Chris Weeden

10/26/2018

CSC 415

Assignment 4

Server: server21

Full pathname: /local/home/sysadmin/devEnv/growstuff

Added code will be in path: growstuff/app/services (this is where their search function is used)

Since the wiki is not limited to branches I cant add this to it without disrupting other people. I have the same issue with making issues and milestones since they are global across branches. I have added a temporary docs folder on the vm for the purposes of this assignment.

Use case: Search for a plant by its main name

Iteration: 1, last modification: 10/24/2018

Primary actor: User using search function.

Preconditions: User must be searching a valid main name for a plant;

System must be fully completed.

Trigger: User presses search button with the appropriate field filled in

Scenario:

1. User enters a name into the search field
2. System searches through database for a match on main names
3. System outputs correct plant that is a match

Exceptions:

1. User enters empty search field- see use case **empty string**
2. User enters secondary name- see use case **search for a plant by a secondary name**
3. User enters a misspelled name- see use case **search for a plant by a close approximation of it's name / misspelling**
4. User enters random string- see use case **search for a plant by a close approximation of it's name / misspelling**
5. User enters a name not in the system- see use case **search for a plant by a close approximation of it's name / misspelling**

Priority: High, base functionality

When available: First increment.

Frequency of use: Frequent.

Channel to actor: Via PC-based browser web-app

Secondary actors: GrowStuff database, GrowStuff webserver.

Channels to secondary actors:

1. GrowStuff database: postgreSQL
2. GrowStuff webserver: PC-based system

Open issues:

1. How do we stop someone from entering in random characters and taxing the system for no good useful

Use case: Search for a plant by a secondary name

Iteration: 1, last modification: 10/24/2018

Primary actor: User using search function.

Preconditions: User must be searching a valid secondary name for a plant;
System must be fully completed.

Trigger: User presses search button with the appropriate field filled in

Scenario:

1. User enters a name into the search field
2. System searches through database for a match on main names
3. System will find no matches on main names
4. System searches database for a match on secondary names
5. System outputs correct plant that is a match

Exceptions:

1. User enters empty search field- see use case **empty string**
2. User enters main name- see use case **search for a plant by a main name**
3. User enters a misspelled name- see use case **search for a plant by a close approximation of it's name / misspelling**
4. User enters random string- see use case **search for a plant by a close approximation of it's name / misspelling**

5. User enters a name not in the system- see use case **search for a plant by a close approximation of it's name / misspelling**

Priority: medium, secondary functionality wanted by GrowStuff issue

When available: First increment.

Frequency of use: Infrequent.

Channel to actor: Via PC-based browser web-app

Secondary actors: GrowStuff database, GrowStuff webserver.

Channels to secondary actors:

1. GrowStuff database: postgreSQL
2. GrowStuff webserver: PC-based system

Open issues:

1. How do we stop someone from entering in random characters and taxing the system for no useful output

Use case: Search for a plant by a close approximation of it's name / misspelling

Iteration: 1, last modification: 10/24/2018

Primary actor: User using search function.

Preconditions: User must be searching a valid secondary name for a plant;
System must be fully completed.

Trigger: User presses search button with the appropriate field filled in

Scenario:

1. User enters a name into the search field
2. System searches through database for a match on main names
3. System will find no matches on main names
4. System searches database for a match on secondary names
5. System finds no matches on secondary names
6. System computes edit distances on both secondary and main names keeping the 10 most promising matches
7. System outputs list of top ten matches

Exceptions:

1. User enters empty search field- see use case **empty string**

2. User enters main name- see use case **search for a plant by a main name**
3. User enters secondary name- see use case **search for a plant by a secondary name**

Priority: Low, to be implemented after secondary name search

When available: First increment.

Frequency of use: Frequent.

Channel to actor: Via PC-based browser web-app

Secondary actors: GrowStuff database, GrowStuff webserver.

