

# WebCheckers Project Design Documentation

Updated 4/21/2020

## Team Information

- Team name: C - Dingos
- Team members:
  - Frank Abbey
  - Raisa Hossain
  - Summer DiStefano
  - Stephen Bosonac

## Executive Summary

WebCheckers is a web-based implementation of American checkers. Users can play a game of checkers with other signed-in players. This project was built on the Spark web framework in Java 8.

## Purpose

The purpose of this WebCheckers game application is to allow users to play a game of checkers on a web browser. This game adheres to the American rules of checkers.

## Glossary and Acronyms

Term	Definition
MVP	Minimum Viable Product
VO	Value Object
UI	User Interface
Appl	Application
Util	Utility

## Requirements

This application features a functioning web version of the game Checkers. Players are able to sign in with a user name and select from a list of currently logged in users to play a game with.

The game follows the standard American Checkers rules and features the ability to resign and backup your moves before submitting them.

## Definition of MVP

The minimum viable product (MVP) is defined as a fully functioning game of American Checkers, complete with all basic rules and features of a standard Checkers game implemented:

## MVP Features

- Signing in and out of the game
- Piece movement (moving a single piece diagonally, one space)
- Jumping movement (jumping over an opponent's piece in order to capture it)
- King piece functionality
- Ending the game by taking all opponent pieces
- Ending the game by the opponent being unable to make a move
- Resigning from a game

