Chapter 1 - use cases

Use case 1: Creating a new turing machine

* **Actor**: Member
* **Precondition**: Member in the system
* **Postcondition**: System creates the turing machine
* **Parameter**: python code
* **Actions**:

1. navigate to the turing machine creation section
2. Use the visual code editor to create the turing machine states, transitions and initial/final states
3. Create the turing machine with a unique identifier

|  |  |  |  |
| --- | --- | --- | --- |
| **Participants** | **Parameters** | **Expected Result** | **Scenario** |
| Member | username/id, code | Successfully created | Good |
| Member | empty string ,invalid code that does not compile | the system throws error message describing the error | Sad |
| Member | username/id, bad code | code with compilation error, message will be thrown | Bad |

Use case 2: Editing an Existing Turing machine

* **Actor**: Member
* **Precondition**: Created turing machine by the member
* **Postcondition**: Turing machine updated
* **Parameter**: code
* **Actions**:

1. Access the list of saved Turing machines
2. selected desired turing machine
3. user the visual code editor to make modifications
4. Save the changes

|  |  |  |  |
| --- | --- | --- | --- |
| **Participants** | **Parameters** | **Expected Result** | **Scenario** |
| Member | valid code | update turing machine code successfully and saved data | Good |
| Member | malicious code | the system detects the attack , and prevents it using several techniques like honey pot.. | Bad |
| Member | invalid code that does not compile | error message that describes the error and the lines where the compile error occurred and turing machine is not saved | sad |

case case 3: Saving a Turing machine

* **Actor**: member
* **Precondition**: Turing machine created
* **Postcondition**: system save the turing machine
* **Parameter**: user\_id, code , id / name of the turing machine
* **Actions**:

1. Complete the creation or modification of a turing machine
2. choose to save the turing machine
3. provide a name or identifier for the saved turing machine
4. confirm save action

|  |  |  |  |
| --- | --- | --- | --- |
| **Participants** | **Parameters** | **Expected Result** | **Scenario** |
| Member | code of turing machine and distinct id | turing machine code is saved | good |
| Member | code of turing machine and already used id or name | system throws a message describing the problem and how to solve | sad |
| Member | malicious code | system detect malicious code inputs and  member is blocked from further usage of the system  relevant message will be thrown | Bad |

Use case 4: Running a Test on a turing machine

* **Actor**: member
* **Precondition**: Turing machine created
* **Postcondition**: system simulate the test’s input on the turing machine
* **Parameter**: test input(string)
* **Actions**:

1. Open the test execution interface
2. input a test case for the selected turing machine
3. initiate the simulation
4. view the step-by-step execution and final result

|  |  |  |  |
| --- | --- | --- | --- |
| **Participants** | **Parameters** | **Expected Result** | **Scenario** |
| Member | test input | the system runs the test and displays results | good |
| Member | test input , but the test makes infinite loop | the system runs the test, but after defined time it will throw error timeout and suggests common mistakes that may has occurred | sad |
| Member | test input not in the turing machine (abc / letters) | error message will be thrown | bad |

Use case 5: Running multiple Tests

* **Actor**: member
* **Precondition**: Turing machine created
* **Postcondition**: system simulate the tests on the turing machine
* **Parameter**: tests list
* **Actions**:

1. Select a turing machine for testing
2. input a set of test cases
3. initiate the tests
4. review the results for each test case

|  |  |  |  |
| --- | --- | --- | --- |
| **Participants** | **Parameters** | **Expected Result** | **Scenario** |
| Member | tests as lists | the system runs the tests and displays results | good |
| Member | tests as list , but at least one test makes infinite loop | the system runs the tests, but after defined time it will throw error timeout and suggests common mistakes that may has occurred | sad |
| Member | invalid tests input(contains letters not recognized by the turing machine) | error message will be thrown | bad |

Use case 8: Retrieving a saved turing machine

* **Actor**: member
* **Precondition**: Turing machine created and saved
* **Postcondition**: system retrieves turing machine
* **Parameter**: turing machine id/name
* **Actions**:

