Getting comfortable with LegCRM in 5 Sessions

Five Sessions

- The PHP Language as Implemented in LegCRM -- *why anyone who can build C# programs can maintain LegCRM.*
- The major object classes of LegCRM -- *why anyone who can build C# programs can modify LegCRM*
- Access control in LegCRM -- *why unauthorized PHP code execution is "impossible" in LegCRM*
- Database access in LegCRM -- *why unauthorized SQL execution is "impossible" in LegCRM*
- The client side of LegCRM -- *why unauthorized Javascript execution is "impossible" in LegCRM*

Anyone who can build C# programs can maintain LegCRM.

- How PHP is executed on a website (in general and in Azure)
- How errors are logged in PHP
- Why there are exactly two entry points into the code of LegCRM
- The object structure of LegCRM -- include, require and the autoloader
- Reference documents for PHP

PHP Background

- PHP *"was deliberately designed to resemble C in structure, making it an easy adoption for developers familiar with C, Perl, and similar languages."* (Source: https://www.php.net/manual/en/history.php.php)
- PHP can run on multiple operating systems – Linux, Windows, etc.
- PHP is used by 79.1% of websites in the world followed by ASP.NET at 9.1% (Source: https://w3techs.com/technologies/overview/programming_language)

How PHP is interpreted on a website

- PHP – the executable which interprets PHP scripts -- must be installed
- The server must be configured to direct requests for .php files through the interpreter [see in Azure]
- The PHP interpreter runs code included between start/end tags and just writes the rest to output [see in editor – start as txt>html>php]
- PHP allows but does not require a model of programming in which HTML and PHP are mixed within files.
- ***LegCRM never mixes PHP and HTML.***
- *LegCRM is structured like one big C# project-- one entry point (OK, actually two), class instantiations and a few public function calls*

What a PHP Script has to work with

- Environment global variables
  - GET: the query string
  - POST: posted values
  - Azure Active Identity: User details
- PHP Extensions – Microsoft drivers for sql server access
- Error log
  - PHP errors
  - Programmatic log entries

Why there are exactly two entry points into the code of LegCRM

- Entry points (c# void main)
  - Index.php script (with required load)
  - Ajax.php script
- Global function declarations (can be run but no action)
- All other "scripts" are class declarations (can be run, but no action)

Firing up a GET request (Index.php)

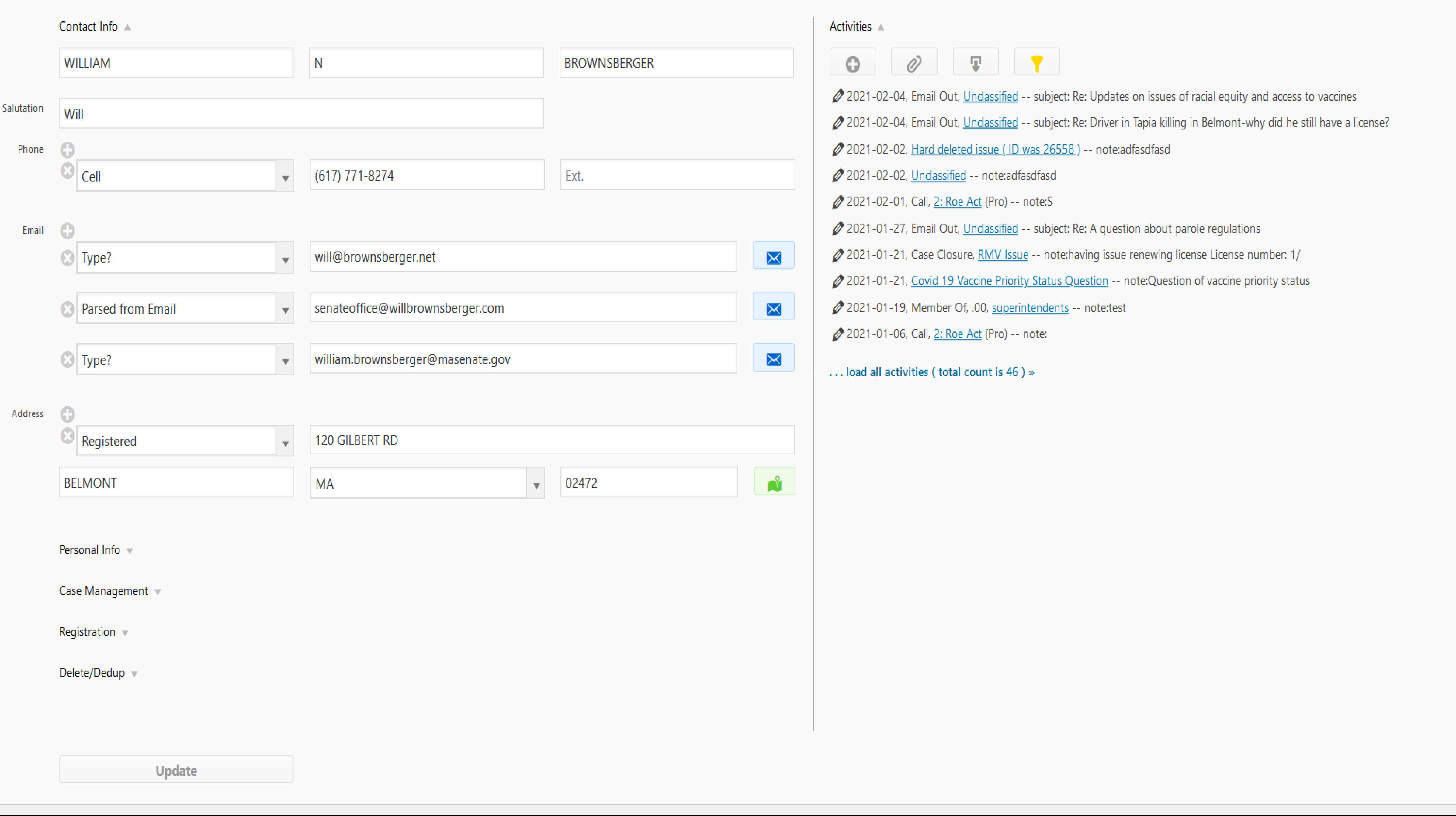
- Require load.php
  - Include config.php
    - Defines constants
    - Registers autoloader for classes not yet in name space [view directory and code]
      - Map class name to directory/file
      - "Require" file
  - Require global functions
  - Instantiate sql server interface object
  - Instantiate user object
  - Instantiate navigation object
- Instantiate header class (html doc and all css and js loads)
- Instantiate body class (apply navigation and invoke entity/list/form/db classes)
- Finish