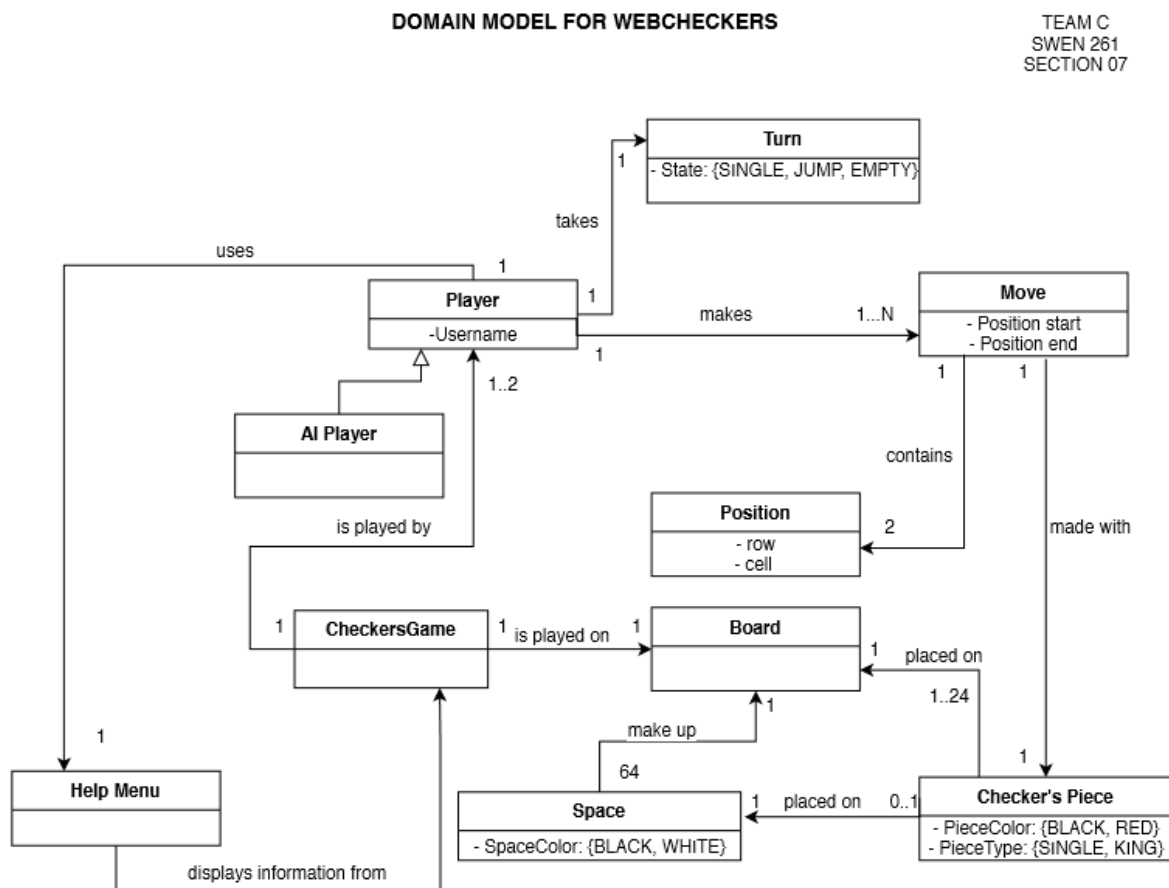
## Roadmap of Enhancements

- A.I. Player available to chose as opponent
- Help Menu to point out available moves and number of pieces taken\*

\*(not implemented)

## Application Domain

The Domain Model represents the relationships and interactions between items involved with the MVP. The model does not display code-specific classes and structures, only a high level concept used to understand how a Checkers Game should flow in the context of a web application.



## ## Architecture and Design

The application architecture breaks down into 3 main tiers:

1. **UI Tier:** route handlers, classes that handle requests and responses
2. **Appl Tier:** logical flow of the application

### 3. Model Tier: business logic of the domain

#### Summary

As a web application, the user interacts with the system using a browser. The client-side of the UI is composed of HTML pages with some minimal CSS for styling the page. There is also some JavaScript that has been provided to the team by the architect.

The server-side tiers include the UI Tier that is composed of UI Controllers and Views. Controllers are built using the Spark framework and View are built using the FreeMarker framework. The Application and Model tiers are built using plain-old Java objects (POJOs).

Details of the components within these tiers are supplied below.