Channels to secondary actors:

1. GrowStuff database: postgreSQL
2. GrowStuff webserver: PC-based system

Open issues:

1. How do we stop someone from entering in random characters and taxing the system for no useful output

Use case: empty string

Iteration: 1, last modification: 10/24/2018

Primary actor: User using search function.

Preconditions: User must be searching a valid secondary name for a plant;
System must be fully completed.

Trigger: User presses search button with the appropriate field filled in

Scenario:

1. User enters an empty string into the search field
2. System defaults to showing empty results screen

Exceptions:

1. User enters main name- see use case **search for a plant by a main name**
2. User enters secondary name- see use case **search for a plant by a secondary name**
3. User enters a misspelled name- see use case **search for a plant by a close approximation of it's name / misspelling**
4. User enters random string- see use case **search for a plant by a close approximation of it's name / misspelling**

5. User enters a name not in the system- see use case **search for a plant by a close approximation of it's name / misspelling**

Priority: low, implementation only after all others are done

When available: First increment.

Frequency of use: Very infrequent.

Channel to actor: Via PC-based browser web-app

Secondary actors: GrowStuff database, GrowStuff webserver.

Channels to secondary actors:

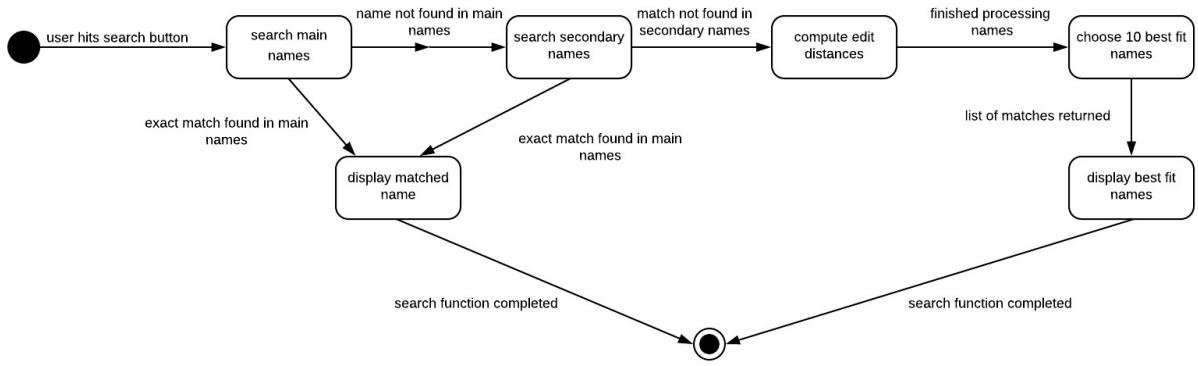
1. GrowStuff database: postgreSQL
2. GrowStuff webserver: PC-based system

Open issues:

