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

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

**Project description:** We will develop an android mobile application that will be used for e-voting. Users can register and login to the application. Then they can browse all the voting subjects that are available at the time and cast their votes on the ones they want. After they cast a vote on a subject they cannot vote again for it. Users can also view the current results of each voting. 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. Users will also be able to subscribe to other users and vote for their voting subjects. The final android app will be uploaded to Play Store. Possible names are: VoteNow, Vote4It, VforVote, The Vote, Votebook, Votify etc.

**Project Characteristics:** We will develop the android mobile application with Android Studio and we will test it in our own mobile devices. Since this is an online application there is a need for a backend system and a database system.

In this project the two main class distributed system issues/challenges that we will tackle are the above:

**Performance / Scalability (*Cluster of Machines / Load Balancing*):** Our backend system will be consisted of multiple equal servers that all of them together will make a cluster. Therefore we will develop a java web application that can be distributed across multiple nodes and we will deploy it on each one of our servers. In order for our web application to be distributed we will use REST services that will provide all the needed functionality to the mobile clients. For this web application we will also use technologies like Spring, Hibernate JAX-RS etc. We believe that our system if implemented this way can scale up to a very big number of mobile clients and will have a very good performance. We will use AWS EC2 instances as our servers. We can have as many EC2 instances as we want (our implementation does not forbid this) but because acquiring those instances costs money for the purpose of this project we will use around 4-5 servers. That should be enough to prove that our system is distributed. In addition we want this cluster of servers to be as transparent as possible and appear to the mobile clients as a unified backend system. To achieve that we will use a load balancer. The load balancer will be the single point of entry to our backend system (all the clients will send their requests to it) and it will distribute all the client requests equally across all nodes in the cluster. This will increase performance and scalability. There are two types of load balancing that can be achieved. The first is an edge load balancing where the load balancer sends a client request to the cluster center that is geographically nearest to the user in order to increase the response time. We will not deal with this kind of load balancing. The second one that we will tackle is the software load balancing. Here the load balancer takes a client request and forwards it to the server that has the smallest workload. This increases system availability and performance as well as it improves fault tolerance because if a server fails then the load balancer will exclude it from the list of available servers to forward the request to until it is operational again. On the downside there is a single point of failure because if the load balancer crashes then the servers cannot receive client requests. In order to implement the load balancing we will use two different approaches. The first approach and the easiest one will be to use the AWS Elastic Load Balancer. This is a commercial load balancer the Amazon provides and it can be used with the EC2 instances mentioned earlier. Our job here will be to learn how to use and configure this tool for our project and how to properly evaluate it. That is find how much it can scale, how does it react to a heavy workload and how it handles failure. To do this we will develop a tool to trigger a very large number of client requests. The second approach will be to implement from scratch our own load balancer. We will develop a Java application that will receive client requests, ping all the available servers from its servers list, count their response times, pick the one with the smallest response time and forward the client request to it. We will later deploy this application to a separate EC2 instance that will serve as our load balancer. Furthermore, in order to deal with failure we will develop a system that will deploy a backup load balancer if the primary one fails.

**Replication (*database replication*):** The database used will be PostgreSQL. As a matter of fact we will use an AWS RDS instance as our database. The servers will read and write data to the database. Clients will have nothing to do with the database. To increase reliability but also performance when a client wants to read some information from the system, a database replication scheme will be implemented. We will have multiple RDS instances. Each one will be a replica of the other and all together will form a database cluster. All read requests will be distributed evenly across all replicas to increase performance. When a server has to perform a write it will write to all the instances in the cluster so that every instance is up to date. This will increase reliability. Fault tolerance will also be increased because when a database instance fails there will be another one to handle the servers’ read/write request. To implement database replication we will again use two different approaches. The first approach like before will be to use the existing AWS RDS Read Replicas system. With this system we will have a single master RDS instance that will serve as our main database. Every server will write to it. We will also have several RDS Read Replicas of this instance that will be automatically updated by their mother in order to have up-to-date data. Our servers will then distribute their read requests across those Read Replicas in order to relieve the master RDS instance from the read workload. Again our job here will be to learn how to use and configure this tool for our project and how to properly evaluate it. The second approach will be to implement the database replication from scratch. To achieve that we will have a number of RDS instances that will be equal. We will develop a system like a database load balancer that will take the write requests from the servers and forward them to all the database instances so that every instance has the same data. This system will then distribute the read requests evenly across the database instances. From our class experience we know that we will have to deal with issues like “What happens when an instance fails? Does it become stale when it is again operational?”, “What happens when the database load balancer fails?”, “What happens when a database receives two write requests from two servers in a different order from what it was supposed to receive them”, etc.

