project: BITCamp Classroom

WHAT team: Nidal Salkic, Boris Tomic, Senadin Botic, Enver Memic, Becir Omerbasic, Medina Banjic

SPRINT 1 PLANNING

Main Site Functionality

1. Login (same for admins, teachers, mentors, students)
2. Post Login Form (same for all, only after first login)
3. Primary page (same for all after login)
3.1. Header with three options
3.1.1. Open sidebar with more options (see all classes, see enrolled classes, see settings, all fields can be clicked and entered)
3.1.2. Option to sign up to new class
3.1.3. Option to enter settings and log out

Work with databases via Java

1. Util class - ConnectToDatabase (method: getConnection[returns Connection])
2. Util class - CloseConnections (methods: closeConnection[void, param Connection], closePreparedStatement[void, param PreparedStatement], closeStatement[void, param Statement], closeResultSet[void, param ResultSet])
3. Util class - StringConstants (not methods, initialised final String constants used for SQL IO)

4. Abstract class - ReadFromDB (method: sendQuery[void, param String])
5. Class ReadStudentTable extends ReadFromDB (methods: TBD - various queries)
6. Class ReadMentorTable extends ReadFromDB (methods: TBD - various queries)
7. Class ReadTeacherTable extends ReadFromDB (methods: TBD - various queries)
8. Class ReadEnrolledTable extends ReadFromDB (methods: TBD - various queries)
9. Abstract class - WriteToDB (method: sendUpdateQuery[void, param String])
10. Class WriteStudentTable extends WriteToDB (methods: TBD - various queries)
11. Class WriteMentorTable extends WriteToDB (methods: TBD - various queries)
12. Class WriteTeacherTable extends WriteToDB (methods: TBD - various queries)
13. Class WriteEnrolledTable extends WriteToDB (methods: TBD - various queries)

14. Class User extended by other user roles
15. Class Admin used to create object Admin for DB work (values same as table Admin, default methods: [get, set, equals, various constructors TBD, toString])
16. Class Student used to create object Student for DB work (values same as table Student, default methods: [get, set, equals, various constructors TBD, toString])
17. Class Teacher used to create object Teacher for DB work (values same as table Teacher, default methods: [get, set, equals, various constructors TBD, toString])
18. Class Mentor used to create object Mentor for DB work (values same as table Mentor, default methods: [get, set, equals, various constructors TBD, toString])

Networking via Java

19. Class ServerForChat used to act as server for Student - Mentor chat
20. Class ClientForChat used to act as client(s) for Student - Mentor chat
21. Class SendEmailNotfication used to send mail notification for started/finished student homework's, also for sending notification to Mentors who didn't finish their reports on time

---

Databases

Users.db (tables: [admin, student, teacher, mentor, enrolled, mentorship_process, classes])
Files.db (tables: [homework, chat_history])

Web via JavaScript

Check all forms if ok before sending requests to Java -> DB

Short description of main classes:

User roles:

- Admin
- Teacher
- Mentor
- Student

*** User**

This is a general user class. It is extended by other user roles classes.
It has some general properties and methods.

- ID
- Name
- Surname
- Username
- e-mail address

- profile picture
- social network accounts (array)
- telephone number
...

- View (+ pic1 homepage) - Initial homepage
- Initial login (+ pic2 login) - login form contains fields email and password and it is same for everyone.
- Register form - everyone except admin have register form when logging in for the first time.
- option comment - leave comments on assignments, posts etc.
- chat - all user can communicate with someone.
- post - make posts
- account settings - edit profile
...

*** Mentor extends User**

- ID
- students (array) min 2
- getNotifications (when students take actions, or warning alert)
- writeWeeklyReport (method from Report class)
...

*** Student extends User**

- ID
- classes (array)
- grades (array)
- markHomework (from Assignment class)
- joinClass (with the given code)
- leaveClass ()
...

*** Teacher extends User**

- ID
- postAssignemt (from Assignment class)
- writeDailyReport (from Report class)
- addStudent (from Class class)
- visibleToMentorsOption (from Assignment class)
...

*** Report**

- ID
- description
- type
- date
- makeAReport (used by Admin class)
- calculateWeekOfClasses (use Class class)
...