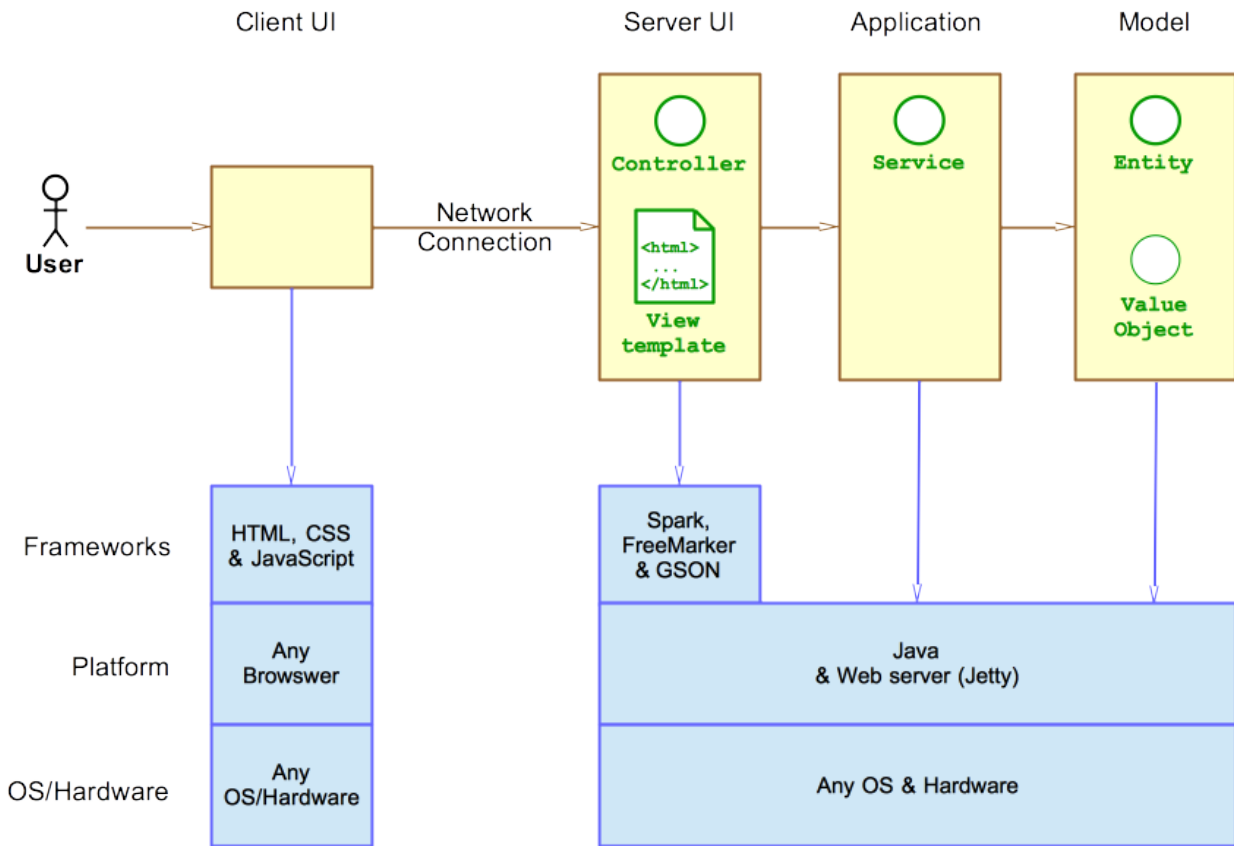


Figure 1: The Tiers & Layers of the Architecture

#### Overview of User Interface

The User interface/experience begins at the **home page** ("/"). Here the User is presented with the number of other Users currently signed onto the system. From there, the User can click **sign-in** to be taken to the **sign-in page** ("/signin"). The User will be presented with a text field to enter a **username**, and upon successful submission will be re-directed back to the **home page**. If their username is deemed invalid, they are notified and remain on the **sign-in page**.

Once a User is signed in, they are presented with the same list of other User's signed in, only now the their user names are displayed. Each username is now a link to commence a game with that User. If one is clicked on, both Users are directed to the **game page** ("/game") to play a game together.

Users will remain in the **game page** until the game has ended.

