

DESIGN MOCK-UP

CRC Cards	1
CRC Model	7
System Interaction and Environment	10
System Architecture	10
System Decomposition	11
Prototype Design Mock-up	12

Components (UI objects)

① App

Parent Class: None

Subclasses: None

Responsibilities

- Encases the other components
- Enables the router
- Defines generic styling

Collaborators

- Router

② Dashboard

Parent Class: None

Subclasses: None

Responsibilities

- Display the user's files
- Provide interface for user to interact with their content
- Allow users to share their files
- Facilitate code importing/exporting for users
- Allow users to create and delete files

Collaborators

- Header
- Router
- File Storage Service
- Authorize Service

③ Settings

Parent Class: None

Subclasses: None

Responsibilities

- Allow user to change their information
- Provide UI option(s)

Collaborators

- File Storage Service
- Authorize Service

④ Header

Parent Class: None

Subclasses: None

Responsibilities

- Provide UI component for the top bar
- Allow bar to be customized
- Allow other components to compose and control behavior of header

Collaborators

- None

⑤ Landing

Parent Class: None

Subclasses: None

Responsibilities

- Provide UI component for the landing page
- Display registration and login options

Collaborators

- Register
- Login
- Router

⑥ Compiler

Parent Class: None

Subclasses: None

Responsibilities

- Display compiled code
- Communicate information to shell
- Stop compilation
- Provide options to interact with files
- Facilitate code saving button event

Collaborators

- Compile Service

7 Guest Coder

Parent Class: None

Subclasses: None

Responsibilities

- Provide UI component for the the interface seen by guests
- Use compiler and header for GUI
- perform code running as normal

Collaborators

- Header
- Router
- Compiler

8 User Coder

Parent Class: None

Subclasses: None

Responsibilities

- Provide UI component for the interface that logged in users see
- Allow Code saving option
- Allow code exporting / downloading
- Authenticate user

Collaborators

- Header
- Compiler
- Router
- File Storage Service
- Authorize Service

9 Shared View

Parent Class: None

Subclasses: None

Responsibilities

- Header
- Compiler
- File Storage Service
- Router

Collaborators

- Header
- Compiler
- File Storage Service
- Authorize Service

10 Registration

Parent Class: None

Subclasses: None

Responsibilities

- Provide fields for creating accounts
- Secure user credentials
- Route users to their dashboards

Collaborators

- Router
- Authorize Service

11 Login

Parent Class: None

Subclasses: None

Responsibilities

- Provide fields for users to log in to the site
- Route users to their dashboards

Collaborators

- Router
- Authorize Service

12 Not-Found

Parent Class: None

Subclasses: None

Responsibilities

- Notify users of errors
- Route users to the home page

Collaborators

- Router

Services (Injectable service objects / DAOs)

① Authorize Service

Parent Class: None

Subclasses: None

Responsibilities

- Authenticates user logins
- Protects and prepares user requests (like log ins and registers)
- Creates and edits tokens for persistent authentication

Collaborators

- Router

② File Storage Service

Parent Class: None

Subclasses: None

Responsibilities

- Facilitates file sharing and access control
- Facilitates adding, saving/storing, deleting files from/to DB

Collaborators

- Router
- Authorize Service

③ Compile Service

Parent Class: None

Subclasses: None

Responsibilities

- Correctly send requests to compilex (module)
- Decode information sent from requester send to module and retrieve output

Collaborators

- None

Misc

① Router

Parent Class: None

Subclasses: None

Responsibilities

- Transition through various tasks and tabs
- Controls the view displayed to the user
- Controls user's access

Collaborators

- None

App

- Encases the other components
 - Enables the router
 - Defines generic styling
- Router

Dashboard

- Display the user's files
 - Provide interface for user to interact with their content
 - Allow users to share their files
 - Facilitate code importing/exporting for users
 - Allow users to create and delete files
- Header
 - Router
 - File Storage Service
 - Authorize Service

Settings

- Allow user to change their information
 - Provide UI option(s)
- File Storage Service
 - Authorize Service

Header

- Provide UI component for the top bar
 - Allow bar to be customized
 - Allow other components to compose and control behavior of header
- None

Landing

- Provide UI component for the landing page
 - Display registration and login options
- Register
 - Login
 - Router

Compiler

- Display compiled code
 - Communicate information to shell
 - Stop compilation
 - Provide options to interact with files
 - Facilitate code saving button event
- Compile Service

Guest Coder

- Provide UI component for the the interface seen by guests
 - Use compiler and header for GUI
 - perform code running as normal
- Header
 - Router
 - Compiler

User Coder

- Provide UI component for the interface that logged in users see
 - Allow Code saving option
 - Allow code exporting / downloading
 - Authenticate user
- Header
 - Compiler
 - Router
 - File Storage Service
 - Authorize Service

