TransportIT

Christos Mourouzi
Department of Computer Science
University of Bristol
Bristol, UK
cm16663@my.bristol.ac.uk

Katerina Baousi
Department of Computer Science
University of Bristol
Bristol, UK
ab16651@my.bristol.ac.uk

***Abstract*— TransportIT is web application deployed in a cloud-based environment which creates and manages a goods transportation network. Its users can login, declare if they want to use the application in the scope of either a sender or a transporter, search for users that meet the requirements they have inserted in the application and communicate with them in order to confirm the transaction. Lastly it is hosted in Google App Engine.**

The present paper aims in describing the concept behind TransportIT and its realisation. The application can be run online at https://transport-it.appspot.com without predefined login in credentials, this meaning that each new user must sign up before attempting to log in.

Keywords- transport network, Google App Engine, Google Cloud Datastore

I. INTRODUCTION

In today's globalized world, individuals can travel securely and effortlessly everywhere. On the other hand, when it comes to goods transportation, things can become more complex. Post offices and courier services take charge of this kind of transportation. Furthermore, the fees that apply for this kind of service discourage its use from the consumers. TransportIT web application tries to solve this problem by encouraging travelers from around the world to transport goods to people in need of them.

The aforementioned web application allows users to log in using their credentials (username and password defined once they signed up in the application), sign in as a sender or a transporter and find users that match their needs. In more detail, once a user logs in the application, he is able to choose if he wants to be classified as a transporter or a sender. In each case the user needs to provide the required information regarding the weight and volume of the item to be transported and the possible arrival date of the item. The data provided by the user are then stored in the database, triggering queries in order to find other users that correspond to them. Finally, if a data matching occurs, diagnostic messages are shown on each user's main page.

II. CLOUD COMPUTING

A. Benefits of Cloud Computing for Implementing TransportIT

Cloud computing technologies have proven to be extremely helpful in implementing TransportIT. First of all, as mentioned above TransportIT uses extensively a database which storage needs grow accordingly to the number of the application's users. This characteristic might cause problems if we try to implement the application in the conventional way but thanks to cloud computing's scaling ability there is no need to worry about storage constraints. [1]

In addition, TransportIT delivers a big amount of static files (HTML5, CSS3, Javascript) to its end-users and for that reason it would be really important to deliver this kind of files from servers that are located as close to the users as possible in order to achieve the lowest latency possible. This is also impossible to accomplish because we would have to buy a lot of servers and distribute them globally. In a nutshell, there is virtually no possible way to build the same scalable, worldwide, reliable and efficient production environment near as cheaply as offered by cloud services.

B. IaaS vs PaaS

The Platform as a Service, or PaaS, and Infrastructure as a Service, or IaaS, are two of the service models used by NIST definition [2] formed to help us distinguish the differences between the offered remotely hosted services. PaaS gives its users the capability to create their own applications without having to worry about building the infrastructure needed to support the development environment. On the other hand, IaaS gives the opportunity to its users to hire bare machines and equip them as one wishes, define the memory size of the machine and its processing power. IaaS implementations are often used to replace internally managed datacenters.

The requirements of the application described in this paper are met by PaaS. The underlying infrastructure is not that important in deploying TransportIT. PaaS has helped us to implement our web application in a less complex way and enjoy the resources provided by it.

C. GAE vs AWS

The two main cloud services providers are Amazon and Google. Amazon Web Services offer the opportunity to use IaaS (Infrastructure as a Service) products such as AmazonEC2 (elastic cloud compute service) and AmazonS3 (simple storage service). AWS give developers the chance to be more specific about the number of instances they want to spin up and how their application will scale up when it faces high traffic.

Next in order, Google App Engine offers PaaS (Platform as a Service) for developing and hosting web applications in Google-managed data centers. Applications are sandboxed and run across multiple servers. As commented above we prefered PaaS over IaaS for deploying TransportIT and for that reason Google App Engine is hosting our application.

III. THE MODEL-VIEW-CONTROLLER PATTERN

TransportIT uses the MVC Pattern [3] to deliver its contents. This pattern helps us to break the application into three basic areas (Model, View and Controller) and add new functionality each time in the proper place. Briefly the Controller orchestrates all the activities performed in the application; the Controller receives an incoming HTTP request, handles the session, makes any Model changes, selects the View, renders the View and sends back to the browser the HTTP response.

