group-3-milestones.pdf

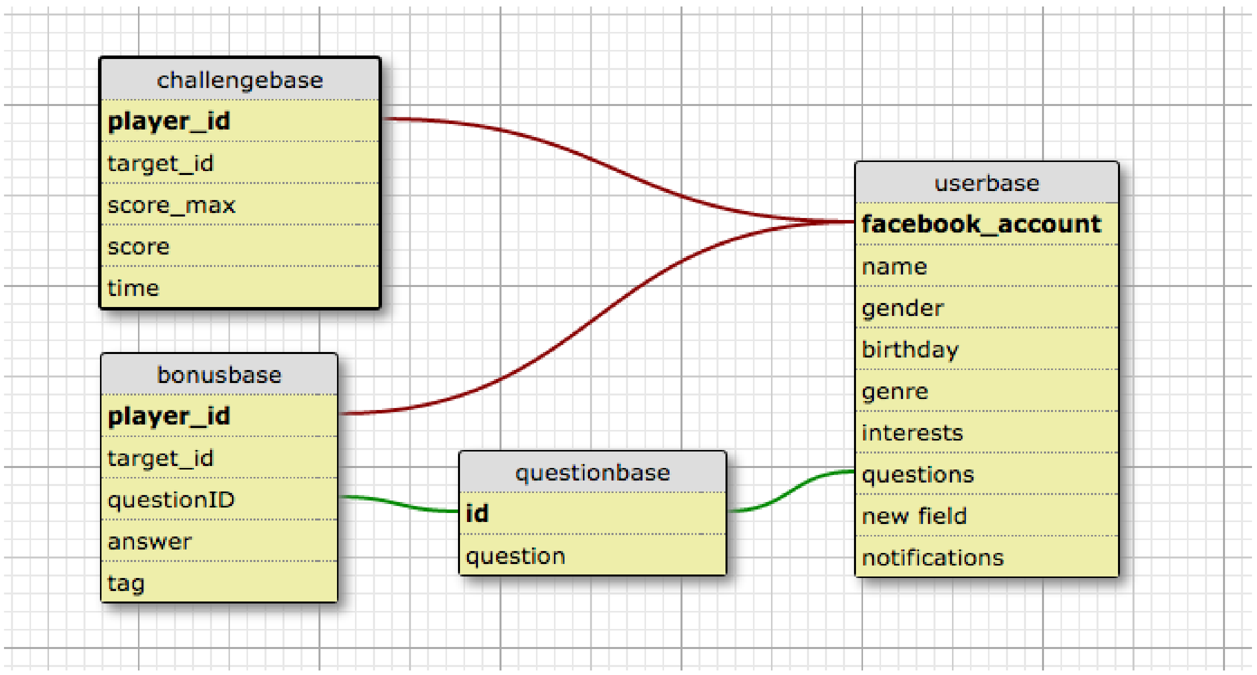
Milestone 0:
Friends become strangers, as technology develops. This shouldn't have been the case, since the social platform is becoming more accessible and convenient. However, with the huge information source, we began to know less about our friends and tend to communicate less with them.

Milestone 1:
Our application will let the users enjoy a simple and lightweight game, designed for them to explore and get to know more about their friends. The game is conducted in a way that users answers questions related with a chosen friends, and gain points based on the amount of correct answers and response time. The data library of the game comes from the input of the users, their facebook profile, as well as the "bonus questions" raised by their friends. The question related with a same friend would be random and different on every entry.

Milestone 2:
We are targeting everyone with a smartphone and a Facebook account, and those who would like to know more about their friends and share with the others more about themselves. By making the app available on phone devices and their app stores, people would explore and spot this fun game. Besides, the users will be login using their facebook account, which would allow them to share their game status and invite their friends to join the game.

Milestone 3:
We call this game "Quiz!t", pronounced as "Quizit". It is easy to see from the name that it's a quiz format!

Milestone 4: Draw the database schema of the app



Milestone 5:
Prominent request:
GET: `/quiz`. The function for this API is to get questions either answerd by the user or generated by the user profile.
The parameter is Facebook account: `/quiz?fb_account=”77777777777”`
Then the return value is an array of questions.

POST: `/challenge`. The API is used to send the quiz result to the server. Afterwards, the server will record the player, target and score information of this quiz.
The request body queries include playerID, targetID and the score of this quiz. like challenge playerID targetID score.
The return value is a confirmation message when the data is successfully stored.

GET: `/bonus` get bonus question for the target user to reply or the player to read the respond from the target user. These are intended to be displayed in the notification panel.
The parameter is userID like `/bonus?userID=”777777777”`
The return value is the bonus question related to the user, which also includes information like answer and ignore/unverified tag.