Shared View

- Header
 - Compiler
 - File Storage Service
 - Router
- Header
 - Compiler
 - File Storage Service
 - Authorize Service

Registration

- Provide fields for creating accounts
 - Secure user credentials
 - Route users to their dashboards
- Router
 - Authorize Service

Login

- Provide fields for users to log in to the site
 - Route users to their dashboards
- Router
 - Authorize Service

Not-Found

- Notify users of errors
 - Route users to the home page
- Router

Authorize Service

- Authenticates user logins
 - Protects and prepares user requests (like log ins and registers)
 - Creates and edits tokens for persistent authentication
- Router

File Storage Service

- Facilitates file sharing and access control
 - Facilitates adding,saving/storing, deleting files from/to DB
- Router
 - Authorize Service

Compile Service

- | | |
|--|--|
| <ul style="list-style-type: none">• Correctly send requests to compilex (module)• Decode information sent from requester send to module and retrieve output | <ul style="list-style-type: none">• None |
|--|--|

Router

- | | |
|--|--|
| <ul style="list-style-type: none">• Transition through various tasks and tabs• Controls the view displayed to the user• Controls user's access | <ul style="list-style-type: none">• None |
|--|--|

System Interaction and Environment

The technologies GRACE will be using will include:

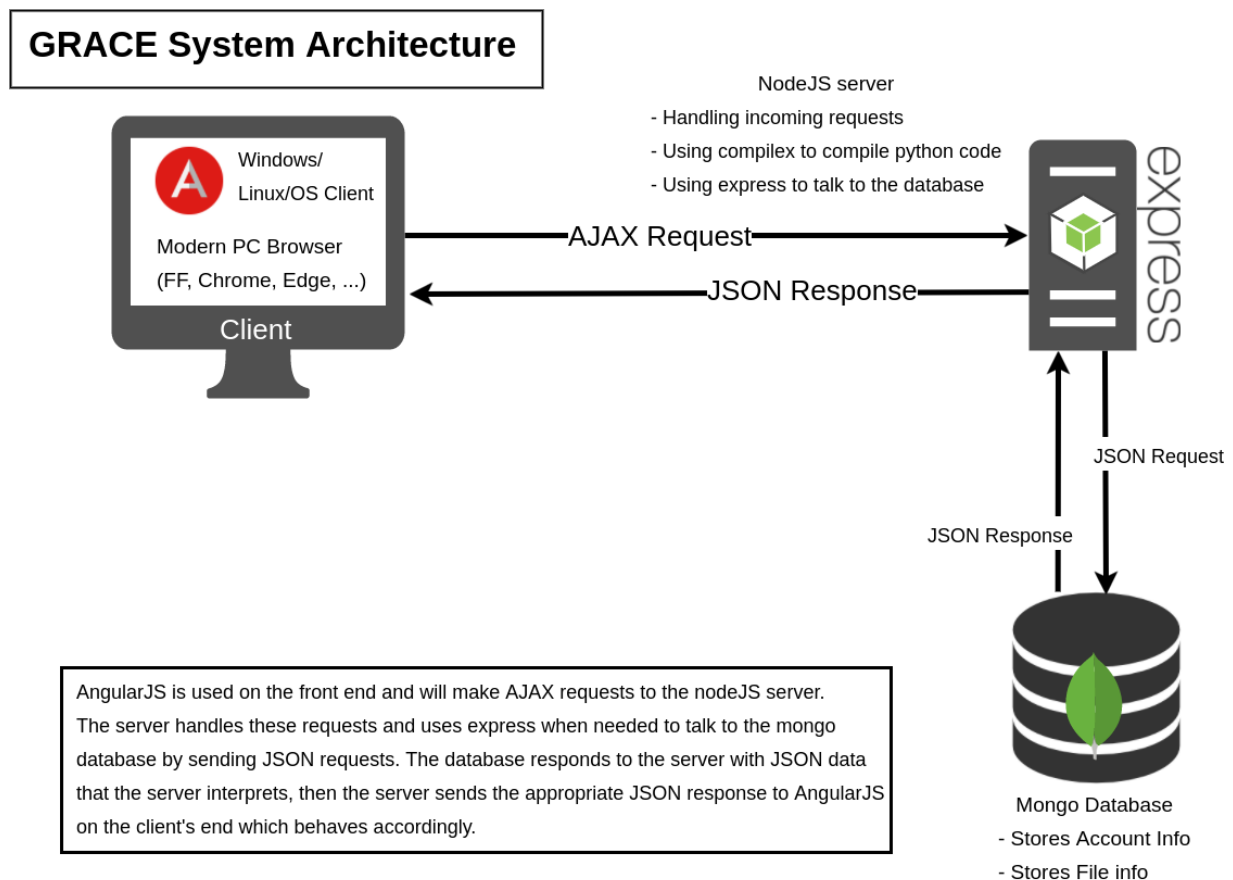
- Server: NodeJS, Express, Compilix (NodeJS library)
- Database: MongoDB
- Front-end: HTML, AngularJS, JavaScript

Most widely used PC operating systems support a modern browser such as Chrome, Firefox, or Edge.