1. member enters turing machine saved list tab
2. select previously saved turing machine
3. system provide the user the relevant turing machine

|  |  |  |  |
| --- | --- | --- | --- |
| **Participants** | **Parameters** | **Expected Result** | **Scenario** |
| Member | valid saved turing machine id/name | the system provide the user the turing machine code | good |
| Member | invalid saved turing machine id/name | error message will be thrown | sad |
| Member | valid saved turing machine id/name | database connection failed, error message will be thrown | bad |

Use case 9: Viewing animated model

* **Actor**: member
* **Precondition**: valid turing machine
* **Postcondition**: animation and visualization of the turing machine
* **Parameter**: turing machine id ,word , tape , operator , r/ w head
* **Actions**:

1. member access desired turing machine
2. member provides which word to run on the machine
3. the system will display the animation of the machine running the word

|  |  |  |  |
| --- | --- | --- | --- |
| **Participants** | **Parameters** | **Expected Result** | **Scenario** |
| Member | word and id of turing machine | the system displays the animation of the machine while running the word | good |
| Member | invalid turing machine id | error message will be thrown | bad |
| Member | invalid word input(contains letters not recognized by the turing machine) | error message will be thrown | sad |

Use case 10: Deleting a turing machine

* **Actor**: member
* **Precondition**: Turing machine created and saved
* **Postcondition**: system delete the saved turing machine
* **Parameter**: user\_id, id / name of the turing machine
* **Actions**:

1. user creates and saves new turing machine or accessing existing turing machine
2. user selects to delete the turing machine
3. system deletes the turing machine

|  |  |  |  |
| --- | --- | --- | --- |
| **Participants** | **Parameters** | **Expected Result** | **Scenario** |
| Member | valid turing machine id/name | turing machine code is saved | good |
| Member | invalid turing machine id/name | relevant error message will be thrown | Bad |

Use case 11: User Logout

* **Actor**: User
* **Precondition**: User registered to the system before and is currently logged in
* **Postcondition**: user logged out of the system
* **Parameter**: event logout button clicked
* **Actions**:

1. user is currently logged in
2. user selects to log out
3. system successfully logs out the user

|  |  |  |  |
| --- | --- | --- | --- |
| **Participants** | **Parameters** | **Expected Result** | **Scenario** |
| User | logout click event  and user is currently logged in | Successfully logged in to the system | Good |
| User | logout click event and user currently not logged in | relevant error message will be thrown | sad |
| User | logout click event  and user is currently logged in | system failed to update database, relevant message will be thrown | Bad |

Use case 12: User edits Account settings

* **Actor**: user
* **Precondition**: valid user
* **Postcondition**: edited account settings
* **Parameter**: user email address, password, name
* **Actions**:

1. user navigate to the account sittings
2. user updated the desired information ( email, password etc.)
3. system saves the changes

|  |  |  |  |
| --- | --- | --- | --- |
| **Participants** | **Parameters** | **Expected Result** | **Scenario** |
| Member | valid email address / password / name etc.. | the system saves the new information and updates the database | good |
| Member | invalid email address ,empty strings , illegal passwords that are easy to guess | error message will be thrown | sad |
| Member | malicious code | the system detects the attack , and prevents it using several techniques like honey pot.. | Bad |

Use case 13: Language Selection

* **Actor**: User
* **Precondition**: User enters the home page (index.html)
* **Postcondition**: website language changed
* **Parameter**: desired language
* **Actions**:

1. user select language tab
2. user selects new language (english / hebrew)
3. system successfully change the website language

|  |  |  |  |
| --- | --- | --- | --- |
| **Participants** | **Parameters** | **Expected Result** | **Scenario** |
| User | english / hebrew | Successfully changed language | Good |
| User | not valid language (not in the list we implemented) | system failed to change language, language do not change, relevant error message will be thrown | Bad |

Use case 14: Reviewing Test Results History

* **Actor**: member
* **Precondition**: Turing machine tests simulated before
* **Postcondition**: user review previous tests results
* **Parameter**: turing machine id/name
* **Actions**:

