simple databases * Not portable without spending time replicating the database |

### The API

Creating an API was a top priority for the back-end as it enables a consistent communication format between the front and back-ends. Sleepify’s API exposes URLs in which data can be sent or retrieved, including but not limited to the following: machine learning results, raw sensor data, graphs, calendar events, and sending push notifications.

To create the API, a communication format and architectural style had to be decided. Nurseitov et al. compares the two main communication formats, eXtensible Markup Language (XML), and Javascript Object Notation (JSON) [21]. XML follows a rigid pre-defined structure while JSON does not have any pre-defined structures (large companies such as Google, Yahoo, and Microsoft have a web repository of such pre-defined structures but this is by no means compulsory - <http://schema.org/>), so initially it seemed XML was the way forward. However, since everything in XML is stored in strings, parsing the XML data takes relatively more processing power than that of JSON, which can have single entries or arrays of strings or integers – making JSON much more efficient, especially on mobile platforms as demonstrated by Sumaray et al. [22], making it Sleepify’s choice for the data format.

To decide on the architectural style, the pros and cons of Simple Object Access Protocol (SOAP), Representational State Transfer (REST), and Remote Procedure Call (RPC) were compared in the following table [23], [24], [25].

Table 2: Comparison of API styles

|  |  |  |
| --- | --- | --- |
| Style | Pros | Cons |
| SOAP | * Versatile, can use different protocols other than HTTP | * Extremely complex payload * Way too verbose for simple tasks * Legacy |
| REST | * Uses standard HTTP verbs * Intuitive and clean looking API URLs * Suited for getting data * Embraced by many as the way to go when designing APIs | * Not suited for calling functions or actions (URLs are nouns) * Not easy to do more than one thing in one request |
| RPC | * Uses standard HTTP verbs * Suited for verbs, functions, actions * Can have a custom verb do more than one action at once | * Getting data using RPC architectures is messy and inconsistent * Naming conventions up to the developer |

As SOAP relied on XML, it was not chosen. Based on these results, Sleepify chose to use a mixture of REST and RPC architectures. Data retrieval and insertion would be done using RESTful nouns such as /user/, /raw\_data/, /stats/, while push notifications and the machine learning training would be done using RPC verbs such as /push\_to\_devices/, /migrate\_features/.

As the API is built on the Django server, the django-rest-framework was leveraged to provide the skeleton of the API. Converting functions into RESTful and RPC compliant APIs were as simple as wrapping the function in an ‘APIView’ class. The library also provided an appealing interface to display data retrieved from the API without any added custom user styling. Authentication to the API is done through sessions/cookies, and is supported through defining permission classes in the API functions (e.g. statistics only available to logged in users, user registration, log in/out, open to the public). Another style of authentication is using JSON Web Tokens (JWT), but JWT does not allow pushing notifications to logged in clients as there is no way to know whether a user is logged in or not, as opposed to authenticating using sessions [26].

## Machine Learning

## iOS Application

## Web Interface

As per Sleepify’s promise, the web interface should be modern, intuitive, and easy to use.

# Evaluation Criteria and Setup

# Results

# Discussion

# Conclusion

In conclusion, this report highlighted the motivation behind in building a system that is capable to alter the users thermal sleep environment to achieve better sleep quality. We have identified that the thermal environment is a key factor in affecting sleep quality, this justifies our rational in controlling this factor in order to provide better sleep quality to users. We have discussed related works, however to the authors’ knowledge there is no work that has developed a complete system to alter sleeping environments. Finally, we have also presented some of our prelimiarly works.

# References

[1] ‘Enter the Matrix: App Retention and Engagement’, *Flurry Blog*. [Online]. Available: http://flurrymobile.tumblr.com/post/144245637325/appmatrix. [Accessed: 19-Mar-2017].

[2] ‘App Retention Improves - Apps Used Only Once Declines to 20%’. [Online]. Available: http://info.localytics.com/blog/app-retention-improves. [Accessed: 19-Mar-2017].

[3] U. Team, *Sleep as Android Unlock*. Urbandroid Team, 2016.

[4] M. M. Ohayon and C. M. Shapiro, ‘Sleep disturbances and psychiatric disorders associated with posttraumatic stress disorder in the general population’, *Compr. Psychiatry*, vol. 41, no. 6, pp. 469–478, Nov. 2000.