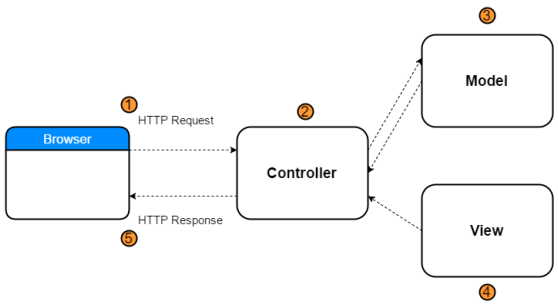

Fig. 1. The MVC Pattern

In more detail the aforementioned pattern is deployed as follows:

A. Model/Datastore

In the Model, we store the persistent data that we keep in the database. For the storing needs of our application we used Google Cloud Datastore which is a NoSQL document database using Google's Bigtable technology. The focus of the Bigtable and the Google Datastore is to build high performance applications that scale automatically. [4] According to this, data are stored as entities in Bigtable by row. Entities are formed by kinds. [5] Each entity has its own properties and it is identified by a unique key. Bigtable can retrieve the entities by using this unique key. The limitation presented by Bigtable is that it is does not support queries, i.e it is not possible to fetch an entity based on a property. However, the Datastore supports queries by building multiple single property indexes. An index table is created in order to perform inequality, equality and order queries on a single property.

B. View/Static Files

This part consists of the HTML5, CSS3 and Javascript used to bring about the look and the feel of the application. This kind of files do not change during the regular operation of the site and for that reason they are widely known as static files. The code for the static files used by TransportIT was uploaded right alongside the application code. Since the delivery of these files doesn't involve application code, it's unnecessary and inefficient to serve them from the application servers.

Instead, App Engine provides a separate set of servers dedicated to delivering static files. These servers are optimized for both internal architecture and network topology to handle requests for static resources. To the client, static files look like any other resource served by our app.

C. Controller/webapp2

This part is implemented using the webapp2 framework [6] in Python 2.7 programming language that is provided as part of Google App Engine. The webapp2 framework takes care of many of the mundane details of the HTTP interactions (request/response cycle). This framework offers request-handlers responsible for receiving an HTTP request (GET or POST), storing or retrieving data for the model, selecting the appropriate View and sending back to the browser the HTTP response.

IV. DESING OF THE APPLICATION

A. Back-End

The back-end implementation for TransportIT is developed in Python programming language. Webapp2 request-handlers are responsible for completing a specific task. Also, a simple session utility [7] is used for storing data that persists across multiple request/response cycles. The handlers implemented in this application are described below:

- SignUP Handler
  The SignUp Handler, takes as input data the user's personal details and stores them in the Datastore. That means that it creates a new User(kind) datastore entity, giving to its properties the correct input values. It also makes sure that the account name does not already exist using a Datastore query. If the sign up is successful, the handler renders the login page.

