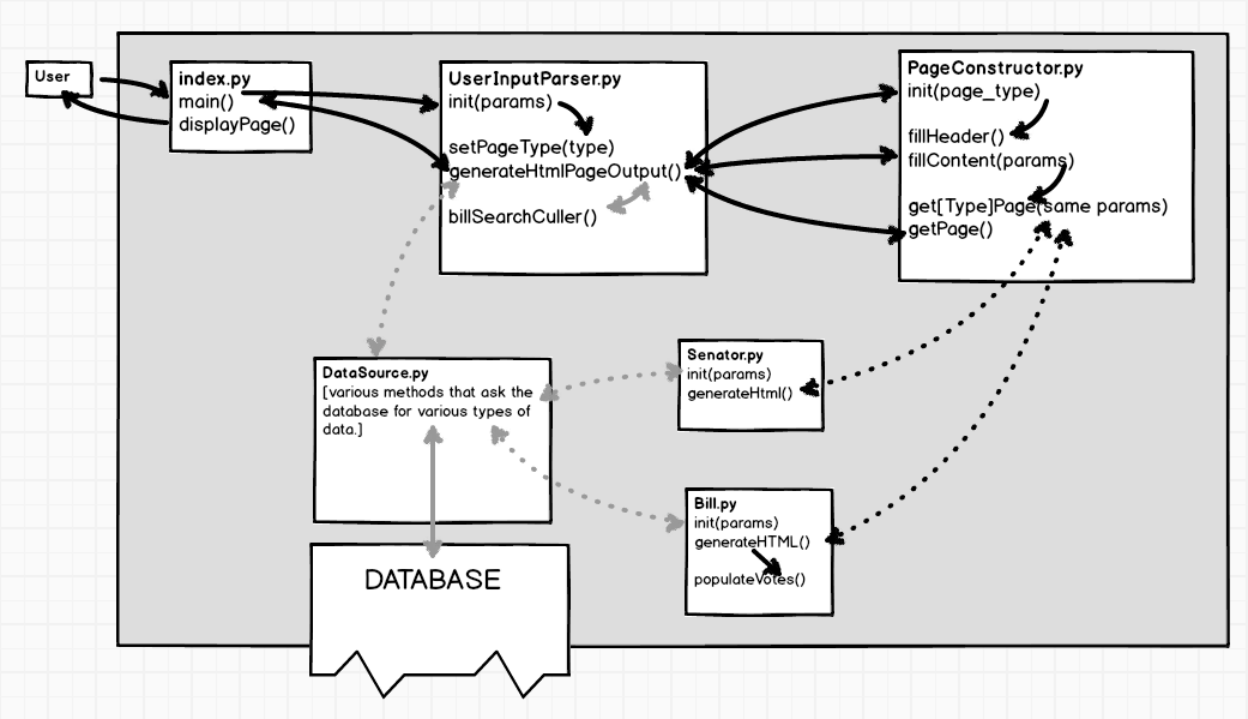
System Architecture



Deliverables

Model

//class that takes the user input from params and decides what information it needs to collect from the database and then pass onto the controller.

Class **UserInputParser**

Variables

- page_type - a behind the scenes variable that get set when the parser reads the type parameter from the CGI generated params.
- params - storage for the passed CGI input from the user that was passed into the constructor
- page_constructor - a variable that stores a new page constructor object that is built at initialization.

Methods

- **__init__(***self, [params]***)**: Constructor that stores parameters internally for later calling as well as calling setPageType() so the class knows what it will be trying to return later.
    - Testing: We initialize UserInputParser with a set of parameters, then get those parameters back to make sure they're accurate.
- **setPageType(***self, type***)**: Reads the page variable from the passed by the user and figures out what page_type the webapp should try to generate. It will also create a new PageConstructor object with the proper type of page primed and stored as a page_constructor variable
    - Testing: We call setPageType with a given type and check to make sure nothing's wrong.
- **generateHtmlPageOutput(**self**)**: Based the page_type it grabs the relevant variables from params (with potential for throwing an invalid user input exception). The class then uses those variables from params to query Database.py to grab the requested information and feed it to page_constructor to build the page. Then returns the html from page_constructor. (note much logic occurs here. If there is a search query in the user input it calls billSearchCuller to remove from the list any bills that do not meet the criteria after the database search.
    - Testing: Give it the parameters and see how it runs the gauntlet of if/else statements. Give it invalid parameters. Check whether it throws the exceptions.
- **billSearchCuller(**self, [bill_list], search_param**)**: takes a list of Bills and a search criteria and returns a new list of bills (empty list if none meet the criteria) with everything that meets the criteria. (ex. takes a list of bills from the 114th congress and returns only the bills that "John Snow" voted on)
    - Testing: We throw the function a variety of bills and search criteria, making sure to include both criteria that remove all the bills and criteria that don't affect any of them.

//class that stores the database information pulled about a specific senator with some rudimentary link generating power.