Our API does not require the client to know anything about the structure of the API. Rather, the server provides whatever information the client needs to interact with the service. An *HTML form* is an example of this: The server specifies the location of the resource, and the required fields. The browser doesn't know in advance where to submit the information, and it doesn't know in advance what information to submit. Both forms of information are entirely supplied by the server.

Milestone 6.
Query 1: In order to display a leaderboard in front end, the query is to find the top 10 students with the hightest accumulated scores.

```
  req.db.challengebase.aggregate(
   {
     $group: {
       _id: "$player_id",
       total_maxscore:{$sum: "$score_max"},
       total_quiz: {$sum:1}
     }
   },
   { $sort:{total_maxscore:-1}},
   { $limit: 10 }
   );
```

Query 2: Get a random document (question) in the collection (questionbase)
```
var randNum = Math.floor(Math.random()*count);
req.db.questionbase.find().limit(1).skip(randNum).next(function(error
, data){
     res.send(data);
});
```

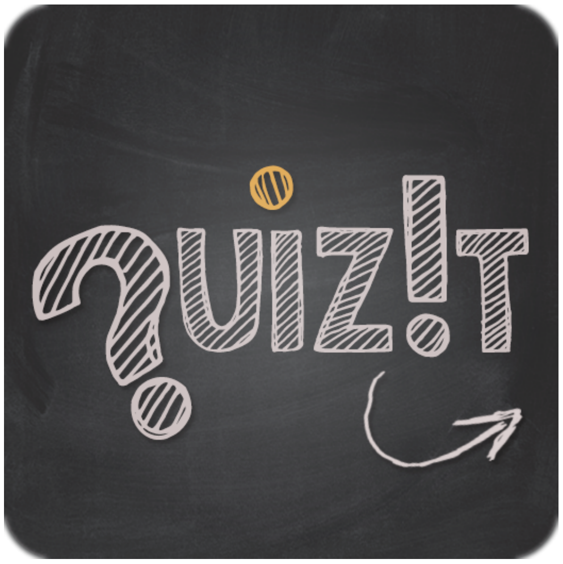
Query 3: Select the perople gets the highest score when they take quiz on the target user.
```
   req.db.challengebase.find({target_id: query.target_id}).
     sort({"score": -1}).limit(1).toArray(function(error,
challengebase){
     if (error) return next(error);
     res.send(challengebase);
   })
```

Milestone 7: Find out and explain what [QSA,L] means. Tell us about your most interesting rewrite rule.

The QSA flag means to append an existing query string *after* the URI has been rewritten
The L flag simply means to stop applying any rules that follow.

In our app, the URL matches “`^users/(.*)/player$`” (format: users/[userID]/player) will be rewrited to “`users/playerid=$1`” for GET request sent to the server.

Milestone 8: Icon and splash screen (ref to app)



Milestone 9:
The decoration of the app is simple, with not so much of fancy designs, yet not boring nor dull, that fits to the theme of a quiz game. Color of the app follows the background we chose as well as “yellow” and “tiffany blue”. The colors give it a style that is vibrant, young and interactive, and never lose the classy feel embedded with them.
The user interface is easy to use. There are easy linkages at the side bar, which could guide the users to the Quiz, Leaderboard, History, Notification pages easily, as well as the logging out button. The Challenge interface is designed in the format of a chatting window, such that it fits the situation better and more familiar for the users to interpret the questions. They would be all yes or no questions, and all a user needs to do would be tab on a tab!

Milestone 10:
When the app goes offline, the app offers some basic functionalities such as navigating between pages. Quizit will also skip the login page and load data for friends’ information, quizing history and leaderboard when the internet connection is down. Application Cache is used to achieve this. Moreover, we use localStorage from HTML5 to store the state of the application variables. This is crucial as, for example, we need the model of the history and leaderboard page in order to load the pages properly when offline.