1. select a turing machine
2. access the result history section
3. review the detailed results of past test cases

|  |  |  |  |
| --- | --- | --- | --- |
| **Participants** | **Parameters** | **Expected Result** | **Scenario** |
| Member | valid turing machine id/name | the system shows all previous test cases results | good |
| Member | valid turing machine id/name  but no tests has been simulated before or implemented | the system shows all previous test cases results | sad |
| Member | invalid turing machine id/name | error message will be thrown | bad |

use case 15: System Administrator Monitoring

* **Actor**: User
* **Precondition**: user is administrator
* **Postcondition**: admin monitors system performance, and user activity and overall health of the system
* **Parameter**: desired dates activity
* **Actions**:

1. user logs in and identifies as administrator
2. admin access special tab, that is only visible to the admin
3. selecting the desired date to view

|  |  |  |  |
| --- | --- | --- | --- |
| **Participants** | **Parameters** | **Expected Result** | **Scenario** |
| admin | valid dates activity | the system displays the report | good |
| admin | invalid dates activity | error message will be thrown | sad |
| admin | malicious code | the system detects the attack , and blocks it. | Bad |

Chapter 2 - System Architecture

our system will include a server and client, the server will be implemented using react and the client will be implemented using javascript, the server mainly has 2 purposes:

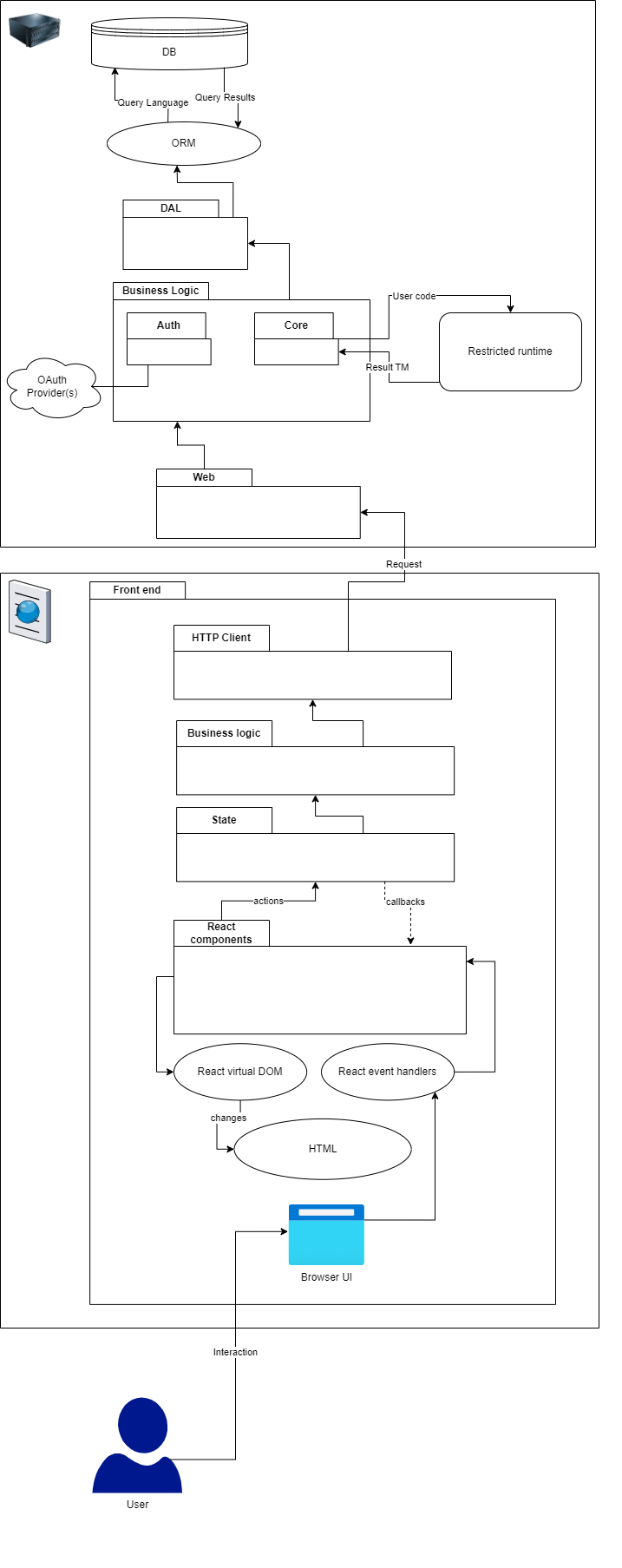
1. to display the functionality and of the system interactively
2. receive data from client , process it and display results, the data will be as strings or python code, in case of python code we will use a package called pyodide, which knows how to convert python code to javascript , and gives it to the business layer to continue the process.