Class **Senator**

Variables

- (*gettable*)name - the senators name
- (*gettable*)id - the id tag value for the senator
- (*gettable*)state - string representation of the senators state
- (*gettable*)congresses_served - list of the id tag values for each congress that the senator has served in.
- (*gettable*)party - party the senator is affiliated with

Methods

- **__init__(***self, id, [params]***)**: Constructor that builds a senator object from the senators id and an array of string parameters containing the variables from the senator as generated by the database interface.
    - Testing: We create a Senator with various parameters and use the get functions to verify that it's working correctly. (This involves both checking that variables are correct as well as that they're in a standard form - e.g. that vote_tally is a 2xN array.)
- **generateHtmlNameLink(***self***)**: Generates a link to the senator's page with the properly formatted text and returns it as a string (for example, (R-AK) Richard White).
    - Testing: We run the function on our sample Senator, comparing both the link's text and its url with what we expect.

//class that stores the database information pulled about a specific bill with the ability to further query the database to get the list of senators and who voted on it when asked, though this is not done by default

Class **Bill**

Variables

- (*gettable*)name - the bills name
- (*gettable*)id - the id tag value for the bill
- (*gettable*)hr - the HR tag for the bill
- (*gettable*)date - a string representing the date the bill was voted on
- (*gettable*)congress - the id of the congress that voted on the bill
- (*gettable*)description - a string representing a description of the bill as is present in our data source.
- (*gettable*)vote_tally - a list of length 2 string list objects that contains a vote (e.g. Yea, Nay, Abstain, Absent) and a senator object corresponding to who cast that vote. Example:

"Richard White"	"John Smith"	"Clint Williams"
"Yea"	"Nay"	"Absent"

Methods

- **__init__(***self, id, [params]***)**: Constructor that builds a bill object given an ordered list of strings generated by the database interface (this leaves vote_tally empty by default)
    - Testing: We create a Bill with various parameters and use its get functions to make sure everything's in order. This involves both checking that variables are correct as well as that they're in a standard form - e.g. that vote_tally is a 2xN array.
- **populateVotes(***self***):** Speaks to the database interface and populates vote_tally with the requisite senators and their votes from the database
    - Testing: We send a request to our database and check that we get back the proper information.

- **getBillLink(***self***)**: Generates a link to the bill's page with the properly formatted HTML text and returns it as a string(e.g. HR578 - Motion to withdraw the Motion [Passed]).
    - Testing: We run the function on our sample bill, comparing both the link's text and its url.

View

//class that accepts input from UserInputGenerator in the form of lists of senators and bills and uses template classes to fill out the html formatting for the page then return the html code for the page as a string

Class **PageConstructor**

Variables

- page_type - variable that stores what type of page has been given to the constructor for what type of page is to be made
- page - variable that stores the current state of the page as it has

Methods

- **__init__(***self, page_type***)**: Constructor that builds a PageConstructor object and stores what type of page this constructor is building then calls fillHeader(self) and fillContent(self) to populate the page.
    - Testing: Try invalid page types. Also test whether the html matches what is expected.
- **fillHeader(***self***)**: method that reads headerTemplate.html and makes changes to it based on page_type (for example graying out the current page link) then adds it to the beginning of page
    - Testing: Try invalid page types. Also test whether the html matches what is expected.
- **fillContent(***self, params***)**: Checks the page type and calls the appropriate method to fill the main content of the page, passing along the parameters.
    - Testing: Try invalid page types. Also test whether the html matches what is expected
- **getPage(**self): Returns the page variable.
- **makeHomepage(**self,[bill_list]**)**: Gets html from homepageTemplate.html, fills in a list of the most recent senatorial activity, then appends the now-complete html for the homepage to the page variable.
    - Testing: Tries to get homepage string with a list of known bills and compares it to a hand crafted hompage html text representing what we hand craft (using get page for return)
- **makeBillPage(**self,Bill bill**)**: Gets html from billPageTemplate.html, fills in the info of the bill passed to it, and appends to the page variable.
    - Testing: Tries to craft bill page string with a list of known bill and compares it to a hand crafted hompage html text representing what we hand craft (using get page for return)

- **makeBillComparisonPage(**self,Bill this_bill,Bill that_bill**)**: Gets html from billComparisonTemplate.html, fills in the info of the *two* bills, and appends to the page variable.
    - Testing: Tries to craft bill comparison page string with a list of known bill and compares it to a hand crafted hompage html text representing what we hand craft (using get page for return)
- **makeBillSearchQueryPage(**self,[Bill],Senator=empty**)**: Generates the html formatting for a given search query page and adds it to the already partially filled in page variable then returns the html code for the page
    - Testing: A more extensive set of known bills and senator combinations will be fed to it and compared to hand crafted html page strings representing how the page should look (using get page for return)
- **makeSenatorPage(**self,Senator senator**):** Gets html from senatorPageTemplate.html, fills in the senator's info, and appends to the page variable.
    - Testing: Give the page a known senator and compares what it generates to hand crafted html page strings representing how the page should look (using get page for return)
- **makeStatePage(**self,String stateName, [Senator]**):** Gets html from statePageTemplate.html, fills in the state name and senator list, and appends to the page variable.
    - Testing: Give the page a known state and list of senators belonging to that state (importantly having some that are current and non current) what it generates to hand crafted html page strings representing how the page should look (using get page for return)
- **makeAllStatesPage(**self**)**: Gets html from allStatesPage.html (literally a list of states) and appends it to the page variable.
    - Testing: compares what is generated to the template page it comes from (using get page for return)
- **makeCongressPage(**self, int congress,[senator_list],[bill_list]**)**: Gets html from congressPage.html and adds into it a list of all the senators from the congress and the last few bills from the congress
    - Testing: Give the page a known list congress and a known set of the most recent bills from that congress as well as senators and compare it to a hand crafted html string representing the page (using get page for return)

