Design Documentation

Created	@October 21, 2021 6:33 PM
Last Edited Time	@January 11, 2022 12:44 AM
Type	Technical Spec
Status	
Created By	
Last Edited By	
Stakeholders	

BLG 411E Software Engineering

Design Documentation

24.12.2021

GROUP - 8

ILC

Furkan Hayta

Ahmet Polat

Utku Sabri Kaya

Buğra Aydın

Table of Contents

1. Introduction

In this documentation, the software design of the project is done in 2 parts.

In the first chapter, System Architecture, the High-Level Architecture of the project was determined, shown, and explained in the diagram. In the High-Level Architecture section, which architectural structure will be used and the content of this structure is detailed. Afterward, the Component diagram was created, the components to be included in the project were determined and the relationship between them was defined.

In the second part, the Low-level Design of the project was released. In this part of the project, some more coding studies were carried out. While developing the project, it was tried to determine user classes and modules.

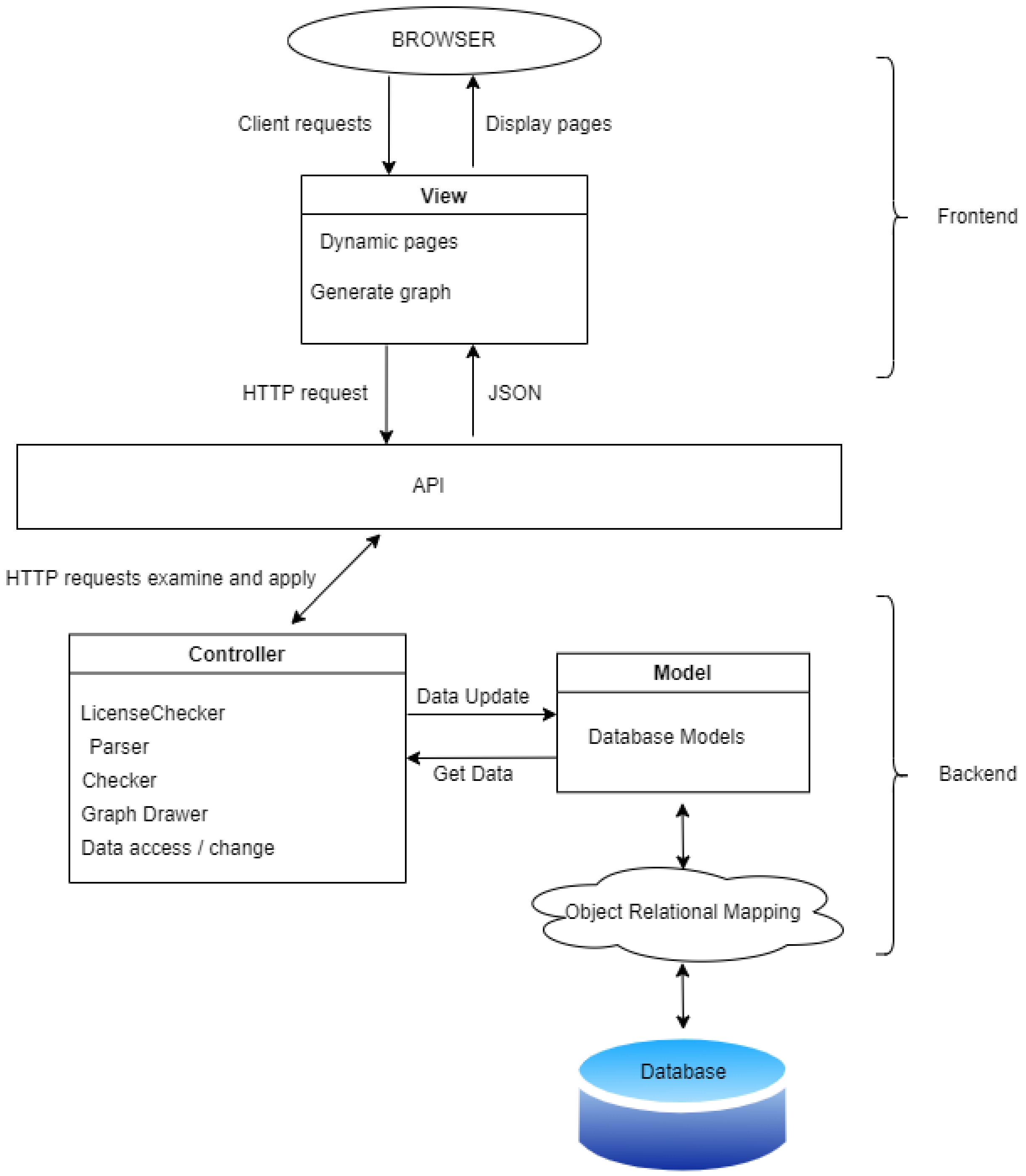
The classes that will be required in the project were drawn in the Class diagram, their attributes and methods were detailed. The relationship between these classes is given.