Since these browsers support our front-end technology, and the ability to make the requests to our server, most users will be able to use our web application. Support for mobile browsers will not be present in the first release of our app but is being considered as a feature in the future.

The server will be running Linux and [MEAN](#) stack from Bitnami (version 3.4.2). NodeJS on our server will include the [Compilix](#) library (version 0.7.3).

System Architecture



System Decomposition

There are three main components in the GRACE REPL system architecture: client, server, and database.

The classes pertaining to the server component are File, Compiler, User, UserPreference, FileSystem, and Shell. The reason being is because the client will be making requests to the server which one or more of these classes will fulfill appropriately. More specifically, the compiler and shell in GRACE REPL will use a NodeJS library that will do the computation/interpretation on the server side. Similar to User, UserPreference, FileSystem, and File, they will be relaying on the server side receiving requests from the client while sending and receiving requests to the database.

The client component's classes are Navigator and Webapp since those classes relay the interaction between the user and the site. In addition, those interactions will essentially lead to Ajax requests to the server.

The database component consists of the classes, FileDAO, UserDAO, FileDAOImp, and UserDAOImp. Those classes deal with data access object and are in charge of storing data such as account and file information.

The system decomposition is on the system architecture diagram. It illustrates each component and its role in the higher-level architectural view.

The strategy for dealing with errors and exceptional cases will differ for each case. For the IDE and related errors such as third-party issues, the approach will be to have the users report the bug and then notify all GRACE REPL users of the existing bug. Furthermore, the GRACE team will try to resolve the issue since the Complex library is open source and notify the Complex developers.

For server connection issues, such as a user's inability to share and/or view files from other users, there are limitations for what the GRACE team can do. Connection issues occur all the time, so on our end the strategy is to prompt the user to refresh the page after a certain amount of time or wait until the connection is re-established. This strategy is the same for unexpected code compilation errors, since the server compiles/interprets the code.

Another case to consider is browser issues. Some less used web browsers may not be compatible with the site. The strategy will be to suggest the user of using a compatible browser; in the meantime, the team will try to alleviate the browser incompatibility.

Other general failures such as unexpected input to our backend will be dealt with by having multiple layers of validation that will eventually attempt to output an error message to the users.



DESIGN MOCK-UP



* elements are clickable

*login

*LOGO

language select

*code as guest

*register

* elements are clickable

*login

*LOGO

language select

*code as guest

Register

username

password

confirm password

email

*create account

* elements are clickable

*login

*LOGO

language select

*code as guest

*register

username

password

*login

*settings

*LOGO

* elements are clickable

*logout

Your files

↑ upload a file

📄 new file

✕ delete

🔒 make private

☒ *file name

last modified: 01/06/2017

↓ download

✉ invite



☐ file name

last modified: 01/06/2017

↓ download

✉ invite



☐ file name

last modified: 01/06/2017

↓ download

✉ invite



☐ file name

last modified: 01/06/2017

↓ download

✉ invite



☐ file name

last modified: 01/06/2017

↓ download

✉ invite



☐ file name

last modified: 01/06/2017

↓ download

✉ invite



☐ file name

last modified: 01/06/2017

↓ download

✉ invite



*settings

*LOGO

* elements are clickable

*logout



Your username

feature



feature



feature



 change password

*(this region is clickable)

1 replace file



new file



delete



make private

last modified: 01/06/2017

 download

 invite



last modified: 01/06/2017

 download

 invite



last modified: 01/06/2017

 download

 invite



last modified: 01/06/2017

 download

 invite



last modified: 01/06/2017

 download

 invite



last modified: 01/06/2017

 download

 invite



last modified: 01/06/2017

 download

 invite



*settings

*LOGO

* elements are clickable

*logout

↓ download

📁 save

▶ run

■ stop

✕ clear

```
1 print("Hello World")
```

Running python version 2.7.10

> Hello World!

> _

*back

*LOGO

* elements are clickable

↓ download

📁 save

▶ run

■ stop

✕ clear

```
1 print("Hello World")
```

Running python version 2.7.10

> Hello World!

> _

*settings

*LOGO

* elements are clickable

*logout



save



run



stop



clear

*(this region is clickable)

Running python version 2.7.10

> Hello World!

> _

feature



feature



feature



Invite URL

<https://www.grace.com/f8s9f7>

CP

delete

new invite