the client will include http client and maybe business logic and state,

the server (which represents the frontend - presentation layer) will include

the react component which interacts with the browser UI, and flows data to the business layer below which includes the core which runs on restricted runtime , and auth which is responsible for registering and signing in users , using created username and password or google , twitter , yahoo… , and below the business layer will be the persistence layer , which will be implemented as an ORM.

we added a picture to illustrate the architecture.



Chapter 3 - Data Model

3.1 Description of Data Objects

our system will include many entities:

1. **turing\_machine**, which is our main entity

attributes:

ID, name , author\_id , creation\_date , code.

the main attributes are ID which identifies this object from other turing machine objects , and the code in javascript version , which was converted to javascript from python using pyodide , and it was provided by the user , the code represents the turing machine, using this code we will test the correctness of the turing machine.

1. **challenge**, it is an entity the represents the challenges that are created by the lecturers

attributes:

ID, name , creator\_id , creation\_date , challenge\_description, expiry\_date , list\_of\_legall\_words (for testing), list\_of\_illegal\_words (for testing).

the main attributes are ID which identifies this object from other challenges objects , challenge\_description will be a string and it describes the turing machine that is challenging to create, list\_of\_legall\_words which are words in the language of the intended turning machine , and list\_of\_illegal\_words which are words not in the language of the intended turning machine, these words will be used as tests to check the correctness of the turing machines which are provided by the users.

1. **user**, it represents the user in the system

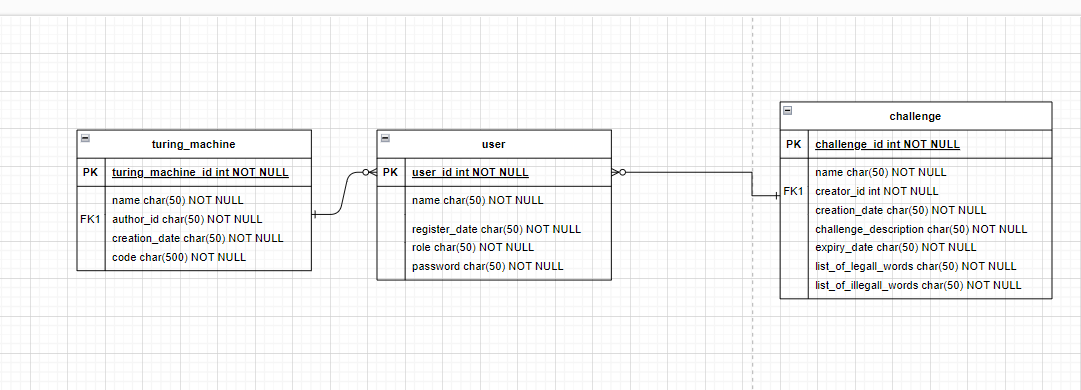
attributes:

ID, name , register\_date, role (student / lecturer / admin), password (hashed for security reasons) .

the main attribute is ID which identifies this object from other user objects, the role attribute might be implemented as state design pattern.

3.2 Data Objects relationships

3.3 Description of Data Objects



4 Behavioral Analysis

4.2 events

- the even: user registration.