1. How do we stop someone from entering in random characters and taxing the system for no useful output

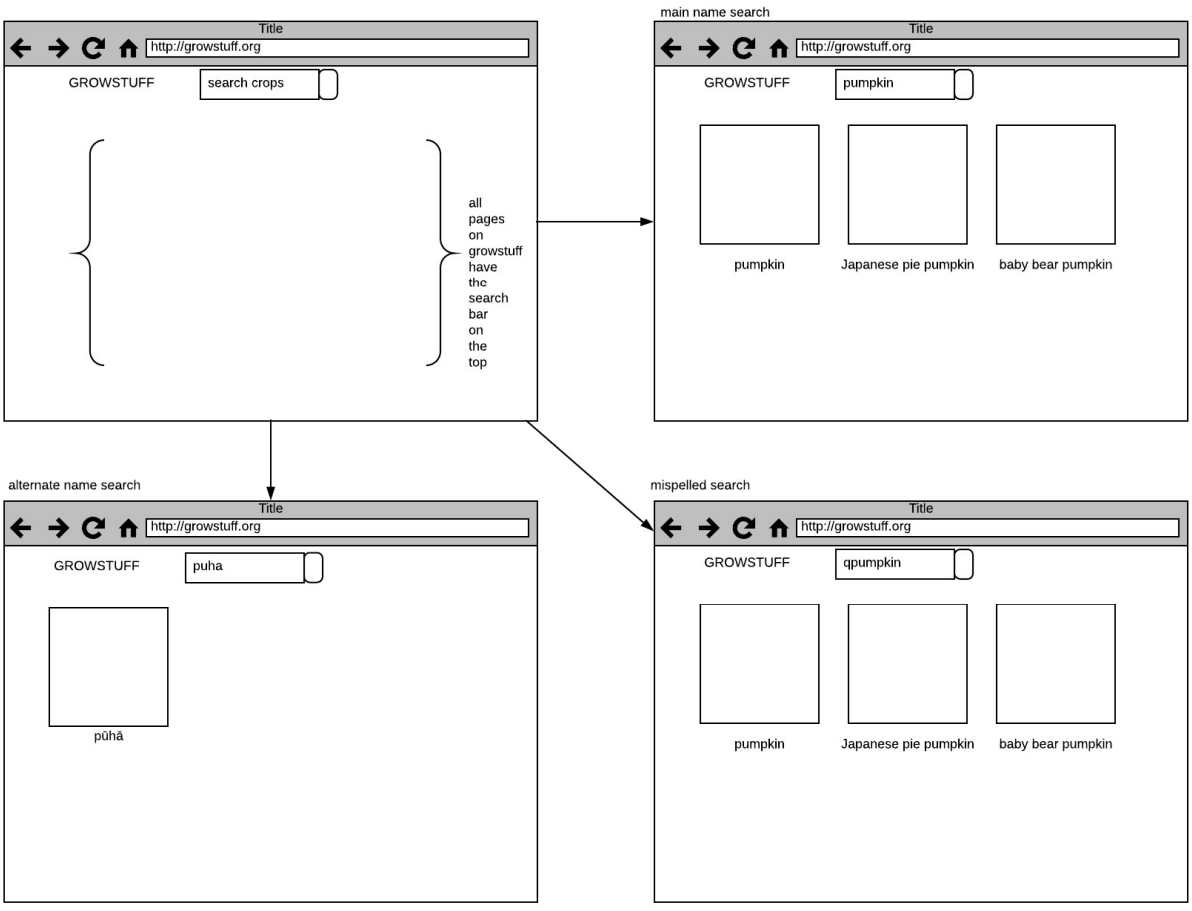
Design class diagram: *

State chart:



System sequence diagram: *

UI:



UI:

The UI doesn't do anything other than use already in place UI that growstuff already uses for its search function. It shares the same color scheme as the rest of the website making it consistent. There are no shortcuts for search functions and without heavy modification to the

database no way to store users' inputs. Right now, if nothing is found as an exact match it returns a generic error of no crops found I plan to rectify that by always displaying some output either exactly what the user wanted or a close approximation. There is a very clear sequence of steps for the user and the system takes care of the rest and after the user clicks the search button the page refreshes and loads the search results giving immediate feedback. Error handling is done in the sequence of events where we assume the user enters a main name-secondary name then assumes the input was spelled incorrectly and then does that computation. Since the last part is a catch all there is no user input that could cause the system to fail. Since the search function is always somewhere on the growstuff pages there is always a way to easily reverse their actions. By being able to choose what to search we maintain the user's internal locus of control since they directly influence what output is presented to them. Lastly this project of enhanced search functionality works to lighten the short-term memory load on the user since when the project is finished they shouldn't need to remember the exact name of the plant that they want information on or what it is mainly called.

Design:

Modularity and encapsulation: by splitting off my main algorithm into multiple functions and its own class can help with modularity as each split requires fewer specific inputs and outputs e.g. the misspelled full function takes in 2 arrays of names and a string but split into multiple parts and looped can receive 2 strings and return an integer that is the edit distance between the two strings which is more readily usably for other problems not related to a search functionality. By splitting up the program into smaller pieces and grouping them together in a larger piece you allow people to use your code without expressly knowing how it works.