In the sequence diagram, how each use case in the project will work, which stages it will go through, what kind of interactions there will be are described.

In the last section, the Data Flow Diagram was drawn. The flow and features of each action are shown.

2. System Architecture

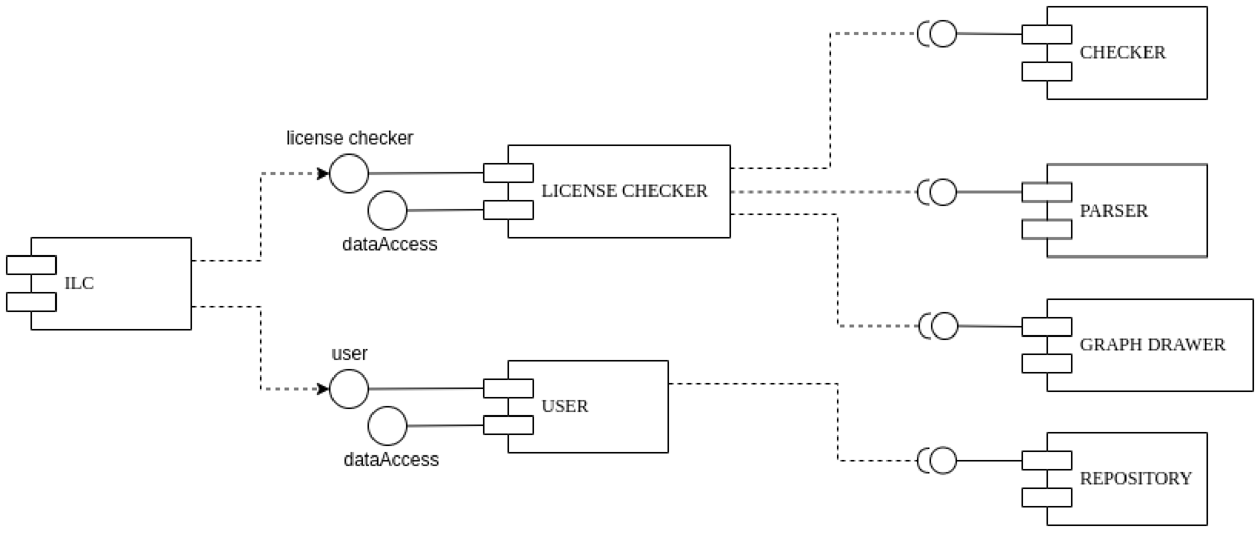
2.1 System High-Level Architecture Diagram and Explanation



- Since the project will be a web project, it has been created to separate the frontend and backend parts and to offer a specific workspace to those who will work in these areas.
- The communication of the Frontend and Backend parts will be carried out with the REST API.

