GROUP 16 FINAL DESIGN REPORT

Date: 1st of November 2019

Authors:
Can Ölmezoğlu
Batu Budin
Joris Jonkers
Buğra Veysel Yıldız
Önen Ege Solak

INDEX

STAKEHOLDERS:

Stakeholders	**Needs/expectations**
Event organizer	Secure, unique tickets that can identify attendants,receive feedback
Service members	Easy of use, fast operation, accessibility of the database from any network, commenting on events, event information
Ticket shops	Reliability and security
Programmers	Money, clear requirements and good grades
Owner	Improved ticket revenue, low service costs, high satisfaction from customers, reliability
Security personnel	Identity verification, high throughput, data on event demographic

Event Organizer : Normal Operator. Because event organizers are Interacting with the system to manage the event.

Ticket shops : Normal operators. Because they are also interacting with the system by reaching guests

Service members :Functional Beneficiaries They are the guests that receives service from the normal operators by our system.

Programmers: Maintenance Operator. Creates services and repairs the product.

Security personnel: Functional Beneficiaries,Normal Operator. Security Personnel works at the events and uses the software to check ticket validity, the number of attendees and whether attendees have previously attended events with disorderly conduct.

Owner: Sponsor.The owner provides financial support and managing services for the other operators.

TECHNICAL REQUIREMENTS:

1. - read RFID tags using an Arduino and RFID reader.
2. - securely send the UID of an RFID tag to a computer.
3. - compare the UID of an RFID tag to UIDs stored in a database to determine whether it is authorized.
4. -Add the UID of an RFID tag to the database or remove it.
5. -Prevent people from copying the RFID tag.

THE REQUIREMENTS AS USER STORIES:

MUST:

As a guest, I want to get into the event with my RFID tag after purchasing a ticket online
Test if RFIDs UID is in the database
Test if associated ID has a ticket for the event
Test if the online purchasing system works

As a guest, I want my personal details to be secure against attacks to steal my account data.
Test if transmissions to the server are encrypted
Test if postgresql server is password protected
Test if passwords are encrypted while stored

SHOULD:

As a guest, I want to be able to contact the support service of the event, when I have questions about or problems with the tickets or my RFID tag.
Test if there is contact information on the website

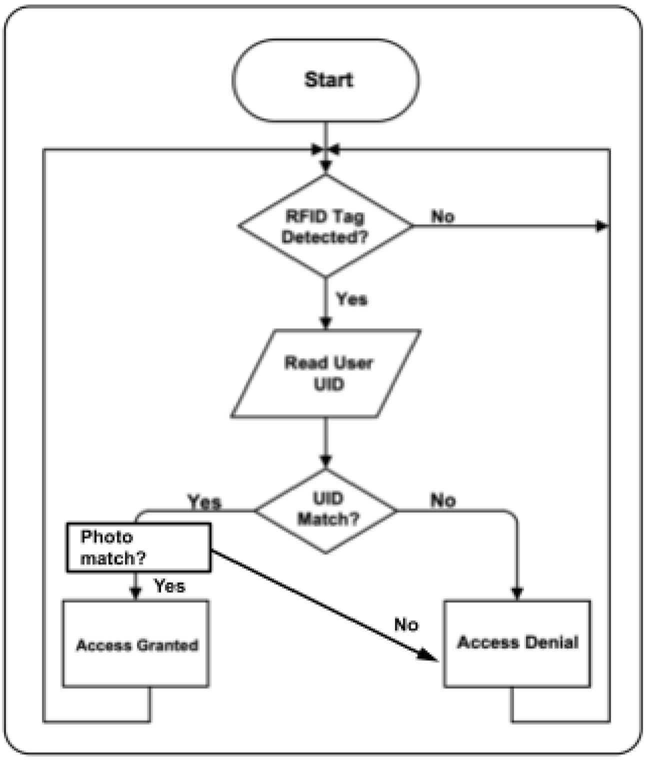
COULD:

As an online ticket shop, I want to be able to easily integrate the new ticket format.
Test if the database can be easily integrated with external programs