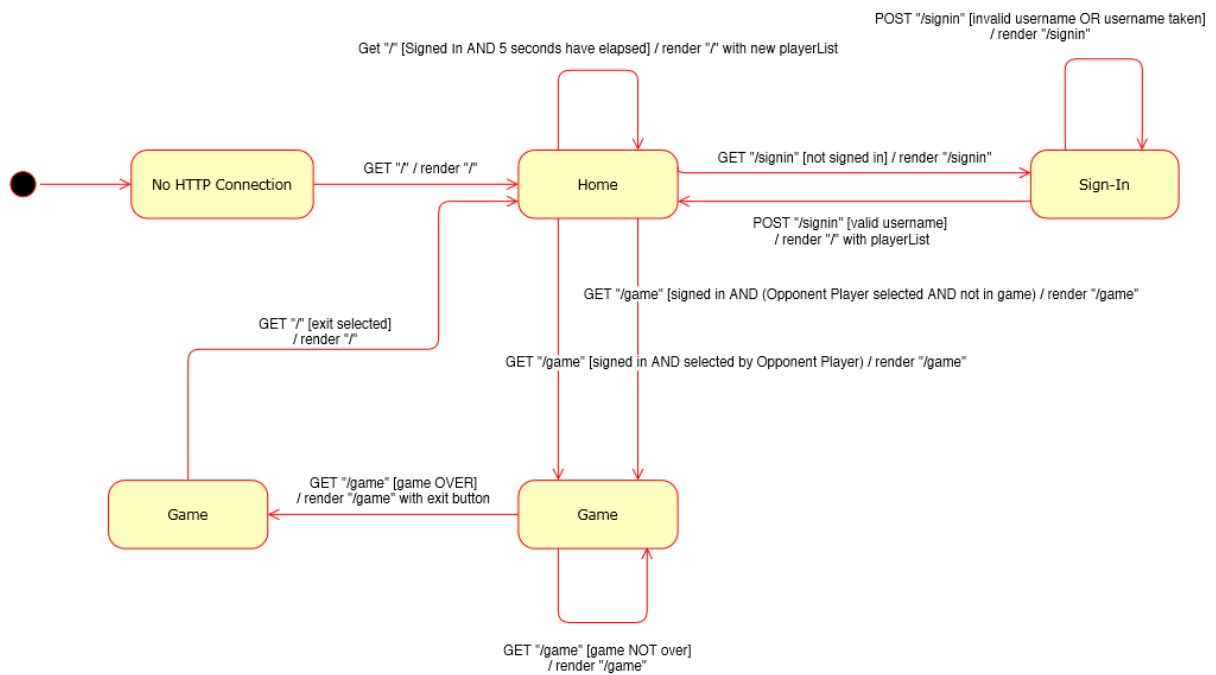


Figure 2: The WebCheckers Web Interface Statechart

## UI Tier

The **UI tier** handles the server-side functionality of the application. The **WebServer** class handles the initialization of the HTTP request handlers. **Application** level items such as the **GameCenter** and the **PlayerLobby** along with the template engine are passed to the various route handlers through their constructors.

Regarding the process of move validation, the **PostValidateMoveRoute** pulls the current **Player** and **Move** objects from the HTTP request.

The **PostValidateMoveRoute** then passes the information over to the **Model tier** class, **Turn**.

## Application Tier

The Application tier contains the logical flow of the application.

- **PlayerLobby**

Important mostly in handling the logic of Player sign in, the PlayerLobby keeps a list of currently logged in users. This class also handles username verification and Player retrieval.

- **GameCenter**

This class is used heavily during game play. It contains the list of all active CheckersGames and can determine if a player is in a particular game (a very crucial part of the domain logic).

## Model Tier

The Model tier of the application contains the underlying business logic of the domain classes. Items such as move validation, removing a piece after a jump move, and turning a piece into a King all happen within these classes.

## Design Improvements

Some game logic was consolidated. In designing the **AI Player** taking their turn, originally the game logic resided in the **GetGameRoute**, but was later handed off completely to the **AIPlayer** class.

Future improvements to the project include: - More advanced AI - Help Menu - Easier to follow application flow

## Testing

### Acceptance Testing

All acceptance criteria have been tested and passed.

Our only concern was that the player that Resigned from a game is redirected to their Home page, but this might be a feature of the front end code.

### Unit Testing and Code Coverage

For unit testing, first we wanted to tackle a class from every tier in order to have a mold we could follow when creating other unit tests for classes within that same tier.

After that, we targeted classes that were used heavily during Sprint 2 of the project. We wanted to make sure that the most non-trivial pieces of these classes were being tested properly.

In Sprint 3, we wanted to tackle more crucial Unit Tests, and get as many done as we could in the time allotted.

The total coverage of our unit tests is still relatively low, as we are missing coverage in highly used areas of the application.