*** Assignment**

- ID
- title
- date
- type
- sourceClass (class, exercises)
- markStarted (used by Student class, onclick mentor gets a notification)
- markFinished (used by Student class, onclick mentor gets a notificatio)

*** Class**

- ID
- name
- teacher
- students (array)
- isActive (method)

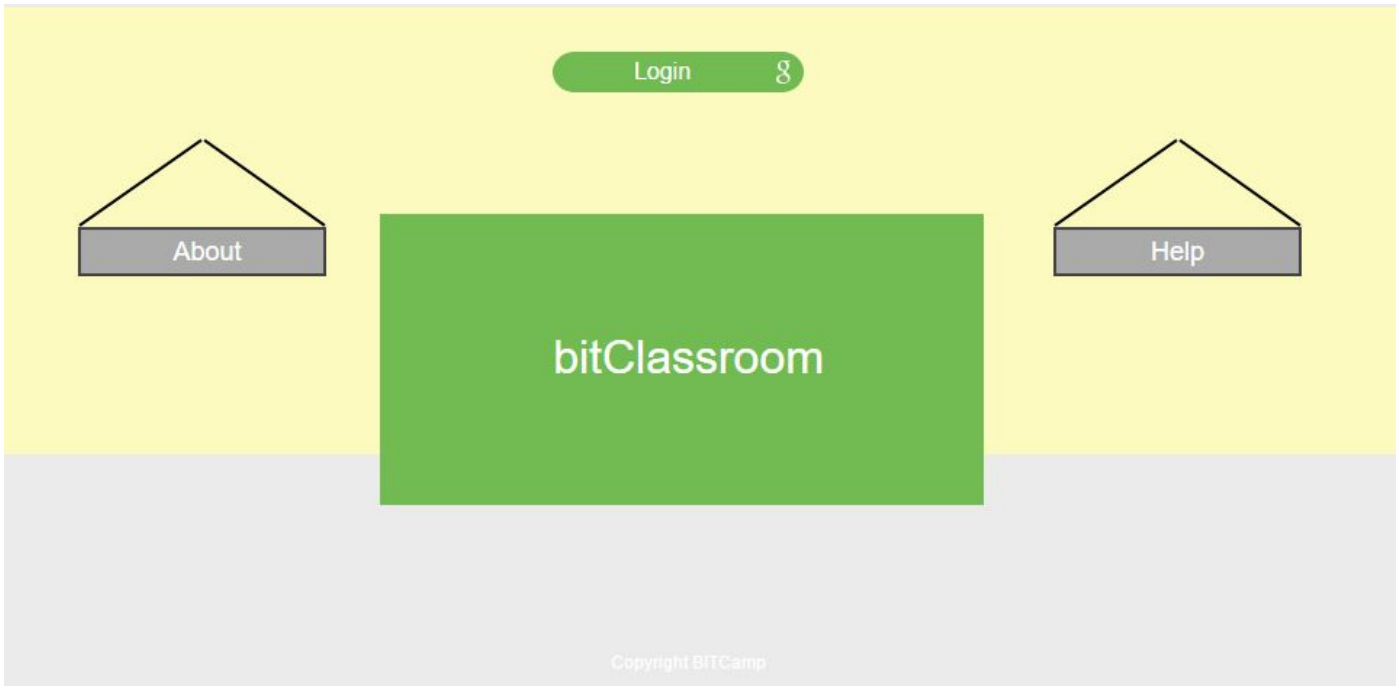
*** Materials**

- postMaterials
- shareMaterials
- deleteMaterials

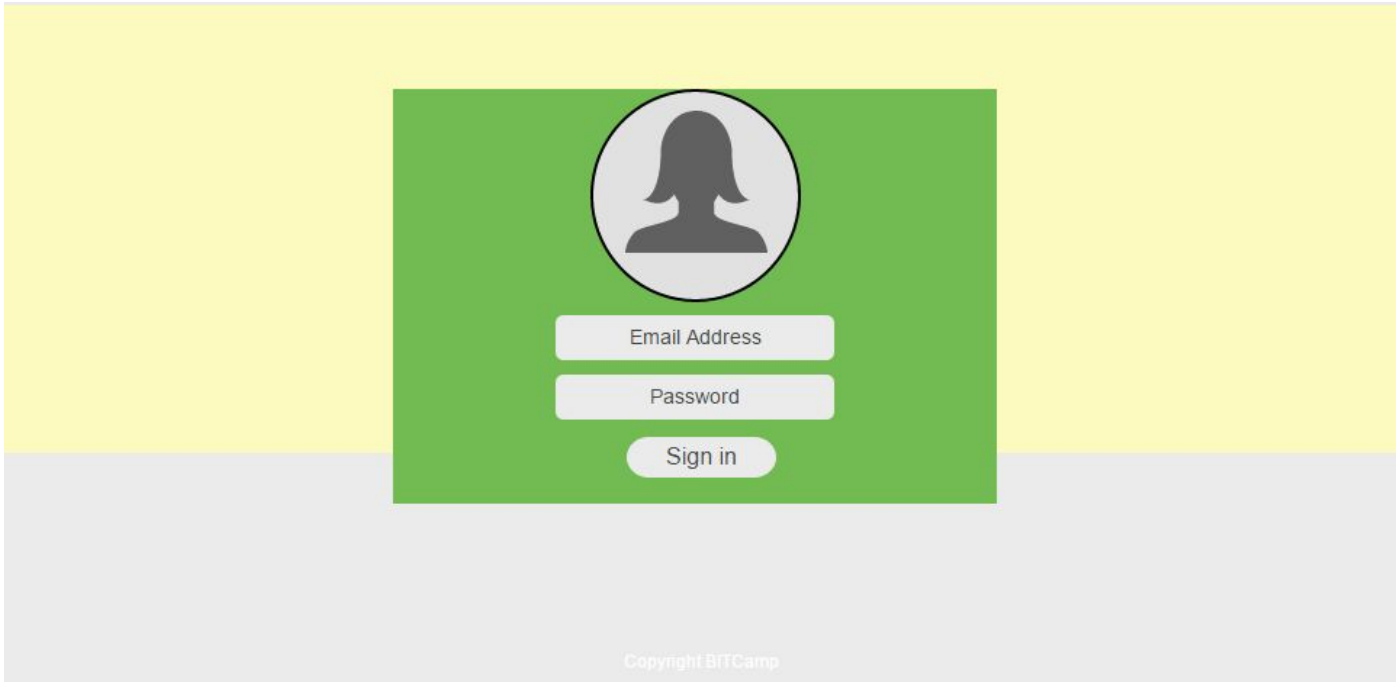
Generally we expect to have three times more classes, this is what we could imagined when dealing with this problem for the first time.

For our first sprint we planned to make LOGIN form (initial page), database for login, HTML file.

It should look something like this, but it is not a final version, it is just our first vision which will guide us in this week:



This should be a homepage. It will show up every time the app is opened. If the Login is clicked form below should show up:



Work with databases

All work will be done through Java classes.

First of all there will be class of constants for declaring string type variables that will hold name of driver and name of database location among other strings. All the constants declared in this class will be used for setting and getting various inputs from database tables.

Example of class with string constants:

```
public static final String DRIVER = "org.sqlite.JDBC";
public static final String SQL_DB_LOCATION = "jdbc:sqlite:databases/login.db";
public static final String CREATE_ADMIN_TABLE = "DROP TABLE IF EXISTS admin;"
public static final String ADD_TO_TABLE_ADMIN = "INSERT INTO admin (user_id, unique_id, name, email, password, first_entry) VALUES (?, ?, ?, ?, ?, ?)";
```

We will have number of utility classes used to open connection to database, close all connections etc. For example Class ConnectToDatabase will hold method that will return established connection to the base. In order not to write this code every time we need connection we will have classes and methods like this. Another example is CloseConnections class that will hold methods that close various connection to databases in exception handled manner:

- void closeConnection(Connection)
- void closePreparedStatement(PreparedStatement)
- void closeStatement(Statement)
- void closeResultSet(ResultSet)

Implementation of inheritance is used in work with databases. Abstract class with defined methods for various querys will be used by other more specific classes that will handle more specific querys. One model is used for reading from database, and one for writing to database. For example if you want to search for mentor you will have ReadMentorTable class that will extend ReadFromDB class. In ReadMentorTable few methods will be overrided in order to get dependable results, but most methods defined in ReadFromDB class will be used throughout all classes that read from database.

