# Purpose of this Document

The main purpose of this document is to highlight specific features that are within both the game and the server and describe how they were implemented. This includes a listing of relevant files to each feature. Therefore with this document we aim to help guide future developers with their own implementation and refactoring efforts.

# Game Features

The game was created using the Unity Game Engine (Unity 5). Almost everything in the game is controlled by scripts assigned to specific game objects. These scripts define the object’s behavior and actions. The game utilizes a SQLite database for storing game information locally. The database contains tables with information about the molecules in the game as well as a list of tasks available to the player and their progress on the tasks. The decision to have a local database was made for several reasons. One of the biggest was enabling us to have an offline mode. Another reason was we did not see the benefit of keeping the molecule information stored on the server. For accessing the database we utilized a third party library called SQLiteUnityKit. This allowed us to get up and running with SQLite without having to roll our own access library. This could be changed by later teams if they desire more control over how queries run. There were three main features added to the game by our team. The Moleculog, the task system, and programmatic generation of molecule models.

## Moleculog

This is similar to the Pokedex from Pokemon or the scanning and Logbook from the Metroid Prime series. Users can go up to a molecule and scan it for more information. The Moleculog consists of an overview screen where all of the molecules in the game are listed and a details screen. The details screen contains a 2D image of the molecule (not implemented), a 3D rotatable view, a description, and a list of the angles between different bonds in the molecule. The intention was to only show molecules the students had already scanned but we did not get to implementing that.

## Task System

The task system was implemented to give the game a little more direction. The idea is if students have goals to complete within the game they will be more likely to actually play it. The system as it stands is very basic. Right now we only have two tasks, Drive the Ship, and Collect Four Molecules. Only the collection task is actually able to be completed and turned in. To keep track of tasks there is a Task Log similar to the Moleculog. It contains a list of accepted tasks on the left which when clicked on provide details about the task on the right. The information included is the description, status, progress, and pay for the task. The flow of a task is: accept a task from a computer in the hub world, enter a level, collect molecules (or whatever the goal is), open task log and hit turn in when complete. Several other task types were discussed but not implemented such as collecting specific molecules and collecting molecules with specific geometries or bond angles. Another discussed but not implemented feature was getting paid for tasks. This money could then be used for upgrading your ship. See the Design Document for more information.

## Programmatic Molecule Model Generation

## Reference

This is a file which contains helper functions required for the game. All of the queries for the database are contained in this file. It also contains the definitions of how molecules and tasks are represented in the game (MoleculeDef and TaskDef structs respectively). Static functions for building lists of MoleculeDefs and TaskDefs based on certain criteria are also contained here.

# Server Features

The server side uses Node.js and Express.js to handle web traffic and enables access to the web client, which students and instructors can use to access the game and student stats from the game. In addition, the server side uses the Jade templating engine to generate HTML. Specifically, all of the server side javascript code and Jade files are found under the ‘views’ folder, while all of the CSS and client side javascript code are found under ‘public/views’ folder. In terms of interacting with the database, the server uses Mongoose to interact with MongoDB, so the models.js file initializes all of the database models, which are found within the ‘schema’ folder. Finally, the API that the game uses is located entirely within the ‘api’ folder, and it uses its own separate router, model, and controller files for each specific feature exposed by the API. The API interacts with the game only through JSON and the server side currently does not track any session based information about API calls.

## Signup Feature

**Files:** Everything in the ‘views/instructor\_signup’, ‘public/views/instructor\_signup’, views/student\_signup’, and ‘public/views/student\_signup’ folders

The signup feature is divided between students and instructors with the idea that in order to sign up as an instructor you needed another form of identification in the form of a token. However, due to other priorities set by the 2015-12016 team this is currently implemented as a server side check for another specific password, which is far from ideal. Therefore, this should be implemented so that a random token is generated, sent to the instructor, saved in the database, and the queried when the instructor tries to sign up. Also, this feature is great for looking into the workflow utility, which is a wrapper for Node.js EventEmitter.

## Logging in / Logging Out

**Files:** Everything in the ‘views/login’, ‘public/views/login’, and views/logout’

This feature handles the authentication of a person on the server. It also handles how the server should redirect based upon whether or not the person that logged in is a student or an instructor. It is important to note however that the email features of resetting a password through the login functionality does not appear to be working, and this was not made a priority for the 2015-2016 team.

## Students - Downloading the Game

**Files:** Everything in ‘views/game’ and ‘public/views/game’

Note: Game executables should be stored in ‘public/views/game/game\_executable’ as .zip files, but these files should not be committed to the repository.

This feature is quite simple, it allows for students to be able to download the game once they log in. However, it is important to note that the links reference files that are named ‘Molecular\_Mission\_<OS>.zip’ where <OS> can be Mac, Windows, or Linux and that these files must be in the game\_executable directory as noted above.

## Instructors - Viewing Game Stats

**Files:** Everything in ‘views/admin/students’ and ‘public/views/admin/students’

Note: Everything in the ‘admin’ directories correspond to pages that are only accessible to instructors.

One of the main priorities of instructors is that they need to be able to see stats generated from a student playing the game. Therefore, the main page in ‘views/admin/students’ brings up a list of all the students so that an instructor can see a student’s stats if they hit the ‘Details’ button. Specifically, once the Details button is pressed, it will update a modal embedded within the page and display the modal with the student’s stats. This can still be expanded upon so that only students for a particular instructor are shown, but this requires modification to the database schema as well as to the student sign up process.