**Team members:** ALTIN CIPI (M1309), VASILIS SKOURTIS (M1341), EYAGGELOS KARAGEORGOS (M1364)

**Project title:** E-voting android mobile application + system backend (VoteForIt)

**What we are doing to carry out the project:** This class is very demanding in terms of effort and time spending due to its weekly paper reviews. We also have a very large workload from other courses, let alone the fact that all three members of the team have full time jobs. As a consequence we have fallen behind with our schedule. We were to have finished the development of our two main applications (mobile app and backend) by the start of June but from what we calculate that is not possible now. Our new estimation is that we will finish development by 20th June and then we will have 10 days in order to do testing, evaluation, the development of our own load balancer and database replication implementations and the final presentation and demo. That is not enough time because in June we have our final exams, that is why we are asking for a deadline extension (Please!!). At this point we have made all the initial configuration and setting for our system and began its implementation. We have set up a list of tasks for every member to do and we are working on them. We have a skype meeting every other week to coordinate our work. We demonstrate below what we have done so far, what we are doing now and what remains to be done as well as what we decided that we will not do.

**What has been achieved thus far:**

* We have created an email account at Gmail for our application ([ds2015.dit@gmail.com](mailto:ds2015.dit@gmail.com))
* We have created an account at Amazon Web Services with the above email.
* We have created an EC2 instance (t2.micro, 1CPU, 1GB RAM) in Ireland Availability Zone. We have installed in it Java 1.7.75 and Apache Tomcat 8.0.15. Its Public IP address is 52.17.140.15. So if you open a browser and click <http://52.17.140.15:8080/> you should be able to see our Tomcat’s welcome page. (\*)
* We have created our Amazon Elastic Load Balancer which we have set to listen on ports 80, 443, 8080, 8443. Its DNS Name is ds-di-180488603.eu-west-1.elb.amazonaws.com. So if you open a browser and hit <http://ds-di-180488603.eu-west-1.elb.amazonaws.com:8080/> you should be able to see our Tomcat’s welcome page. (\*)
* We have also created from this site <http://www.selfsignedcertificate.com/> a self-signed SSL certificate for the domain name [www.voteforit.com](http://www.voteforit.com). We have attached this certificate to our load balancer mentioned above on his ports 443 and 8443 so now when a client makes an HTTPS request to our load balancer the communication will be SSL encrypted and thus safe from man in the middle attacks. But because this is still a self-signed certificate the client must accept it first. We plan to purchase a valid certificate from a trusted CA later. So if you open a browser and go to <https://ds-di-180488603.eu-west-1.elb.amazonaws.com:8443/> you will get a warning that says that our certificate is not valid because it is self-signed for [www.voteforit.com](http://www.voteforit.com). Add an exception and proceed and you will be able once again to see our Tomcat’s welcome page. (\*)
* We have created an AWS S3 bucket which is a scalable storage system provided by Amazon. We will use this bucket to store user’s photos and other media that may be needed. We do this in order to free a lot of space from our database (the other option would be to store the photos as byte arrays inside the database or to store them in the file system of our EC2 instances) and to balance the load between our backend components. Sometimes in order to have better scalability you have to put as many different components as possible that do different jobs and not have a single type of component do all the jobs. So with this implementation when a user uploads a photo it goes to our backend server, the backend server uploads the photo to the S3 bucket, S3 returns a URL for the photo, our backend server stores the URL in the database and when the user requests the particular photo from our backend server, the backend server retrieves the URL from the database, gives it to the client and then the client downloads the photo from the S3 bucket using the URL. Our buckets name is “dsdi” and it is not located in a single region as Amazon maintains replicas all around the world and serves user requests based on their location. You can download a photo that is uploaded to our bucket using this URL: <https://s3-eu-west-1.amazonaws.com/dsdi/avatars/1_acipi.jpg> . Our bucket is read-only for everyone except for authenticated users. So you cannot put/delete anything to/from it. Our free tier storage quota is 5GB.
* We have created our primary database RDS instance (db.t2.micro, 1CPU, 1GB RAM, 20GB SSD). It is a PostgreSQL 9.4.1 database and is located in Ireland. The database’s name is “ds” and we can connect to it using this endpoint ds.carpf7xinhv2.eu-west-1.rds.amazonaws.com:5432. We cannot give you the credentials to connect for obvious reasons. We have enabled automated backups, so every day a backup of the database is created and the latest 7 backups are maintained if something goes wrong and we want to restore to a previous point in time. We have not created any Read Replicas yet because we have not implemented the replication system. We plan to do this in the near future.
* We have created a Virtual Private Cloud. We put our RDS and EC2 instances inside that cloud and have applied security restrictions to it. So no one from the outside world can communicate with our database and backend servers unless we allow it in our security group. The components that are inside the VPC can communicate with each other. Now we have opened some ports in our security group to allow testing but when the app is released every port will be closed and the only way to communicate with our backend servers will be through our load balancer, and the only one that can communicate with our database will be our backend servers.
* We have created an authenticated user named “ds” and we have given him full access privileges to the S3 system. We have created access keys for that user. Now our backend servers use those access keys in order to connect to S3 and upload/delete user photos.
* We have created the DSBackendServer webapp in Eclipse Luna. It is the web application that provides the REST Services to the clinets. It is a combination of 6 projects: ds-dao (for providing DAOs and Services), ds-dto (for providing the common Data Tranfer Objects between the server and the android app), ds-model (for providing the model objects of our system), ds-pom (the Maven parent pom for all the other projects), ds-utils (for providing various utilizations used by all the other projects) and ds-web (the actual web service that provides the Rest services). So far we have implemented functionalities for the Users. We have uploaded the web app in our EC2 backend instance and all the rest services can be accessed through our load balancer.
* We have created the DSAndroidApp project in Android Studio. It is the actual android app that our clients will use. So far we have implemented the Login Screen, the Registration Screen and the Home Screen. We have installed the application in our mobile apps and we are testing it. The application communicates with our load balancer and uses the rest services that our backend server provides. If you want we can demonstrate the application live in class for you.
* We have implemented a session token technique on every REST service request. Now on every request the client must provide his username and his valid session-token in order to get authenticated and to receive a response from our backend server.
* We have implemented password advanced encryption and salt hashing for better security.
* We have created a repository in acipi’s account in Github and we have uploaded all our code there. The repository is this <https://github.com/std08010/ds2015> . You can visit the repository since it is public and view our code (so you know that we are not lying about what we have implemented so far :P).