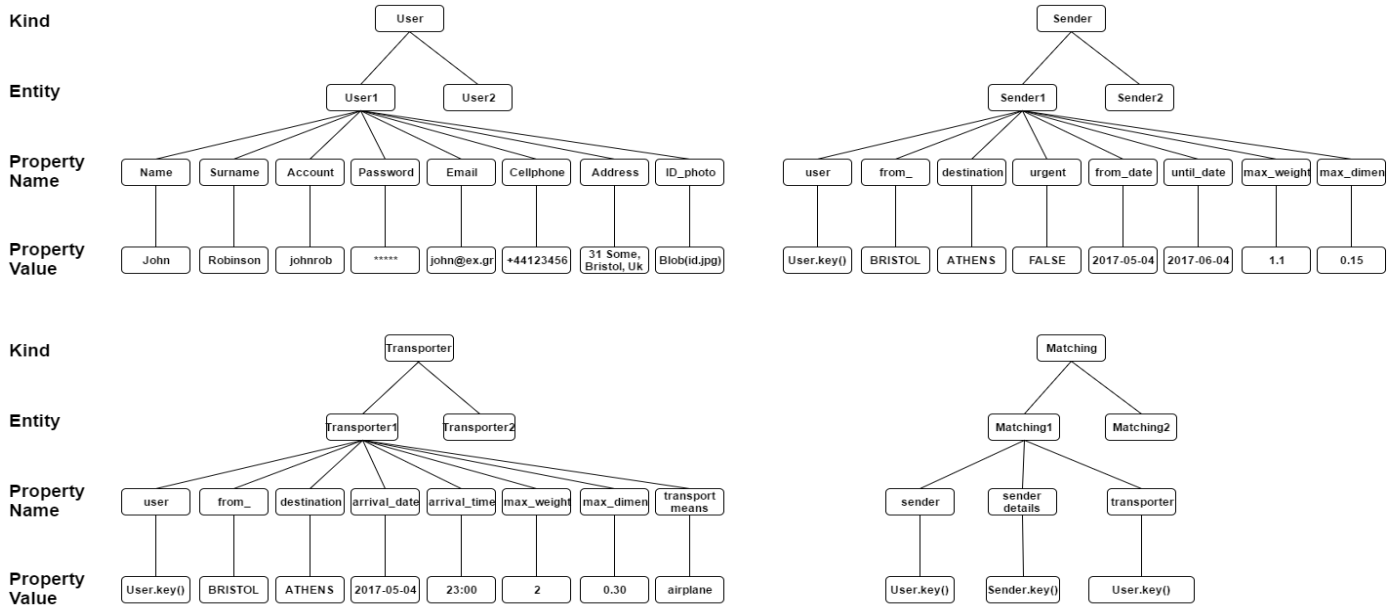

Fig. 2. Datastore Models that are implemented in TransportIT

- Login Handler
  This controller checks if the login credentials are correct. This is done with a Datastore query, checking if the username and password exist in the database. If the login details are incorrect the handler gives a response to the end-user, asking him to put the correct details. With a successful login, the handler stores the username and the user's entity key in the session and then renders the main page of the application.

- Sender Handler
  If the Sender navigation button is clicked, the handler renders the Sender's page. In this template a user can declare the journey's details of an item he wants to send or receive. First, the handler checks if the dates are inserted correctly (cannot insert a date that is before current date) and then stores the input data in the Datastore. Finally, it allows the sender to search for transporters.

- Transporter Handler
  The transporter handler is responsible for storing a transporter journey's details. It also checks if an arrival date is inserted correctly.

- MainPage Handler
  After a user is classified as a sender or a transporter, a corresponding flag is stored in the session in order to save his current state. When the MainPage handler is called, it checks whether the user is a sender or a transporter. If a sender flag is true, then the handler searches in the Datastore for transporters travelling between the time period that the user wants to send or receive an item. If the user is a transporter, then the MainPage handler just gives him the choice to see his messages.

- Matching Handler
  For a successful search of transporters, the sender has the opportunity to choose the one that fulfills his needs. When he selects a transporter, the Matching handler creates a new matching entity that it has as properties: the sender's user key, the journey's details key and the transporter's key. If the sender chooses to cancel the deal, then the handler deletes the entity and returns to the previous page. If there is a confirmation, then the sender can see in his history notifications the contact details of the transporter.

  If the current state of the user is transporter, the Matching handler makes a query, searching for a matching entity that has the transporter's key. If the query is successful it shows to the transporter all the current matches from senders that chose him with their contact details.

- SenderHistory Handler
  This handler is responsible to search the Datastore for matchings that have the sender's key and displays their details to the notification screen.

- ChangeDetails Handler
  This handler is responsible to update the user's account and contact details in the Datastore. It also prompts the user an error message if an account name already exists.

- LogoutHandler
  The logout handler deletes everything that is stored in the session and renders the login page.

B. *Front-End*

The front-end side of TransportIT was written in HTML5, CSS3 and Javascript (in some cases we also used jQuery library). The webpages were created without using templates in order to be able to customize the application needs.

All the pages created are stored in a templates directory and the style and images used by them (CSS3 and png files) are stored in a static directory within the application. The advantage of placing the style files and images in this directory is that it does not use the main program (index.py) for serving these files and ergo we avoid processor usage on serving static content. Even more importantly, by indicating in app.yaml that these files are stored in static directory, we enable Google App Engine to distribute them to many different servers geographically and leave the files there. In this way, these files can be retrieved from servers closer to the user of our application. Thus, our application can scale to far users more efficiently.

TransportIT consists of 6 webpages briefly described below:

- loginIn.html
  This is the page where the user lands on when first visiting the application. It includes a link to the signUp page and it can receive an error message from the main program (*index.py)* if the user does not provide the correct credentials.