- Modified the classical architecture models to be compatible with the project MVC structure and to work in harmony with the REST API.
- View structure in MVC architecture will be on the frontend, Controller and Model will be on the backend.
- View, which is the structure that the User displays and interacts with, will send user requests to the API in the form of HTTP requests and receive JSON type data from the API.
- HTTP requests coming from View are examined and the necessary Controller works.
- Some services that Controller has are Parser, Checker.
- For the necessary database operations in the Controller, a connection can be made with the Model and the Database can be updated or data can be retrieved.
- Model structure interacts with Database using Object Relational Mapping. Thus, Database becomes abstract and DB models are written and implemented as Class without the need for SQL queries.

2.2 Component (Package) Diagram

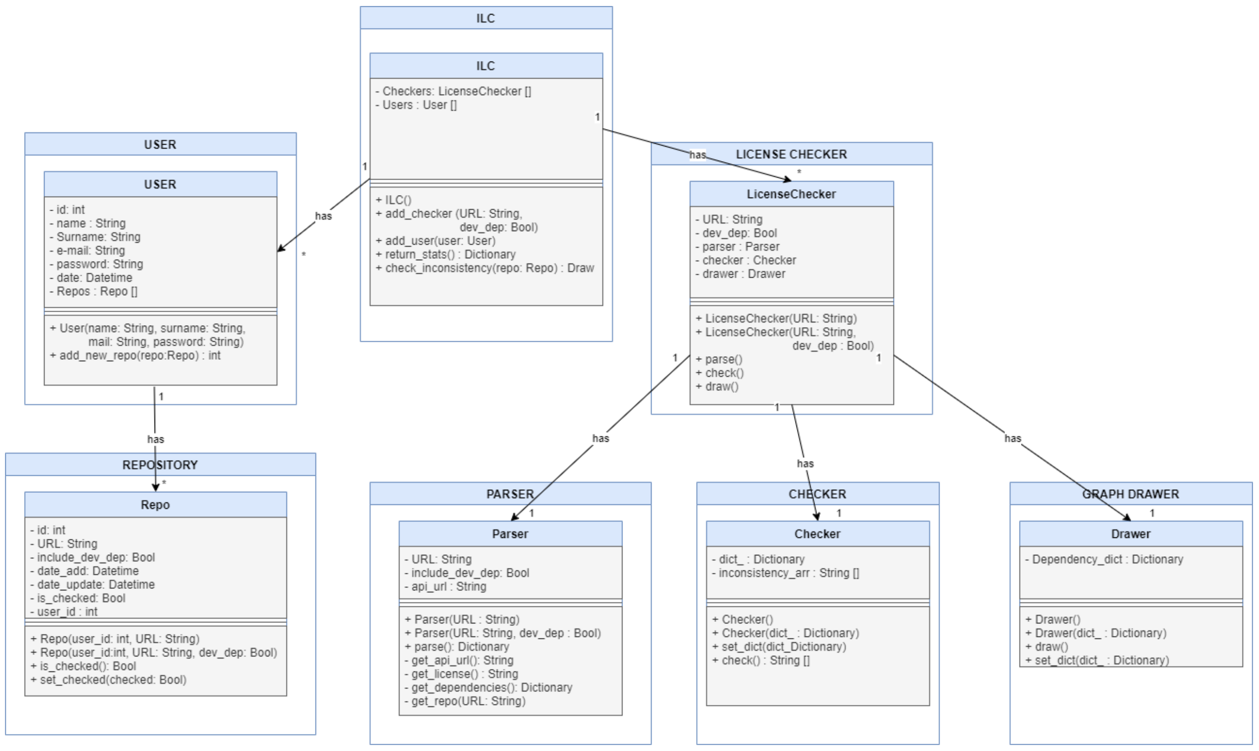


- ILC is main component.
- Dependicies of the ILC are License Checker and User.

- The repository is required for the User.
- Checker, Parser and Graph Drawer are required for License Checker.