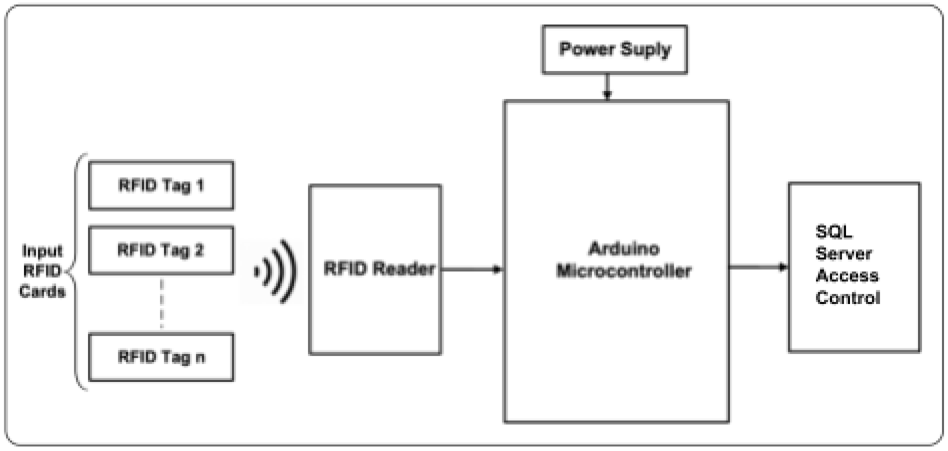
TECHNICAL DESIGN:
Description of the design:
The users will be asked to register on a website and will receive an RFID tag. The RFID tag's unique ID will be matched with the user. We intend to use unique the unique IDs of the RFID tags to distinguish users from one another. The unique IDs will be stored on our postgreSQL server. Unique ID data will be read from the RFIDs via our Arduino, which will then be compared with the unique IDs in the SQL server. If the unique ID exists in the server and has a ticket, the person will then be allowed to enter the event. In terms of challenge-response authentication, we will will pull up a photo of the tag owner when a tag is scanned so that security can verify that the owner of the RFID tag is the same as the holder. This will ensure people will not be able to enter an event with tickets that they do not own.

DIAGRAMS:

Flow Chart



Hardware Chart



Description of the design compared to the requirements:
- read RFID tags using an Arduino and RFID reader;
“THE RFID will be read by the Ardiuno.
- securely send the UID of an RFID tag to a computer;
We will have challenge-based authentication with photos.
- compare the UID of an RFID tag to UIDs stored in a database to determine whether it is Authorized;
The database will be encrypted in a server with a backup server so it will be secure.
- add the UID of an RFID tag to the database or remove it.
The database will be in a server where SQL can be used to do the aforementioned tasks.

THE SECURITY:

Potential Attacks:

SQL Injection:

The database is protected against SQL injection attacks from the website by Django’s security protocols. “Django’s querysets are protected from SQL injection since their queries are constructed using query parameterization. A query’s SQL code is defined separately from the query’s parameters. Since parameters may be user-provided and therefore unsafe, they are escaped by the underlying database driver.”

An SQL injection attack to the database via the ticket checker is prevented by giving the connecting the python code to the SQL server via a user account which has read only rights to the three relevant tables, hence an SQL injection cannot be performed.

DDOS attacks:

For distributed denial of service (DDOS) attacks which make it impossible to connect to a server, we have a backup server for our website which makes sure the website comes back on when our main server is DDOS’ed.

On the other hand, since our website has no automatic protection against DDOS attacks and we haven’t placed limits on request for registering, someone could send millions of requests and our servers would not be able to cope and would become unreachable. Similarly someone could upload an extremely large file in the upload profile picture section and make the whole server unreachable.

Cross Site Scripting:

As Django is open source and is used by developers worldwide, features have been added to make it secure against most cross site scripting attacks. In addition, HTML is not stored in our database, which prevents hackers from disturbing it when we would have retrieved it.

Hackers stealing the database:

The database is not stored locally hence an action like stealing is rather unexpected since hackers do not really steal server hard drives from datacenters.

The database has one administrator with permissions to edit the database which means there are fewer accounts that can be hacked. The database also cannot be accessed without a password and a valid username, hence a hacker cannot easily access the database.

Hackers stealing a pass:

If a person loses their pass physically, they can use the contact part to prove their identity and re-order a new pass.

On the other hand,to prevent a hacker from digitally reading a pass and then duplicating it,our system uses an extra picture verification step, when an RFID tag is scanned we retrieve the user’s profile picture in order to compare it to the holder of the RFID tag meaning that even if a UID gets copied it still can’t be used by a third party to enter an event.

Man in the middle attack:

We wanted to use SSL however it needs to be done at server creation and we didn't have enough time to create a new server and copy our data over, leading us to be vulnerable to MITM’s.

User data privacy:

The passwords are secured using the one way encryption of Django. Django provides a flexible password storage system and uses PBKDF2 with an SHA-256 hash by default.

TECHNICAL IMPLEMENTATIONS

TECHNICAL IMPLEMENTATION OF DJANGO:

- We choose Django for our GUI because Django has compatibility with many operating systems and provides good security for the websites. It also has excellent support from the internet community.
- We got our webpage framework from Medium.com. Getting help from the website, we created a folder and virtual environment for Django. Following that, we installed Django in that virtual environment and started our Django project.After that we created additional folders like templates,media and statics and migrated them using models.py into a SQLite 3. After that, we modified the project files and created some files.
- We followed the instructions that they provided for the registration,login and the admin parts and then we found a theme from Bootstrap and implemented the code from the files that we downloaded. Following this, we personalized the template by adding photos of our own and we also added our contact information.
- We also erased the pictures that they provided from the static/img file to add our events brochures. But the problem was that we need to make the computer read the images from the static file.So we write {% static '.....' %} to the images and vendors in the templates/index files to inform the computer.We also set our database in the database code in the dprojx/settings.py file.