We will also tackle some other issues/challenges in a secondary level. Those are:

**Naming:** Our mobile clients will only know the IP address of our load balancer. It will know the IP addresses of our servers. The servers will know the IP addresses of our database instances. In order for this to be more user friendly for the clients we might buy a domain name for our application. Then we will use Amazons’ DNS servers to translate this domain name to the IP address of our load balancer.

**Communication:** The client/server communication will be with REST services. Since REST services are by nature stateless and we want the system to be as scalable as possible there cannot be any kind of stateful connections between a client and a server. Therefore there will be a session-token implementation technique for user authentication each time a client uses a REST service.

**Security:** We will use SSL between the clients and the load balancer to prevent MITM attacks. We might purchase a valid certificate, otherwise we will use a self-signed one. Also SLL might be used between the communication of the servers and the database instances. Or we might user VPN connections that are offered from the AWS. Password advanced encryption and salt hashing will also be used.

**Project Time Schedule:** The first part of our project will be to develop simultaneously the android application and the backend web application, since each one needs the other. We will start with implementing the basic features of the backend web application that we consider essential. We expect this to be ready by the end of April. After that we will start the development of the android application because it needs to use some already implemented services. As the android development continuous new requirements for the backend system will arise and will be implemented along the way. We expect both the web and android applications to be ready by the start of June. The second part of our project will be the deployment of our system and its evaluation. For this step we will use the first approaches from both load balancing and database replication as mentioned earlier. We expect this to be ready by mid-June. The third and final part of our project will be to implement the second approaches we mentioned earlier and test them. This will conclude our project and will be ready by the end of June.

**Project Demonstration:** We will demonstrate our backend web application with some UML Diagrams and a powerpoint presentation. We will also make some tests that we will run live. We will demonstrate the android application from a mobile phone as we will show all the features that the app has. If the app is ready to be uploaded to Play Store by the 1st of July then we will ask you to download it to your mobile phone and test it yourself. We will make a small presentation of AWS (EC2, RDS, S3, IAM, etc.). We will evaluate the performance of the overall system by running some use case scenarios and some failure scenarios as well (for example we will shut down on purpose the load balancer to see how the system reacts). Finally we will demonstrate the load balancer and the database replication system we have built from scratch and we will evaluate their performance using the previous scenarios.

**Project Teamwork:** We believe that the best way to divide the project is horizontally and not vertically. That is we won't divide it like this: “someone is doing the android app, another the web application and the other one the load balancer and the replication”. We will all participate equally in every part of the project, with everyone implementing small tasks across each part. We will make a repository and upload it to github where everyone will commit their work on a daily/weekly basis. We will have video conferences via Skype or Google Hangouts on the weekends in order to evaluate the work we have done in the past week and decide on the work we are going to do during the next week. Since all three of us work on full time jobs it will be very difficult to arrange close-up meetings. Our only close-up meeting will be on Wednesdays during and after the class. Vasilis and Altin work on the same company so when the time permits it they can discuss the project there. The ideal scenario is for everyone to work 1-2 hours per day on the project. But most of the work will be done on weekends. Each one of us attends to 9 – 12 hours per week on classes and 25 - 40 hours per week on work. So it will be difficult to afford many hours on the project. We will have to compensate with experience and good team coordination. Each one of us will have an equal responsibility on the project and no one will be in charge of the overall effort. There is no need for a project manager on a team of 3 members. Maybe Altin will be the coordinator of the android app development part since he has previous experience on the subject.