Elegance and efficiency: efficiency is being kept under consideration by making sure only when necessary do we preform the edit distance function since it will be the heaviest hit on completion time. There can be improvements in this area and it is up to me to make sure that when a better solution is found that the ground work is already done for it.

Appropriateness of data structures used: Right now, GrowStuff uses arrays and string compares for its search function. Ruby seems to be fast enough with array insertions and deletions so there's no urgent need to change that however a doubly linked list might run faster. For what I will implement a weighted queue will be good for keeping the list of closest matches sorted and also, I can add a quick way to make sure that I can delete the tail after ten entries have been inserted allowing for a constant time deletion method.

Testing types:

Unit testing: During development It can be easy to separate my 4 use cases (empty search, main name, secondary name, misspelled name) since none of them rely on each other to function just that the ones before it fails and as such I can test each one individually to make sure output is correct. Main name and secondary name searches are easy to make test cases for since they just look for exact matches in the database and return a list of matched objects. Misspelled

names will need extensive unit test to ensure proper function and output e.g. If I put in qpumpkin I should get about the same results as searching pumpkin. I can split up the misspelled search into more parts and test them individually like the edit distance algorithm and weighted queue function to make sure each piece works as it should before moving into integration testing.

Integration testing: This will be done by top down integration testing as the most important/main control modules are finished first. The main module is the main name search once this is done we integrate the secondary name search and then the misspelled name search. The misspelled name search is itself broken into different parts like the edit distance algorithm and its weighted queue and must be put together piece by piece while being tested in between revisions. This is good since the driver to test the component won't need to change very much just what string it inputs for the search

System testing: I don't need to worry about most parts of system testing since my project is an expansion of a select part of an established web app. Recovery is unnecessary as putting in some recovery functionality into a search function might take up too much resources for no foreseeable gain. Security might be an issue but if input is sanitized correctly there should be no problem. Stress testing is important however since the misspelled names function could fail strangely on some specific inputs requiring much more testing than the other parts. There might also be a huge performance hit from repeatedly calling the misspelled name function since the algorithm used to calculate some of the data is very expensive computation wise, time might need to be spent in finding an alternate solution should it prove to be too much for the server to handle. Last is deployment testing and since this is a rails web app it should already run on most browsers which already run on most operating systems, so the only thing left for it is to make sure it runs correctly under some of the most popular web browsers.

Debugging:

Ruby on rails has an inbuilt debugger and ruby itself has a gem called ruby-debug which I will be using. Since they are purpose built for ruby and rails specifically they should help me catch stray values and what talks to what in growstuff's application itself without having to look at every file in the growstuff repository.

Test cases:

Functionality tested	Inputs	Expected output	Actual output
No input	NULL	The search results page empty	
Main name search	pumpkin	The search results page with every plant that has pumpkin in its name: pumpkin, Japanese pie pumpkin, Atlantic giant pumpkin, baby bear pumpkin, crown pumpkin	

All caps main name	PUMPKIN	pumpkin, Japanese pie pumpkin, Atlantic giant pumpkin, baby bear pumpkin, crown pumpkin	
Secondary name search	puha	The search results page with every plant that has the searched term as a secondary name: pūhā	
All caps secondary name	PUHA	pūhā	
Misspelled name	Tormato (misspelling of tomato)	The search results page with the top 10 results closest to the entered term: tomato, potato, beefsteak tomato…	
All caps Misspelled name	Tormato (misspelling of tomato)	tomato, potato, beefsteak tomato	
main name with special characters	a!!p:le}}”	Same output for apple: Apple, apple mint…	
Secondary name with special characters	Purpl$$%@#$^%#e granadilla	Passion fruit	
Misspelled name with special characters	Gral^&*e (misspelling of grape)	Grape, grapefruit, grape tomato, borage(3 matches with 3 indel’s), … should have less matches/more indel’s as the list progresses	

*design class and system sequence diagram are in separate pdf documents