TECHNICAL IMPLEMENTATION OF THE READER:

The RFID reader that we programmed on Arduino defines the UID's of the tags that we used in the project. We made use of serial to read data from the serial port to which the arduino was attached. We then wrote a function uniquot which retrieves the UID information from serial port and turns it into a readable string or a boolean False if the UID information is not of the expected format. This function waits for the arduino to return a string via the serial port. Then we connected to our postgresql server using psycopg2. We connected psycopg2 with our host name, password, host number, port number and database name. For the host name and password we used our readonly account which can only read and not change the relevant tables for our checker program. Next we define the function "RetrieveID". This function takes the ID that was retrieved from the RFID tag using the previous function if this returns False RetrieveID will also return False. It compares the RFID to the RFID's in the relevant table in our database (public.auth_user), if the RFID code is not in the table it fetches an empty list. In case it fetches an empty list it returns false as the RFID is not registered and therefore the tag must be invalid. If the RFID is in the table RetrieveID returns the ID that is associated with the scanned RFID. After this we defined the final function, this function checks whether the combination of the event ID and user ID exists in our attending table in our database if it is the function returns True and the user’s ID otherwise it returns False and 0. We then made a GUI using easygui, the first thing our GUI asks for is the event ID which is stored and compared when we call upon RetrieveID. We then run RetrieveID with the specified event ID, if it returns true we pull up the location info of the profile picture associated with this account and show this picture so security personnel can check if the correct person is using the ticket. If RetrieveID returns False our GUI shows an error “Tag Incorrect or no ticket owned”. It then loops back to the beginning of the GUI program allowing for the next person to scan their tag starting the process over again.

TECHNICAL IMPLEMENTATION OF THE DATABASE:

We made use of Django to create our webpage, by default Django uses SQLite for storing data, however we decided to use a PostgreSQL server as our database. We chose PostgreSQL instead of SQLite because its more flexible and could be accessed by many devices. It’s also open-source software which means it’s free to use and in this case has extensive support. The other reason for choosing PostgreSQL was it’s convenience.

To implement PostgreSQL, we changed Django’s settings and gave it administrative access to our PostgreSQL server by writing the following in our settings.py file:

```
ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'postgres',
        'USER': 'postgres',
        'PASSWORD': 'Keyboard25',
        'HOST': '130.89.229.53',
        'PORT': '5432',
```

We then migrated our database by using the functions **makemigrations** and **migrate** function after which we had finished integrating the PostgreSQL database.

DISCUSSIONS:

Django:

Django is good for editing because it is simple to use and logical in its operation, however while we were working with it we were limited to our theme and the templates that we could find on the internet, as we did not have enough experience or time to create a template ourself, which was not ideal as the templates on the internet did not suit our purposes.Secondly we realized that we haven’t limited the maximum number of requests from a specific address which makes us vulnerable to DDOS attacks. Which can cause our server to crash when faced with such an attack.

At the beginning of the project we lost a bit of time as we used kivy initially instead of Django, however kivy proved to be too complex for our timeline as such we ended up swapping over to Django.

Reader:

Whilst working with serial at first we had some difficulty with retrieving information in the right format. The information we retrieved from the serial port would consist of a list of UID's starting with '\n' and ending with '\r' however we knew that the correct UID's would start with '\n' and at the 13th and 14th position it would have '\r' as the code itself was a string 11 long in the format "xx xx xx xx". We also ran into trouble with kivy which is what we used for our initial GUI as we only needed a simple GUI for our demo we instead used easygui which is incredibly easy to use. The only downside is that there are only limited options.

SQL database:

In order to be able to access our data from anywhere, which is one of the most important requirements as events can be held anywhere. We had to store our data in a server which is accessible from anywhere for this we initially used Microsoft SQL server however we ran into trouble accessing this server from other devices and so at the advice of a TA we swapped to a postgreSQL server. This server was easy to set up and use, we also made use of pgadmin to quickly edit the database and access tables. If we were to undertake this project again we would create a postgreSQL server with SSL activated as we now do not have SSL encryption allowing for data to be intercepted in transit luckily we only transmit passwords when encrypted.

WORKS CITED:

- **DigitalOcean. "How To Use PostgreSQL with Your Django Application on Ubuntu 14.04." *DigitalOcean*, DigitalOcean, 18 Sept. 2019, www.digitalocean.com/community/tutorials/how-to-use-postgresql-with-your-django-application-on-ubuntu-14-04.**
- **Sharma, Himanshu. "How to Create -> Registration + Login Web App with Python Django 2.0." *Medium*, Medium, 28 June 2019, medium.com/@himanshuxd/how-to-create-registration-login-webapp-with-django-2-0-fd33dc7a6c67.**
- **Ez, Orji, et al. "Journal of Engineering and Scientific Research." *Automatic Access Control System Using Arduino and RFID*, 2018, doi:10.23960/jesr.**