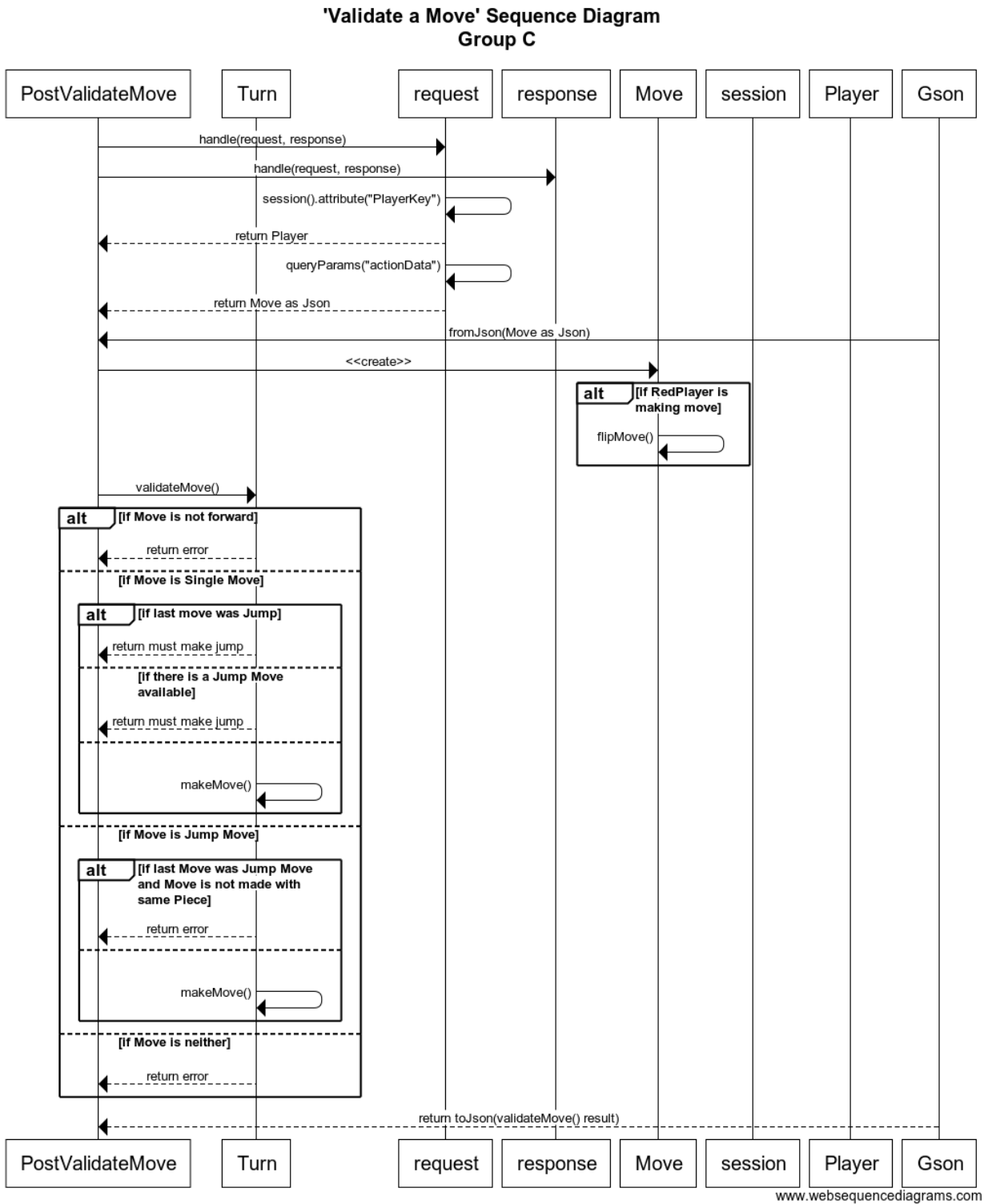


Figure 3: Move Validation Sequence Diagram

## Code Coverage Reports

### Application Tier Coverage

#### com.webcheckers.appl






Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
GameCenter		8%		0%	13	14	22	25	7	8	0	1
PlayerLobby		92%		76%	7	21	3	33	1	8	0	1
PlayerLobby.LoginResult		100%	n/a	n/a	0	1	0	1	0	1	0	1
Total	92 of 260	64%	18 of 38	52%	20	36	25	59	8	17	0	3