PHP Resources

- W3Schools – tutorials and examples: https://www.w3schools.com/php/default.asp
- PHP Manual – tutorials, examples, complete reference: https://www.php.net/manual/en/index.php
- Using Visual Studio Code as editor for php https://code.visualstudio.com/docs/languages/php
- Microsoft drivers for PHP access (you will use these only indirectly) https://docs.microsoft.com/en-us/sql/connect/php/microsoft-php-driver-for-sql-server?view=sql-server-ver15

- *Anyone who can build C# programs can modify LegCRM*

- The major object classes of LegCRM -- *why anyone who can build C# programs can modify LegCRM*
  - Controls
  - Entities
  - Forms
  - Lists
- Form generation in LegCRM
- Query generation for update processing in LegCRM



Contact Info ▲

WILLIAM | N | BROWNSBERGER

Salutation: Will

Phone: Cell | (617) 771-8274 | Ext.

Email:
Type? | will@brownsberger.net
Parsed from Email | senateoffice@willbrownsberger.com
Type? | william.brownsberger@masenate.gov

Address: Registered | 120 GILBERT RD
BELMONT | MA | 02472

Personal Info ▼

Case Management ▼

Registration ▼

Delete/Dedup ▼

Update

Activities ▲

- 2021-02-04, Email Out, Unclassified -- subject: Re: Updates on issues of racial equity and access to vaccines
- 2021-02-04, Email Out, Unclassified -- subject: Re: Driver in Tapia killing in Belmont-why did he still have a license?
- 2021-02-02, Hard deleted issue (ID was 26558) -- note:adfasdfasd
- 2021-02-02, Unclassified -- note:adfasdfasd
- 2021-02-01, Call, 2: Roe Act (Pro) -- note:S
- 2021-01-27, Email Out, Unclassified -- subject: Re: A question about parole regulations
- 2021-01-21, Case Closure, RMV Issue -- note:having issue renewing license License number: 1/
- 2021-01-21, Covid 19 Vaccine Priority Status Question -- note:Question of vaccine priority status
- 2021-01-19, Member Of, .00, superintendents -- note:test
- 2021-01-06, Call, 2: Roe Act (Pro) -- note:

. . . load all activities (total count is 46) »

Data structure for constituent entity

```
SELECT
    c.ID – primary key
    first_name, last_name,
    email_address,
    phone_number,
    city
FROM constituent c
    LEFT JOIN email e on e.constituent_id = c.ID
    LEFT JOIN phone p on p.constituent_id = c.ID
    LEFT JOIN address a on a.constituent_id = c.ID
```