When a user registers the react will pass the account details to the business layer , the controller in the business layer will receive the details, hash the necessary details and invokes the logger to log the event and eventually will pass the details to the repo controller in the persistence layer to save the new user in the database. Along the way if there was an error or exception the object that had the error will call the logger to log the detailed error and will throw the error , each layer will catch the error and will act accordingly , eventually the error will reach the react in presentation layer and will be displayed to the user.

- the even: user logs in.

When a user logs in the react will pass the account details to the business layer , the controller in the business layer will receive the details, hash the necessary details and invokes the logger to log the event and eventually will pass the details to the repo controller in the persistence layer to validate the details with the real ones in the database , in case the details are correct and match the ones in the database , the repo will return the user model to business layer to the controller , the controller will create the user and sends it back to presentation layer to display it to the user. Along the way if there was an error or exception the object that had the error will call the logger to log the detailed error and will throw the error , each layer will catch the error and will act accordingly , eventually the error will reach the react in presentation layer and will be displayed to the user.

- the even: creating turing machine.

After the user selects the desired turing machine to work on, a user writes python code to create turing machine , the react will convert the python code to javascript using pyodide and will pass the converted code to the business layer , the controller in the business layer will receive the code , invoke the logger to log the event and eventually will pass the details to the repo controller in the persistence layer to save the progress, the repo controller will get the selected turing machine and will pass it to the controller in the business layer (including the tests) , the controller will run the tests on the provided code of the truing machine, eventually will send the results back to the react and will be displayed to the user.

Along the way if there was an error or exception the object that had the error will call the logger to log the detailed error and will throw the error , each layer will catch the error and will act accordingly , eventually the error will reach the react in presentation layer and will be displayed to the user.

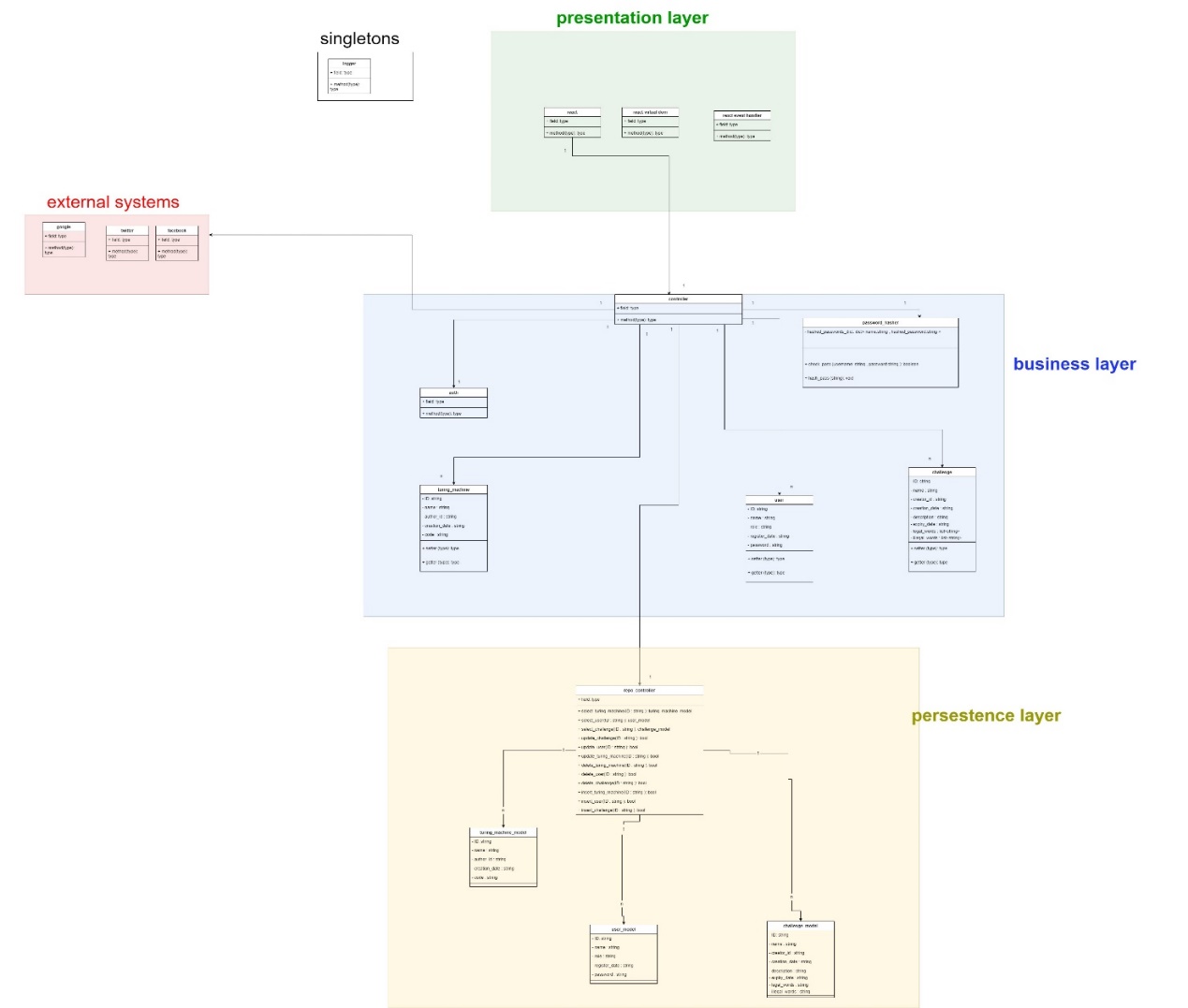
- the event: creating challenge.