[5] ‘S+ by ResMed’. [Online]. Available: https://sleep.mysplus.com/. [Accessed: 01-Feb-2017].

[6] ‘Withings’. [Online]. Available: https://www.withings.com/uk/en/products/aura. [Accessed: 01-Feb-2017].

[7] ‘What is SaaS’, *Interoute*, 01-Mar-2017. [Online]. Available: http://www.interoute.com/what-saas. [Accessed: 19-Mar-2017].

[8] J. P. • M. 30 and 2016, ‘Top 25 Most Popular SaaS, Cloud Applications for Business’, *ChannelE2E*, 30-Mar-2016. [Online]. Available: https://www.channele2e.com/2016/03/30/top-25-most-popular-saas-and-cloud-applications/. [Accessed: 19-Mar-2017].

[9] A. Heredia, R. Colomo-Palacios, and A. de Amescua, ‘Software Business Models from a Distribution Perspective: A Systematic Mapping Study’, *Procedia Comput. Sci.*, vol. 64, pp. 395–402, Jan. 2015.

[10] ‘Weighing the Pros and Cons of a SaaS Solution’, *AT&T Networking Exchange Blog*, 28-Jul-2016. [Online]. Available: https://networkingexchangeblog.att.com/enterprise-business/weighing-pros-cons-saas-solution/. [Accessed: 19-Mar-2017].

[11] S. Das, ‘Which is Better, PHP or Python? A Developer’s Take’, *LinkedIn Pulse*, 11-Jun-2015. [Online]. Available: https://www.linkedin.com/pulse/which-better-php-python-developers-take-srikrishna-das. [Accessed: 19-Mar-2017].

[12] S. M. Srinivasan and R. S. Sangwan, ‘Web App Security: A Comparison and Categorization of Testing Frameworks’, *IEEE Softw.*, vol. 34, no. 1, pp. 99–102, Jan. 2017.

[13] tutorialspoint.com, ‘SQL RDBMS Concepts’, *www.tutorialspoint.com*. [Online]. Available: https://www.tutorialspoint.com/sql/sql-rdbms-concepts.htm. [Accessed: 19-Mar-2017].

[14] ‘NoSQL Databases Explained’, *MongoDB*. [Online]. Available: https://www.mongodb.com/nosql-explained. [Accessed: 19-Mar-2017].

[15] ‘MongoDB at Scale’, *MongoDB*. [Online]. Available: https://www.mongodb.com/mongodb-scale. [Accessed: 19-Mar-2017].

[16] A. Nayak, A. Poriya, and D. Poojary, ‘Type of NOSQL Databases and its Comparison with Relational Databases’, *Int. J. Appl. Inf. Syst.*, vol. 5, no. 4, Mar. 2013.

[17] ‘Models | Django documentation | Django’. [Online]. Available: https://docs.djangoproject.com/en/1.10/topics/db/models/. [Accessed: 20-Mar-2017].

[18] ‘SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems’, *DigitalOcean*. [Online]. Available: https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems. [Accessed: 20-Mar-2017].

[19] ‘Implementation Limits For SQLite’. [Online]. Available: https://www.sqlite.org/limits.html. [Accessed: 20-Mar-2017].

[20] ‘NoSqlSupport – Django’. [Online]. Available: https://code.djangoproject.com/wiki/NoSqlSupport. [Accessed: 20-Mar-2017].

[21] N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta, ‘Comparison of JSON and XML Data Interchange Formats: A Case Study’, Department of Computer Science Montana State University – Bozeman Bozeman, Montana, 59715, USA.

[22] A. Sumaray and S. K. Makki, ‘A Comparison of Data Serialization Formats for Optimal Efficiency on a Mobile Platform’, in *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication*, New York, NY, USA, 2012, p. 48:1–48:6.

[23] ‘REST vs XML-RPC vs SOAP’. [Online]. Available: http://effbot.org/zone/rest-vs-rpc.htm. [Accessed: 20-Mar-2017].

[24] ‘REST vs XML-RPC vs SOAP – pros and cons : Max Ivak Personal Site’. .

[25] ‘How REST replaced SOAP on the Web: What it means to you’, *InfoQ*. [Online]. Available: https://www.infoq.com/articles/rest-soap. [Accessed: 20-Mar-2017].

[26] R. Golwalkar, ‘Pros and cons in using JWT (JSON Web Tokens)’, *Rahul Golwalkar*, 21-Aug-2016. .