Data structure for constituent/activity/issue

```
SELECT
    c.ID – primary key
    first_name, last_name,
    activity_date
    activity_type,
    post_title
FROM constituent c
    LEFT JOIN activity ac on ac.constituent_id = c.ID
INNER JOIN issue i on i.ID = ac.issue
```

Code structure of LegCRM

- Entity classes – constituent, activity, issue, phone, email . . . .
- Control classes – text, select, integer, date, etc.
- Form classes – generate forms from entity collections of controls
- Frame classes – header, body
- DB classes – generate queries from entity collections of controls
- Administrative classes – navigation, access control

Class structure of LegCRM

WIC_Control_Parent (abstract)

- Protected Properties
    - entity_slug, field_slug
    - field_type (text, integer, etc.)
    - field_label
    - HTML characteristics – length, placeholder,
    - Operational characteristics -- transient? required? Dedup?
- On instantiation, loads properties from array passed by entity
- Main Methods
    - Create control – output html (varies in child classes)
    - Sanitize
    - Validate
    - Dupcheck
    - Create search clause
    - Create update clause
- Exercise: View directory and select example – note sanitize; note selectmenu

WIC_Entity_Parent

- Protected Properties
    - Entity, *entity_instance*
    - Fields – array initialized from entity dictionary array in child class
    - Data object array – array of controls initialized from fields
- Some methods iterate over data object array
    - Populate from $_POST or found record or leave blank for blank form
    - Test form values – sanitize, validate, required_check, dup_check
    - Assemble meta_query array for search or update
- Request handler methods use data_object_array methods
    - New blank form
    - Form save update
    - Id_search
- Constructor: routes request to handler
- Exercise: Trace handling of a form_save_update submission

The multivalue recursion concept in LegCRM

- From the standpoint of the constituent entity each *group* of rows (phone, address, email) is a single control of type *multivalue*
- WIC_Control_Multivalue takes a non-scalar value -- an array of entities numbered by instance (numbering may not match row numbering in $_POST);
- WIC_Control_Multivalue responds to requests by iterating over its array and passing the request down to each element (row)
- Multivalue set_value from $_POST extracts row subarray and passes to row entity (why does $_POST have subarrays? Because multivalue row forms are managed by php and js to have two name dimensions) [example, inspect element in form]
- Rows are entities with an extended method set that initializes values from passed sub_array instead of directly from $_POST (for example, WIC_Entity_Email extends WIC_Entity_Multivalue extends WIC_Entity_Parent)
- Exercise – trace recursion of validation of email address in constituent form submission (note extension hook in email control)

WIC_Form_Parent – form generator

- layout_form – takes pointer to data object array from entity and creates form html
  - Message
  - Buttons
  - Groups
    - Main or sidebar
    - Group header and description
    - Controls (with labels)
    - Possible special groups
  - Post form hooks
- Group definitions and control lists in arrays in child classes
- Exercise: Trace constituent blank form layout

WIC_DB_Access: Query Generator

- WIC_DB_Access, parent class built originally to offer single access model for both CRM tables and Wordpress tables – Wordpress data structures accessed through Wordpress query object
- WIC_DB_Access_WIC, child class now supports most entities (see access factory)
- Entity classes iterate over controls to create an array of query clauses (arrays containing essential elements for actual clause generation)
- Access object processes array of query clauses to assemble SQL, executes SQL
- Exercise: Trace assembly of constituent save query; note recursion for multivalue fields

Advanced Query

- Advanced query is an entity – a collection of controls with values; uses standard form generator. See WIC_Entity_Advanced_Search
- Advanced query rows are multivalue controls (like email or address)
- Each advanced query row type has its own entity (like email or address)
- Javascript dynamically swaps rows and fields according to field selection
- Advanced query rows each assemble to a single search term
- Advanced queries are stored as serialized arrays in the search_log
- The search_log is viewable directly using a standard search_log entity
- *Advanced searches have their own query generator, WIC_DB_Access_Advanced_Search*
- Exercise: Trace assembly of query for simple search (from form up to advanced query generator, skip to search_log, show search log functionality).