3. Low-Level Design

3.1 Class Diagrams



- In the above graph, class diagrams are shown for each component.
- Although the component diagram has a higher abstraction level from the class diagram, in our design, each module has exactly one class.
- In the following bullet items, each class is explained in alphabetical order.
- In the Checker class, dependencies and their corresponding licenses and dependencies are kept in the dict_ dictionary as key and the value respectively. Since nested dependencies are possible, a value is an array that contains a string and a dictionary. Inconsistency array is also stored in the class objects. they are set and returned in check() function. It has two constructures, if the argument is not given, dict_ can be set with the set_dict() function.

- In the Drawer class, the dependency dictionary that has the same structure as in Checker is kept. The drawer also has similar constructors and set_dict() function. In the draw function, the dependency tree is drawn according to the dependency_dict attribute.
- In the ILC class, Arrays of the User and LicenceChecker classes are stored. Both of these classes' instances can be added to ILC with corresponding methods. ILC can return the length of these arrays as a dictionary object with the return_stats() method.
- In LicenseChecker class, an object of Parser, Checker and Drawer classes is stored. Also, the URL of the repository and including development dependencies choice are kept as attributes. LicenseChecker can get dependencies, check inconsistencies and draw dependency tree thanks to these objects. Moreover, it has two constructors. If development dependency choice is not given, the default value is not checking the development dependencies.
- In the Parser class, the API URL of the repository is kept additionally from LicenseChecker class. It can be set in get_api_url() method. parse() method is the only public method of the class besides constructors. parse() method gets all dependencies with get_dependencies() method, license of dependencies and the license of repo itself with get_license() method, and Github repositories of the dependencies with get_repo() function.
- In Repo class, id, added date, last updated date, URL, dependency choice, and the owner of the repo and check status is kept. This class and User class are ORM classes for database tables.
- In the User class, the id of the user, full name (separately), login parameters (e-mail and password -hashed of course-), sign-up date, and Repos of the user are stored. A new repository can be added to the Repos array with the add_new_repo() method.