the lecturer goes to the challenges section, and click on add button to add new challenge, then he provides details about the new challenge, name, description, legal and illegal words, the react will pass the details to the business layer , the controller in the business layer will receive the details , and will create the challenge . also will invoke the logger to log the event and eventually will pass the details to the repo controller in the persistence layer to save the challenge, the repo controller will insert the new challenge to the database. eventually if the process succeeded the react will display the result to the user.

Along the way if there was an error or exception the object that had the error will call the logger to log the detailed error and will throw the error , each layer will catch the error and will act accordingly , eventually the error will reach the react in presentation layer and will be displayed to the user.

5 Class Diagrams

5.1



5.2 class description

The presentation layer:

React\_virtual\_DOM

React\_event\_handler

React, react is the main class , it interacts with the browser to receive data

The react has controller as an attribute.

The business layer:

Controller, controller is the main class in this layer, it is the link between presentation layer and business layer and the link between data layer and business layer. It can access auth, in case of signing in using external systems , and it has access to all the objects like user turing machine and challenges.

It has repo\_controller as an attribute , so it can access persistence layer

Password\_hasher , It has password\_hasher dictionary as an attribute, it's operation is check\_pass which receives username and password, and checks if the password is correct using hash function and it's dictionary.

Another operation is hash\_pass which receives password and hashs it with salt , and adds it to the dictionary.

Turing\_machine, the attributes are ID, name , author\_id , creation\_date , code, operations are test\_word , which receives a string and checks if the turing machines accepts it or not, if it does not halt it throws an error.

User, the attributes are ID, name , register\_date, role (student / lecturer / admin), password, no significant operations, regular getters and setters.

Challenge, the attributes are ID, name , creator\_id , creation\_date , challenge\_description, expiry\_date , list\_of\_legall\_words (for testing), list\_of\_illegal\_words (for testing), no significant operations, regular getters and setters.

Persistence layer:

Repo\_controller, it is the main class of the persistence layer , it's operations are select , insert, update and delete all kind of models, like user\_model , turing\_machine\_model, all these operations are invoked by the controller in the business layer.

5.3 packages

As described previously , presentation's layer main class is react which has access to the controller, which is business's layer main class, react "communicates" with the business layer using controller , controller has access to repo\_controller, which is persistence's layer main class, controller "communicates" with the persistence layer using repo\_controller. Each layer knows the one that is bellow him, and each layer doesn't know the one that is above him.

Persistence layer, repo\_controller

business layer, controller

Presentation layer, react