Lists: WIC_List_Parent

- Standard list classes expect to be passed a pointer to search object, including results.
    - Message
    - Buttons
    - Rows – see WIC_List_Parent
- Row generator
    - Driven by $list_fields array defining field selection and order
    - $list_fields also defines formatting routines
- Results object passed must include all needed fields
- Exercise – trace display of list of duplicate constituents

Anyone who can build C# programs can modify LegCRM – add field consented_to_email_list

- Added to database (with index)
- Added to constituent entity dictionary
- Options defined in constituent option groups
- Added to form group
- Added to constituent list export (but not online list)
- No css or js change required (generic)

Class structure of LegCRM

Access control in LegCRM -- *why unauthorized PHP code execution is "impossible" in LegCRM*

- How Azure Active Directory interfaces with PHP Code for authentication
- How LegCRM manages offices and users
- How LegCRM controls access by authenticated users
- How LegCRM prevents cross-site scripting

It's a simplified standalone product

- Originally installed as Wordpress plugin, running with other plugins in diverse hosting platforms
- Next gen was installation on a dedicated server with no other plugins
- Then Azure with no other plugins, but still in Wordpress
- Now, the app is divorced from any non-Microsoft server-side code other than php itself (do use js libraries: jQuery, jQueryUI, TinyMCE and PLUpload)
- **What you see is what you get – nothing else to evaluate**

How Azure Active Directory interfaces with PHP Code for authentication

- Azure Active Directory prevents access to wpissuesprod except for authenticated users authorized by LIS
- PHP global variable $_SESSION['REMOTE_USER'] is the email of the authenticated user

How LegCRM manages offices and users

- Office maps uniquely to legislator's official email address
- Users belong to an office
- User has access only to the records of the office – all tables and all queries include the owning office
- Access levels defined for each user
  - Assigned only
  - All CRM except Email
  - All CRM including Email
  - Super – create users; only show the office menu to superusers.

Access control in GET transaction to LegCRM

- Single entry point for GET – index.php/load.php (all non-root php files only *declare* functions and classes – no calls or construction)
- Load.php ->WIC_Admin_Setup::user_setup(): Die if not authorized user, otherwise, populate office number and role.
- Either do_page (OR emit_stored_file (email attachments only))
    - Validate elements of query string
    - Check page authorization based on role
    - WIC_Admin_Access::check_security – page action authorization and, depending on capability, whether or not assigned.
- Exercise: Trace access flow through permitted action

Access control in POST transaction to LegCRM

- Single entry point for POST – ajax.php (all non-root php files . . .)
- Do *not* immediately set up user (may test within database)
- choose_ajax_router (validates action requested)
    - some routers set up user and check security as on GET request
        - do_download
        - route_ajax_upload, route_ajax_document_upload, route_ajax_attachment_upload
        - route_ajax_form
    - route_ajax
        - passes user email address directly to database for autocomplete and search box (after nonce check) – validate user and choose office as part of fast keystroke response transaction
        - does set up user and check_security for all other requests
- Exercise: Trace access flow through permitted action for search box

How LegCRM prevents cross-site scripting

- Pages and subforms embed a "nonce" (actually more than one use allowed -- valid through AAD session): hash of session_id, remote_user and possibly attachment_id.
  - Scripts loaded on GET include nonce variable for use by subsequent AJAX requests through javascript
  - Attachments include a specific nonce in URL
  - All forms include a nonce field
  - Note that lists are also forms (comprised of buttons) and include nonce
- Check_security tests nonce for all requests, except on initial page generation. Search/autocomplete which bypasses check_security now directly test nonce.

Access control in LegCRM -- *why unauthorized PHP code execution is "impossible" in LegCRM*

- Azure Active Directory identifies authenticated user to PHP
- LegCRM checks user authorization to use application
- LegCRM specifically checks user authorization for action requested, differentiating by role
- LegCRM checks that every update request is coming from a browser tab that has previously received an authorization token ("nonce") as a form variable, preventing unauthorized use of AAD session cookies.

Database access in LegCRM -- *why unauthorized SQL execution is "impossible" in LegCRM*

- Parametrized SQL execution -- distinguishing code from data in SQL
- Security in $sqlsrv -- the single interface for SQL execution in LegCRM
- Additional comfort -- data sanitization and validation in LegCRM
- Preventing sql injection in the advanced query generator

Parametrized SQL execution -- distinguishing code from data in SQL

```
use legcrm1;
declare @name nvarchar(50) = N'''brownsberger''; select top 1 * from activity';
--declare @name nvarchar(50) = N'brownsberger'; -- select top 1 * from activity';

-- UNSAFE
declare @searchString nvarchar(200) = N'select top 1 * from constituent where last_name = ';
declare @teststring nvarchar(200);
set @teststring = @searchstring + @name;
print @teststring;
EXECUTE    sp_executesql @teststring;
/*
-- SAFE
declare @SAFEsearchString nvarchar(200) = N'select top 1 * from constituent where last_name = @name';
declare @parmdefinition nvarchar(200) = '@name varchar(200)';
EXECUTE sp_executesql @SAFEsearchString, @parmdefinition, @name;
*/
```