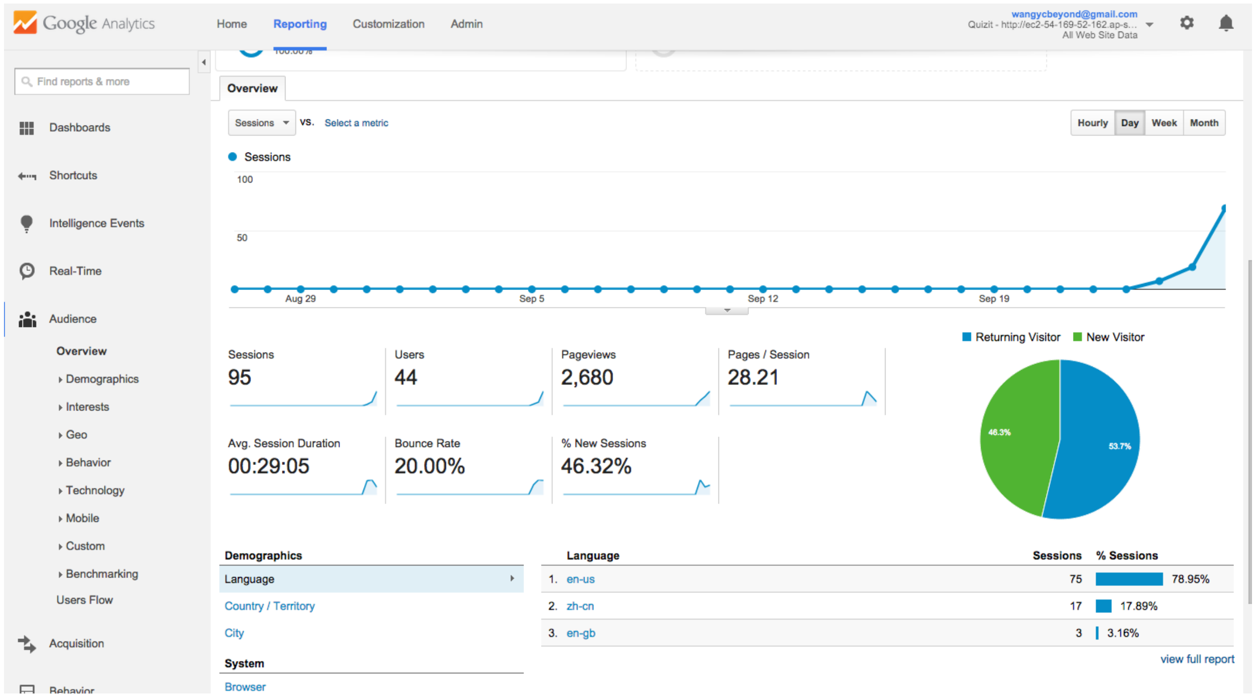
Milestone 11:
We didn’t think it is applicable to implement these syncing-related offline features because of the nature of our app.

Milestone 12:
Compare basic access authentication:
Basic:
pros: small, easy , quick, simple to implement, give decent security if used over SSL
passwords stored on server, can decide encryption method
cons: if used over non-SSL then it is open to man-in-the-middle attack
OAuth:
pros: many. Most social networks use this - popular. Easy to use, just “Sign in with...”. Save time, provide networking easily. Security, data transfers with OAuth 2.0 is over SSL.
cons: lack of anonymity. data can be obtained by app owner and they could misuse it for bad activities.
Cookies:
pros: fast, store session, no need to re-enter authentication
cons: cookies can be stolen and be used as masks for others
Digest:
pros: No usernames or passwords are sent to the server in plaintext => non-SSL connection more secure
cons: open to man-in-the-middle attack too. Not allow server to use custom encryption method => can be hacked
Our security scheme:
Facebook used OAuth, we have Facebook login on the client side.
We plan to have basic access authentication for connection between client and server. For the client side it is done but there are problems with the server, and we could not fix it until the last minute so we just give up on using it (better to have a running program).
Possible exploits (if for some reasons someone decided to attack the server): Get users data without their consent. Post fake data of users to the server, this will make the server database update. We do not provide a delete API though.

Milestone 13:

1. Users are able to search for their friends and choose the ones they want to answer the questions about.
2. Users are able to ask a bonus question to their friends, and receive notifications of these questions, which is more fun and more interaction among the friends.
3. Users are able to login, log out and re-login with their Facebook account.

Milestone 14: Google Analytics screenshot



Sessions	Users	Pageviews	Pages / Session
95	44	2,680	28.21

Avg. Session Duration	Bounce Rate	% New Sessions
00:29:05	20.00%	46.32%

Demographics
Language
Country / Territory
City

System
Browser

	Language	Sessions	% Sessions
1.	en-us	75	78.95%
2.	zh-cn	17	17.89%
3.	en-gb	3	3.16%

Milestone 15:
We integrate Facebook into our application. It is the main source for user account and our question base. We use basic information of the user from Facebook to generate some initial questions for the user. Facebook is chosen for our app because it provides us with the most useful information of the users and it has the largest user base.

Milestone 16:
We didn't use geolocation.

Milestone 17
For front-end, we use Angular.JS and ionic.js. This is chosen as it is native to HTML5 and the javascript environment that our app is built upon. It also allows a very high degree of code reuse. The CSS and JS functions ionic provides also simplify our job significantly and makes our app work smoothly. Ionic also provides us with the ability to easily deploy our app locally in the browser or even to the android device we have.

For back-end, we chose Node.js, Express.js and Mongodb. They are chosen because the requests of our app are all relatively light-weighted so it is faster and more efficient to keep them within javascript. Node.js project is also easier to test locally as we don't need to upload the files to the server every time in order to test. MongoDB is more efficient for this purpose as the data size is relatively small, and compared to MYSQL database, MongoDB integrates better with JavaScript because of its JSON-like query result syntax.