**What is in progress and how it is going:**

* Development of various screens in the android app like: user profile, upload a voting subject, Browse through all voting subjects, vote for a voting subject, subscribe to another user etc. We are currently in the process of designing these screens while simultaneously developing some draft implementations. We should be finishing this by the end of the month.
* Development of the REST Services in the backend webapp that will support all the functionalities needed for the above mentioned screens. We are currently in the process of building our model objects and developing all the services needed for them. We should be finishing this by the end of the month.

**What is not done at all:**

* Development of the tool that will trigger a very large amount of client requests to test the Amazon ELB.
* Development in the backend webapp to support Read Replicas.
* Development of our own load balancer.
* Development of our own replication system and database load balancer.
* Purchase a domain name for [www.voteforit.com](http://www.voteforit.com) and configure the Amazon’s DNS servers to route all requests made on this domain to our load balancer.
* Development of various test scenarios.
* Composition of our final presentation and demo.

**Changes made to the previous proposal:**

* ~~At the end of a voting they participated in they will get a notification of the voting results.~~
* ~~Users will have points that will increase each time they participate in a voting.~~
* They will also be able to initiate a voting subject ~~if they have the necessary amount of points~~.
* ~~Voting subjects will also be categorized.~~
* ~~The final android app will be uploaded to Play Store.~~ (We could upload a beta version of our application but since we are perfectionists we don’t want the application to receive bad reviews and get burned. We will upload a more stable version of the application later if we decide to continue developing it.)
* ~~Also SLL might be used between the communication of the servers and the database instances~~. (no need for that since we are using a virtual private cloud that no one from the outside world has access to)

(\*) Because AWS Free Tier gives us approximately 750 hours of operation/per month for the EC2 instances (which is how much a single instance would consume in a month if it was always operational) we leave our primary EC2 instance closed when we are not doing testing, so that later on we can create more than one instances for free and test the scalability of our program. So if you want to test the above mentioned URLs you can send an email to us (for example [acipi@di.uoa.gr](mailto:acipi@di.uoa.gr)) and we can start our main EC2 instance for you to do your checking.