- MainPage.html
  This is the page that the users are directed to once they have logged in. It is responsible for showing to each user the notifications related to the results of the queries realised in the Datastore. Furthermore, it contains links to senderSignIn and transporterSignIn pages.

- signUp.html & changeDetails.html
  These pages are used to ask from the user to input details related to the account. They both include specific HMTL form input fields in order to POST data. These pages also include diagnostic error messages from the main program in order to inform the user if an account with the same username already exists.

- senderSignIn.html & transportertSignIn.html
  These pages are used to classify the users as senders or transporters and ask them to store the data related to the item they are able to send/transport and the date they are able to do so. Data are stored in the Datastore from these pages as well. Additionally, they include Javascript in order to manipulate the format of the data inserted by the users and fill automatically some input fields. Ultimately, senderSignIn.html contains buttons that trigger the queries responsible for finding the suitable transporter for the sender in question.

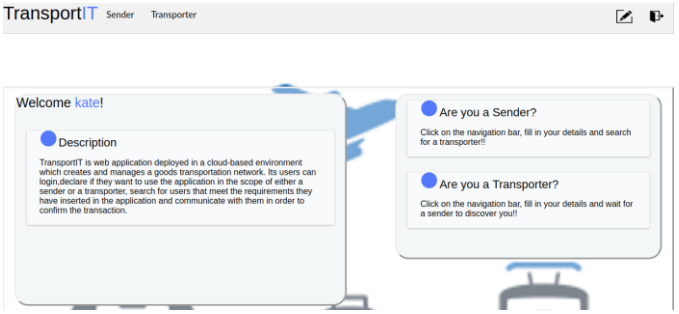

Fig. 3. Home page

V. USING TRANSPORTIT

When users first visit TransportIT they land in log in page and they are given the options to sign up or log in to the application. The users that use the application for the first time must sign up and declare their account details. We also ask from the users to upload a photo of their ID card. This photo could be used as a credential in order for the application's administrators to validate the data provided during the signing up procedure. This step is not implemented yet.

After logging in in the users are presented with a main home page that specifies the way they can use the application and contains a navigation bar. This navigation bar offers to the users the ability to be classified as a sender or as a transporter, change the details of their account and log out from the application. For security reasons users are not allowed to change their Name, Surname and ID card photo after signing up for the first time.

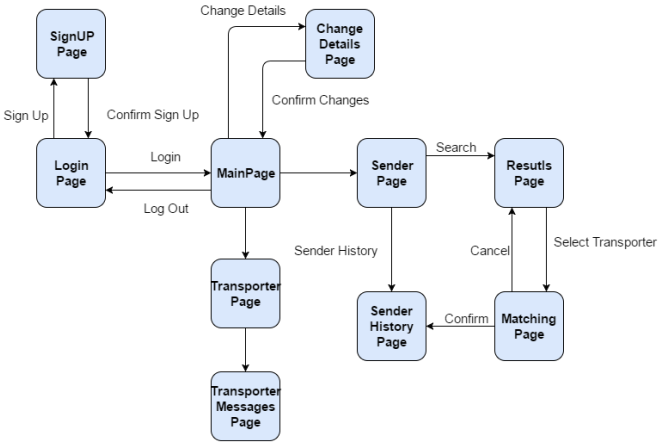

Fig. 4 Schematic diagram of the application's web pages' structure

A. *Being a Transporter in TransportIT*

Transporters in TransportIT have to specify the duration of their trip and the maximum space they can offer for exploitation. The journey's start and end point must be filled in capital letters. They store the data they provide by clicking the button at the end of the page and they get redirected to the main page of the application. At this point they can check for messages from senders.

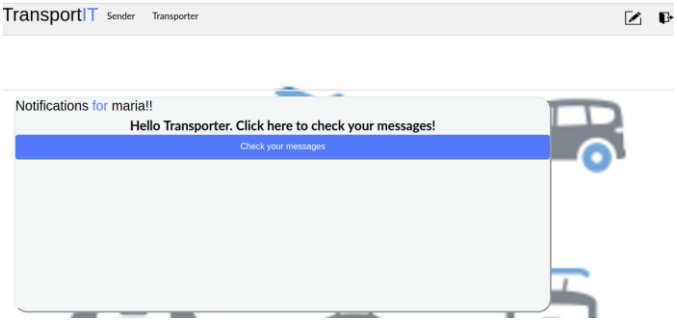

Fig. 5. Transporter's main pages

B. *Being a Sender in TransportIT*

Senders in TransportIT have to declare a time period in which they prefer to send/receive an item, the journey of the item and its dimensions. The journey's start and end point must be filled in capital letters. They store the data inserted by clicking the button at the end of the page. Then they have the option to search for a transporter traveling during the declared time period. If the search is successful the sender is redirected to the main page and presented with the results, i.e the compatible transporters. From this point the senders are able to select one of the transporters, confirm the deal and access the chosen transporters' contact details. The deal is stored in the sender's history section which is available at all times.