Templates

headerTemplate.html

footerTemplate.html
homepageTemplate.html
billPageTemplate.html
billComparisonTemplate.html
billSearchQueryTemplate.html
senatorPageTemplate.html

statePageTemplate.html
statePage.html
allStatesPage.html
congressPage.html

Controller

Class **index.py**

main()
-calls CGI to get user input as a list of params
-passes params to a new UserInputParser
-requests a page from UserInputParser, prints out the page

displayPage(self**)**: Prints the finished html page to standard output.

CGI

- It's a magical black box that returns gets all of the variables that we pass into the page and returns them as params

Database

Class **DataSource.py**

- **__init__(***self***)**: Constructor.
- **getSenatorList(***self***)**: Returns a list of Senator objects for all senators in the senate vote database.
  - Testing: This is a "does it work" sort of thing. Check the size of the list, the first element, the last element, and some random ones.
- **getSenator(***self, id***)**: Returns a Senator object that matches the id number.
  - Testing: Call multiple id numbers and test whether the matching senators are those we expect. Call invalid id numbers.
- **getBillList(***self***)**: Returns a list of the id numbers of all bills in the senate vote database.
  - Testing: This is a "does it work" sort of thing. Check the size of the list, the first element, the last element, and some random ones.
- **getBill(***self, id***)**: Returns a Bill object that matches the id number.
  - Testing: Call multiple id numbers and test whether the matching bills are what we expect. Call invalid id numbers.
- **getSenatorsInCongress(***self, congress***)**: Returns a list of Senator objects for every senator in the specified congress (for example 114 would return all current senators).
  - Testing: Call multiple senates and check whether it works properly. Call invalid senates – negative? 0? too big?

- **getBiographyForSenator(***self, senator_id***)**: Returns an ordered list of strings containing all of the biographical data on the senator including, name, state, party affiliation, years in office, and an arbitrary number of objects at the end which correspond to the congresses that this senator is affiliated with.
    - Testing: As is obligatory, try invalid ids. Otherwise, compare the strings.
- **getVotesBySenator(***self, senator_id, number***)**: Returns a list of the last number Bill_id's that the senator has voted on (with an Absent being ignored). If the number field is left as 0 it scans the entire database for all the bills voted on by the senator.
    - Testing: Test multiple senators, with multiple numbers of bills. Run it with a negative number, zero, and with a float.
- **getBillsInCongress(***self, congress, number***)**: Returns a list of the id numbers for the last number bills in congress in reverse chronological. Specifying more bills than are in the congress returns all available bills and specifying zero bills returns all bills.
    - Testing: Call multiple senates and check whether it works properly. Call invalid senates – negative? 0? too big?
- **getBillBiography(***self, bill_id***)**: Returns an ordered string list that contains the bill name, the bill HR number, its congress, date voted on, and its description for being placed into a bill class.
    - Testing: As is obligatory, try invalid ids. Otherwise, compare the strings.
- **getVotesForBill(***self, bill_id***)**: Returns a two dimensional list of strings that correspond to a Senator_ID and their vote on the specified bill. (Yea, Nay, Abstain, Absent? Something like that.)
    - Testing: As is obligatory, try invalid ids. Otherwise, compare.
- **getSenatorsInState(***self, state_name***)**: Returns a list that corresponds to the id number for each senator in a given state, with the name for each state given as a string.
    - Testing: As is obligatory, try invalid ids. Otherwise, compare the strings.

Development Schedule

- Phase 5:
    - [6-M] Scrape Dataset from source (completed)
        - Begin filling out class and method structure
    - [6-Tu] (5/5) Database scraped from source
        - Turn in design document
    - [6-W] (5/6) Cull downloaded dataset to fit space requirements
    - [6-Th] (5/7) Datasets are loaded into the database. (due)
    - [6-F] (⅝) Dummy page (formatting template for what a generic page will look like)
    - [6-Sat] (5/9) continue work on methods
        - complete html templates for core pages
        - finish all non-formatting related methods and tests

    - [6-Sun] finish core functionality, have all basic html functions complete
        - dataset returns correct values for querys
- Phase 6:
    - [7-Mon] (5/11) Version 1 *due*.
        - catch up to schedule/refine html formatting
    - [7-Thur] (5/11) complete html templates for stretch features
    - [7-Sat] complete stretch features (comparison page, infographics)
- Phase 7:
    - [8-Mon] (5/18) Final version due.