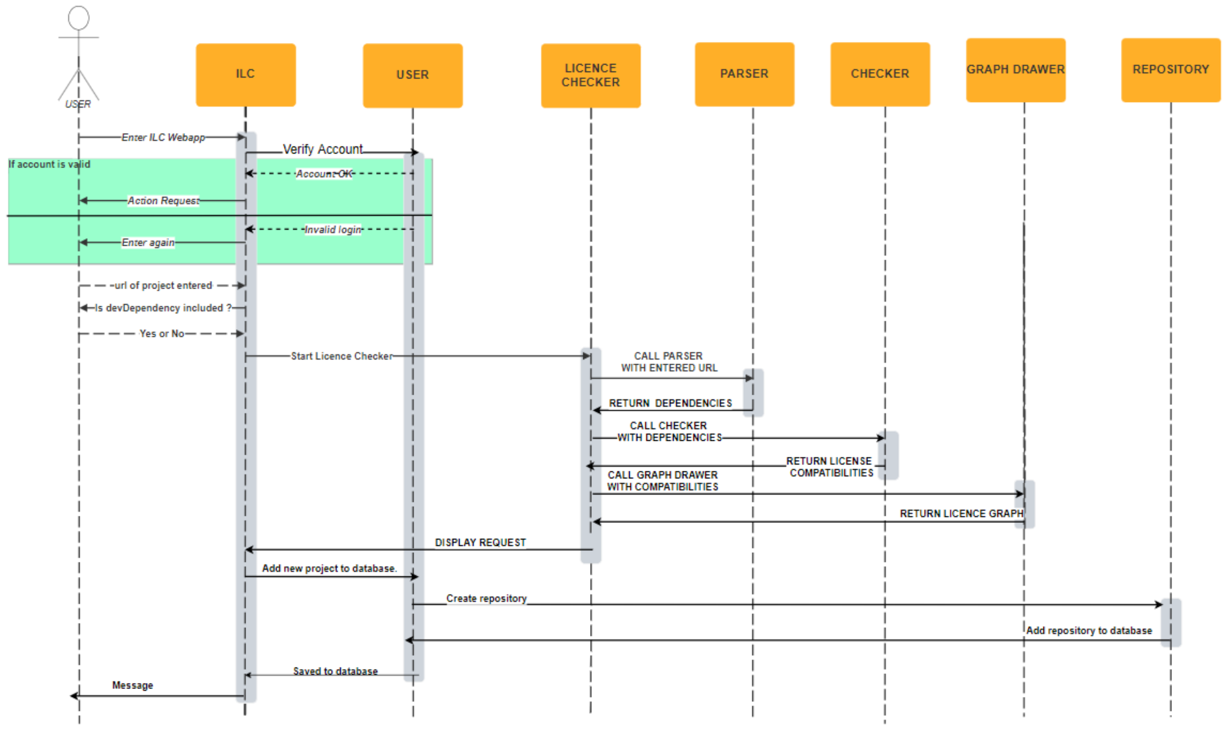
3.2 Sequence Diagrams

3.2.1) Use Case: User adds a new repository to check license consistency.

The user enters the ILC Webapp. It is checked whether the information entered with the database matches. The green box in the sequence diagram represents the if-else condition (Account verified or not). The user wants to check the license consistency in his/her project. ILC calls the Licence checker with entered URL.

License checker calls parser to get dependencies. Parser parses the dependencies of the project. And sends to license checker. License checker calls checker with obtained dependencies. And sends the consistencies between dependencies to license checker. License checker calls graph drawer to draw license graph of the project. The graph is sent to license checker then sent to ILC to display.

After user got the result he/she requested, the new repository is added to the database.



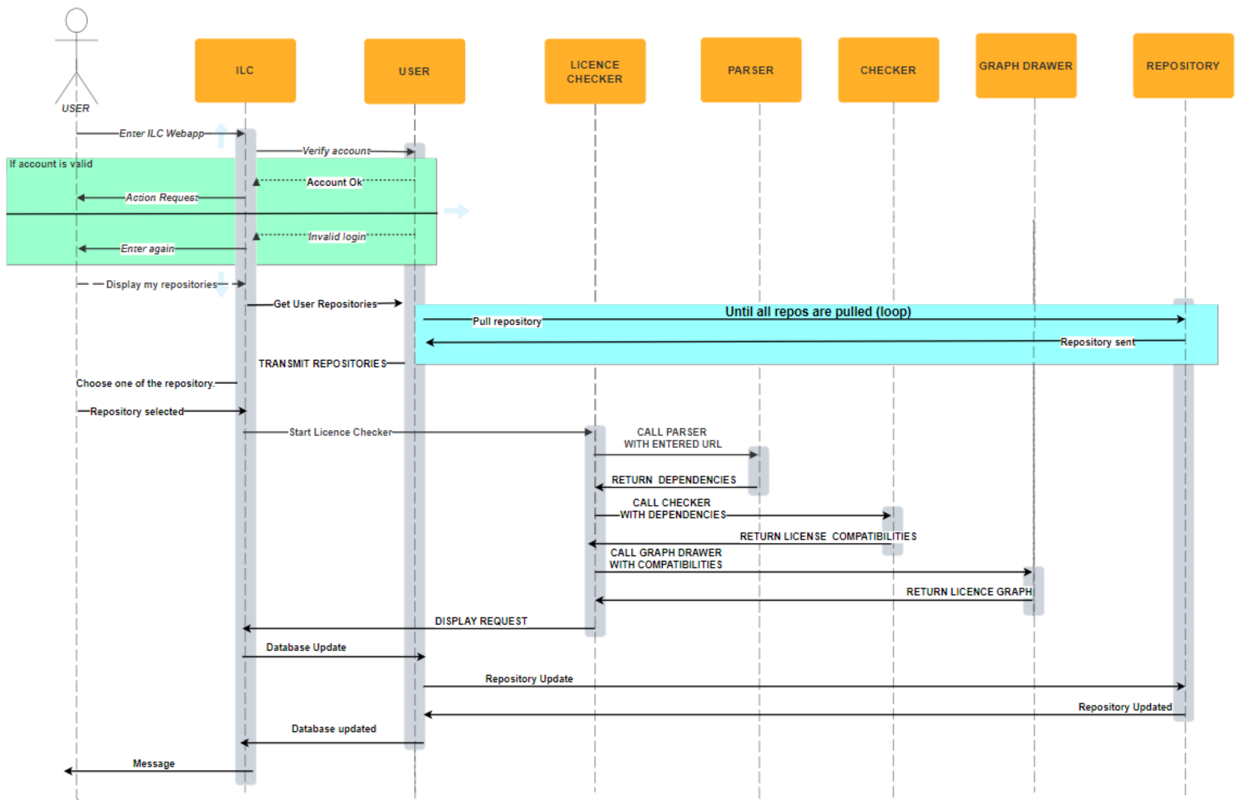
3.2.2) Use Case: User wants to check license consistency of existing repository that added database before.