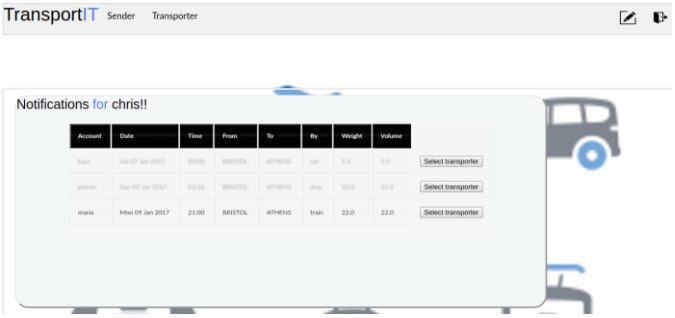

Fig. 6. Sender's search results.

VI. SCALABILITY AND LOAD TESTING

By the term scalability we refer to the ability of the application to handle robust traffic. After carrying out a test provided by Google App Engine [8] we can infer that TransportIT can handle the increasing demand in both computing power and data storage. Due to the fact that the application requires authentication we cannot test all the pages mentioned above. By testing the login page, we get that 95% of the requests were responded in under 35ms.

Fig. 7. Latency behavior during load testing

Ordinarily when a user makes an HTTP request, Google App Engine creates a new instance of the application the request is related to in under a second, responds to the request and then terminates the instance. If the request is not completed in 60 seconds, then the instance is killed anyway. Overall, TransportIT behaves as expected while handling exponential increase in requests by creating new instances.

VII. FUTURE WORK

We propose the following improvements regarding the content of our application:

A. *Security*

Many security issues can be caused if the users do not provide their true credentials. In order to solve this kind of problems we should verify the credentials of each user by using the ID photo uploaded by them. The users should not have the ability to log in to the application before we allow them to do so.

B. *Google Maps*

By incorporating Google Maps in our application, we will be able to use an interactive way to show the transporters' journey. We will also be able to demonstrate to the users the recent journeys that have been completed and collect information about the most frequent journeys.

C. *Email*

By permitting to senders to automatically create emails and send them to transporters in order to confirm the transaction, we will manage to make TransportIT more interactive and responsive. This will also require that we make sure that the users' email addresses, provided while signing up to the application, are accurate.

D. *Motivate the users by introducing a billing system*

Transporters should be rewarded by senders based on their performance. Paying a small amount of money for each successful transaction will be crucial to reviving the interest of its users. The fees related to each transaction will be smaller than the fees imposed by most companies because TransportIT is supported by a users' network. Furthermore, we can also introduce a rating system in order to distinguish the most reliable transporters.

VIII. CONCLUSION

In this paper, we discussed the implementation of a Cloud-based web application named TransportIT. This application aims in creating a users' network that will communicate on the basis of transporting goods. The application is hosted by Google App Engine infrastructure. TransportIT takes advantage of the Datastore to create a scalable database and uses the computing power of the cloud to connect users around the world. In addition, this application has been tested with a significant amount of requests during 15 minutes from 10 separate threads. To use the application in its full functionality visit https://transport-it.appspot.com with a Google Chrome web browser.

REFERENCES

[1] Design for scale, http://cloud.google.com/appengine/articles/scalability

[2] NIST definition, https://www.nist.gov/node/568586?pub_id=909616

[3] Severance, C. (2009). *Using Google App Engine* (1st ed.). Beijing: O'Reilly.

[4] Datastore API, https://cloud.google.com/appengine/docs/python/datastore

[5] Datastore Model, https://cloud.google.com/appengine/docs/python/datastore/modelclass

[6] Webapp2, cloud.google.com/appengine/docs/python/tools/webapp2

[7] Sample appengine apps, http://examples.oreilly.com/9780596801601/

[8] Joe Gregorio, *Getting Started load testing your application, 2009.*