Figure 4: Application Tier Coverage Report

### Model Tier Coverage

#### com.webcheckers.model




































Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Board		41%		39%	40	70	89	169	8	21	0	1
CheckersGame		28%		6%	28	37	55	82	13	22	0	1
AIPlayer		0%		0%	19	19	43	43	5	5	1	1
Move		76%		64%	22	51	14	70	2	16	0	1
Turn		71%		50%	14	25	17	70	2	10	0	1
Position		33%		0%	7	11	5	15	2	6	0	1
CheckersGame.ActiveColor		0%	n/a	n/a	2	2	3	3	2	2	1	1
Player		60%		66%	3	8	6	14	1	5	0	1
Space		78%		50%	7	17	1	19	1	9	0	1
Row		82%		50%	5	14	3	23	2	8	0	1
Piece		89%		60%	4	12	0	14	0	7	0	1
BoardView		100%		100%	0	6	0	16	0	4	0	1
Turn.State		100%	n/a	n/a	0	1	0	5	0	1	0	1
Row.new Iterator().{...}		100%		75%	1	5	0	4	0	3	0	1
Board.new Iterator().{...}		100%		75%	1	5	0	4	0	3	0	1
BoardView.new Iterator().{...}		100%		75%	1	5	0	4	0	3	0	1
CheckersGame.State		100%	n/a	n/a	0	1	0	4	0	1	0	1
Space.SpaceColor		100%	n/a	n/a	0	1	0	1	0	1	0	1
Piece.PieceType		100%	n/a	n/a	0	1	0	1	0	1	0	1
Move.Type		100%	n/a	n/a	0	1	0	3	0	1	0	1
Piece.PieceColor		100%	n/a	n/a	0	1	0	1	0	1	0	1
Total	1,263 of 2,872	56%	188 of 326	42%	154	293	236	562	38	130	2	21

Figure 5: Model Tier Coverage Report

### UI Tier Coverage

### Util Tier Coverage

## com.webcheckers.ui

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
<a href="#">GetGameRoute</a>	<div><div></div></div>	0%	<div><div></div></div>	0%	14	14	60	60	3	3	1	1
<a href="#">WebServer</a>	<div><div></div></div>	0%	<div><div></div></div>	n/a	3	3	23	23	3	3	1	1
<a href="#">GetHomeRoute</a>	<div><div></div></div>	43%	<div><div></div></div>	8%	6	9	19	38	0	3	0	1
<a href="#">PostCheckTurnRoute</a>	<div><div></div></div>	0%	<div><div></div></div>	0%	6	6	17	17	2	2	1	1
<a href="#">PostValidateMoveRoute</a>	<div><div></div></div>	0%	<div><div></div></div>	0%	4	4	14	14	3	3	1	1
<a href="#">PostSignOutRoute</a>	<div><div></div></div>	16%	<div><div></div></div>	0%	2	3	13	17	1	2	0	1
<a href="#">PostResignGameRoute</a>	<div><div></div></div>	0%	<div><div></div></div>	0%	3	3	9	9	2	2	1	1
<a href="#">PostBackupMoveRoute</a>	<div><div></div></div>	0%	<div><div></div></div>	n/a	2	2	9	9	2	2	1	1
<a href="#">PostSignInRoute</a>	<div><div></div></div>	87%	<div><div></div></div>	66%	2	8	4	36	0	5	0	1
<a href="#">PostExitGameRoute</a>	<div><div></div></div>	0%	<div><div></div></div>	n/a	2	2	4	4	2	2	1	1
<a href="#">GetSignInRoute</a>	<div><div></div></div>	84%	<div><div></div></div>	50%	1	4	1	12	0	3	0	1
<a href="#">PostSubmitTurnRoute</a>	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	2	0	6	0	2	0	1
Total	871 of 1,162	25%	50 of 56	10%	45	60	173	245	18	32	7	12

Figure 6: UI Tier Coverage Report

## com.webcheckers.util

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
<a href="#">Message</a>	<div><div></div></div>	82%	<div><div></div></div>	0%	3	9	2	12	2	8	0	1
<a href="#">Message.Type</a>	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	1	0	2	0	1	0	1
Total	12 of 92	86%	2 of 2	0%	3	10	2	14	2	9	0	2

Figure 7: Util Tier Coverage Report