The user enters the ILC Webapp. It is checked whether the information entered with the database matches. The green box in the sequence diagram represents the if-else condition (Account verified or not). The user wants to display his/her repositories.

ILC connects to the database to access user's repositories. Every repository belongs to user are pulled from the database. The blue box in the sequence diagram shows this procedure. Repositories are displayed. ILC waits for request from user. When user chooses one of the repositories, license checker is called.

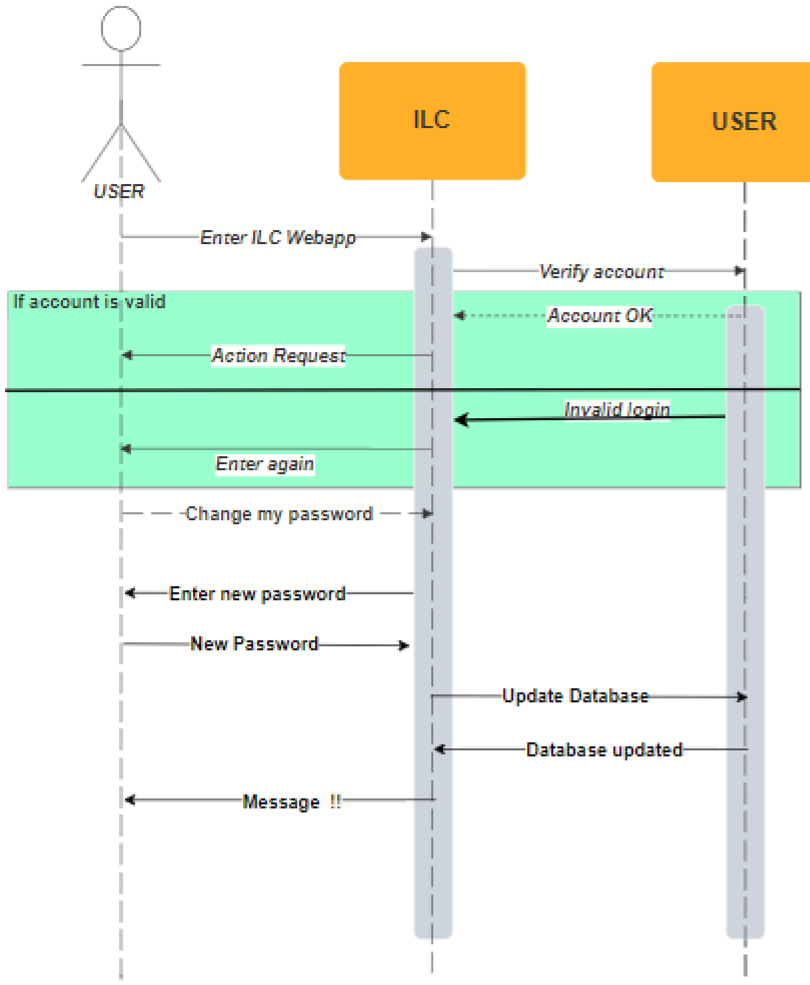
License checker calls parser to get dependencies. Parser parses the dependencies of the project. And sends to license checker. License checker calls checker with obtained dependencies. And sends the consistencies between dependencies to license checker. License checker calls graph drawer to draw license graph of the project. The graph is sent to license checker then sent to ILC to display.