We will implement undefined number of tables at this moment, but we can tell for sure that there will be one database used for all users to login, acces various parts of application, see information that are required, store all data about users, store all data about files being processed etc.
So there will be four tables for each user (admin, student, teacher, mentor), also there will be multiple tables that will join these users on application, like class, enrolled, mentorship_process, tasks, files_submitted etc.

Tables that will join users will primarily have id's of users. Users will have more complex tables, this is example of user table with just few initial required fields upon admin creation of user.
create table EXAMPLE (

```
    user_id integer primary key auto_increment,
    unique_id varcher(25) not null unique, // php function used to generate special id
    name varchar(50) not null,
    email varchar(100) not null unique,
    password varchar(100) not null,    // php base64_encode function for encryption
    salt varchar(20) not null,
    first_entry timestamp,
    last_entry timestamp
);
```

This example table will have fields that will be inputted once user have created full profile after first login on application.

Process of initial login (work with databases)

User is welcomed by GUI part that have one area for inputting email address, one for inputting password and one button for submitting login form.
If everything checks out (and JavaScript part will check this on web itself) JSON message is sent to server. Server now parses this message and first checks if username exists in database. If this check passes, process of checking password is continued, if not user receives false response that is parsed in warning message.

Example in human-readable code:

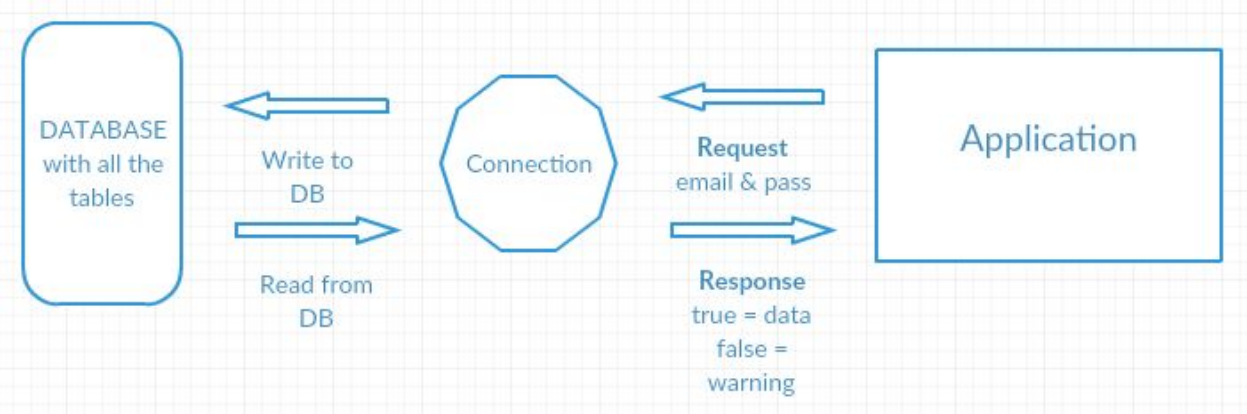
```
class User used for every user in system
// required fields for admin to enter
Long id;
String username;
String email;
String password;

// required fields for user to enter upon initial login
String firstName;
String lastName;
String phoneNumber;

// optional fields for user to enter upon initial login
```

Class Login extends ActionFor html Login Button

email = getEmail = from html field
password = getPassword = from html field

User temp = new User();
temp.setEmail(email);
temp.setPassword(password);

Class SendQuery (method: boolean sendFieldToCheckIfExists())
if true:
Class SendQuery (method: boolean sendPasswordStringToCheckIfMatchesUser())
if true:
Class SendQuery (method: void sendUserToApplication())



Technologies

- Java
- JavaScript
- HTML
- SQLite
- Play Framework
- css, jQuery, ajax
  …