Parametrized SQL execution in PHP SQL Server interface

- $name = "brownsberger";
- $sql = "Select * from constituent where last name ='" . $name . "'";
  - **sqlsrv_query**($sql,array());
    - What if $name = "brownsberger'; drop table constituent; --"
- $sql = "Select * from constituent where last name = ? "
  - **sqlsrv_query**($sql, array($last_name));
  - Statement is compiled first, then executed with $last_name as parameter. **sqlsrv_query** includes both prepare and execute steps.
- MSDN reference https://docs.microsoft.com/en-us/sql/connect/php/how-to-perform-parameterized-queries?view=sql-server-ver15

Security in $sqlsrv (WIC_DB_SQLSRV) -- the exclusive interface for SQL execution in LegCRM

- Every call is parametrized query
- Additional precautions
  - Check parameter count = variables count
  - Generate error if any submitted sql includes any statement terminators (;|--|/*|*/|xp_) -- never allow more than one statement
- Exercise – review WIC_DB_SQLSRV->query method (see calling programs too)

Data sanitization and validation in LegCRM (good practice, but little additional SQL protection beyond parametrization)

- Field length constraints
- Input characters limited to word and limited punctuation (except textarea, limited to utf-8 no tags)
- Data type checking
  - Birth year to integer in range
  - Dates
  - Email
  - Floats/ints
  - Selects limited to options
- Additional special sanitization/validation routines by entity – special handling for message html (validate as html and strip unsafe tags in *strip_html_head*)

Preventing sql injection in the advanced query generator

- Query generator does use form input to structure SQL
- But . . .
  - All SQL terms are hard validated against allow lists
  - All data variables are still parametrized
- Exercise: Review construction of simple constituent query
  - Top level substitution of disallowed combine terms
  - Constituent level substitutions and select validation
  - Field and table names from look up

Why unauthorized SQL execution is "impossible" in LegCRM

- All sql execution is parametrized – no mixing of code and data
- Additional comfort -- data is sanitized as much as possible without losing data
- Even the advanced query generator uses only safe strings or parameters

The client side of LegCRM -- *why unauthorized Javascript execution is "impossible" in LegCRM*

- Of course, the user/client can execute any script it wants to via the browser console – that's why the server side has to check every request for authorization (mention user array)
- Goal is to prevent unwitting execution of script – cross-site or throuugh unauthorized script injected into html as a returned value
- Output sanitization on the server side -- preventing the injection of script
  - Input sanitization
    - Default exclusion of tags (strip characters, strip tags, check type or hard validate)
    - Reliance on Graph for sanitization of incoming email html (see https://docs.microsoft.com/en-us/graph/outlook-create-send-messages#reading-messages-with-control-over-the-body-format-returned)
    - Exclusion of bad tags for outgoing email -- strip_html_head (strip whole forbidden elements)
  - Output sanitization – general defensive conversion of special characters to html entities: when in doubt, convert.

Where we are: Comfort issues covered

- PHP in LegCRM is structured like C# -- object oriented, familiar
- LegCRM is built coherently to be easily maintained
- LegCRM is built from the ground up with security in mind
  - All code enclosed except for initial entry points for GET and POST
  - Identity authenticated through Azure
  - Authorization checked via exclusive class::method list
  - SQL execution centralized and parametrized to prevent sql injection
  - All HTML sanitized on both input to prevent java script injection
- Undiscussed: Email handling, mapping, uploader

Email Functionality

- Inbox image
  - Parsed address
  - Matched constituent
  - Matched subject
  - *One touch reply/booking/delete of messages (with undo)*
  - Bulk sweep of multiple messages
  - Email composition; list send; attachments
- Supported by C# pre-processing of messages
- Exercises
  - Review branches of inbox functionality (UI)
  - Review flow of C# parse main
  - Review class list – some extend parent and map to form; others collect related methods

Geographic functionality

- Show districts
- Map constituent sets
- Select geographically
  - Download
  - Email
- Search log as set definition – search terms and shapes saved
- Exercise – see filter_temp_table

Upload functionality

- Remember column head to data field mapping (flexible)
- Validate data as if from form
- Match to existing data using flexible hierarchy
- Option to generate activity records
- Option to create new issues
- Logged: Restartable, reversible
- Slow
- Exercise:
  - Roundtrip a download as an activity add then purge activities
  - See class structure

LegCRM Summary

- Structured code -- easy for a C# programmer to maintain
- Structured code – secure
- Functionality *integrates*
  - Constituent record keeping
  - Email automation *from a legislator's perspective*
  - GIS *from a legislator's perspective*
  - Basic case management
- Directions
  - Outlook integration (note categories and focused management is a start)
  - Cognitive services