After user got the result he/she requested, the database is updated.

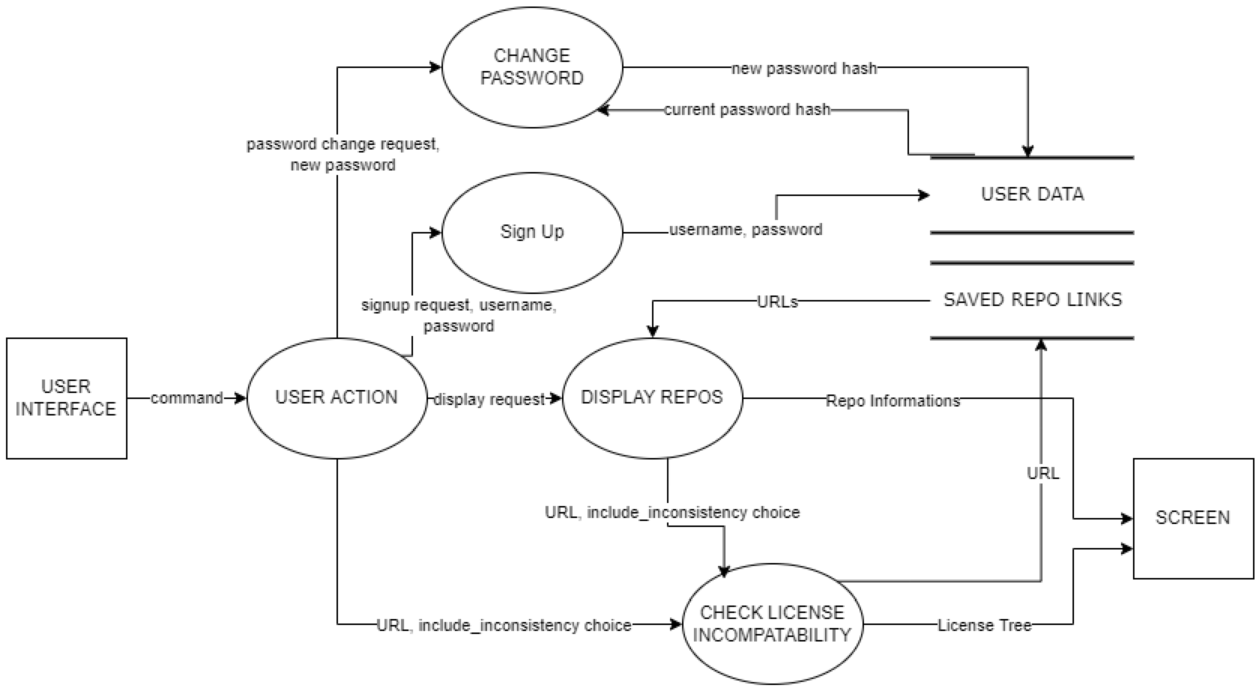


3.2.3) Use Case: User wants to change his/her password.

User logins to ILC Webapp. Clicks to change password button and ILC asks the user what do you want the new password to be. User enters the new password. ILC updates the database.



3.3 Data Flow Diagram



In the above diagram, the data flow of the ILC is shown. Users can do four different fundamental operations in ILC. Firstly, they can sign up for the web app. Users can go sign-up page and send the sign-up form in Section 4. The given information is saved in the Users database in the E/R diagram. Secondly, users can change their passwords at any time. The Hash of the password is updated accordingly in the users' table. Also, Users can check their GitHub repo's license incompatibility giving the URL of the repo and selecting whether development dependencies will be included or not. An incompatibility graph is calculated and displayed to the user. Finally, Users can display all of their repos. In the Repos table, necessary data (URL) is getting and displayed to the user.